# Conditions d'équilibre et gestion d'unités de transport en libre service avec demandes aléatoires

Névine Hafez

# THESE

présentée à

L'UNIVERSITÉ DE METZ
FACULTÉ DES SCIENCES
UFR MATHÉMATIQUES, INFORMATIQUE, MÉCANIQUE



pour obtenir le titre de

DOCTEUR

spécialité

INFORMATIQUE
(mention MATHÉMATIQUES APPLIQUÉES)

par

Névine HAFEZ

Sujet de la thèse

# Conditions d'équilibre et gestion d'unités de transport en libre service avec demandes aléatoires

soutenue le 1er octobre 1999 devant le jury composé de

| | | |
|---|---|---|
| RAPPORTEURS | **M. Pierre Dejax** | *Professeur à l'Ecole des Mines-Nantes* |
| | **M. Bernard Mutel** | *Professeur a l'ENSAIS* |
| EXAMINATEURS | **M. Claude Arnaud** | *Président directeur général d'Eurolum* |
| | **M. Michel Parent** | *Directeur de Recherche a l'INRIA* |
| | **M. François Vernadat** | *Professeur a l'ENIM* |
| DIRECTEUR DE THESE | **M. Jean-Marie Proth** | *Directeur de Recherche à l'INRIA* |

# Remerciements

*Je tiens à témoigner ma reconnaissance à Monsieur le professeur* **Jean-Marie Proth,** *directeur de recherche à l'INRIA Lorraine qui a dirigé ma thèse, pour tous ses conseils précieux, ses critiques constructives et ses encouragements. Qu'il trouve ici l'expression de ma profonde gratitude.*

*Mes sincères remerciements vont à Monsieur* **Michel Parent**, *directeur de recherche à l'INRIA Rocquencourt et directeur de l'action de développement PRAXITÈLE qui m'a accueillie dans son équipe et qui m'a donné toutes les facilités pour effectuer cette thèse dans les meilleures conditions.*

*Je tiens à remercier Monsieur* **François Vernadat,** *professeur à l'ENIM, Monsieur* **Pierre Dejax** *professeur à l'École des Mines de Nantes, Monsieur* **Bernard Mutel** *professeur à l'ENSAIS, Monsieur* **Claude Arnaud,** *Président directeur général d'Eurolum pour avoir accepté de faire partie de mon Jury.*

*Je remercie tous mes collègues de l'INRIA de Rocquencourt et de l'INRIA de Metz pour leur soutien, en particulier* **Fabrice Chauvet** *docteur en Mathématiques appliquées pour son amicale collaboration.*

*J'adresse aussi mes remerciements à ma famille et mes amis qui m'ont encouragée tout au long de cette thèse.*

# Résumé

L'objet de cette thèse est la résolution des problèmes de gestion qui se posent lors de l'utilisation d'un système de voitures électriques mises en libre service en complément des transports en commun. Au bout d'un certain nombre de courses, il se peut qu'un site soit incapable (à cause du nombre limité de places de parking) d'accueillir de nouvelles voitures. La situation inverse peut aussi se produire ; c'est le cas lorsqu'un site est dans l'incapacité de répondre aux demandes des clients et de leur fournir des voitures. Nous proposons deux méthodes afin de redistribuer les voitures présentes dans les sites. La première, appelée *redistribution préventive*, comprend trois étapes. Grâce à la simulation, nous déterminons le moment où une redistribution est nécessaire. Ensuite, par le biais de la programmation linéaire, nous calculons le nombre de voitures qu'il faut déplacer. Finalement, nous établissons les trajets que les camions (qui transportent les voitures) doivent suivre. La seconde méthode, appelée *redistribution permanente*, consiste à faire passer cycliquement un camion dans les différents sites, et à chaque passage de celui-ci, à prendre une décision. Le second problème considéré est la recharge de voitures électriques. Nous calculons un seuil de disponibilité au dessous duquel les voitures ne sont pas autorisées à quitter le site, et ceci afin de limiter les pannes d'énergie pendant les courses. Nous enchaînons avec le problème de la gestion des plots de recharge. Ce problème apparaît lorsque le nombre de plots est inférieur au nombre de voitures présentes. Nous déterminons le moment où le processus de recharge d'une voiture pouvant être interrompu, la voiture peut alors être déplacée et laisser la place à une autre voiture. La dernière partie de ce travail consiste à regrouper les différents logiciels développés précédemment et à fournir un logiciel de simulation intégrant tous les aspects du programme.

*Les mots clefs: Simulation, Transport, Programmation Linéaire, Programmation Dynamique, Heuristiques, Logistique, Voitures en libre service, Voitures électriques*

# Abstract

*This thesis addresses the transportation problem using public electric cars. The management of this new form of transportation means considers two issues: the redistribution of the cars among the stations, and the recharge process. Concerning the redistribution problem, we developed two methods. The first one, based on simulation and linear programming, determines the time of initiating the redistribution process, the number of cars to be moved and, finally, the routes to be taken by the trucks in charge of the process. The second method consists in determining fixed routes for the different trucks. When a truck arrives to a station, a decision has to be made. Concerning the recharging problem, we determine a threshold to overcome the problem of running out of energy during a journey. A car is available to clients only if its level of energy is greater than a threshold that maximizes the probability of a successful journey completion. Moreover, we investigate the problem when the number of recharging systems are less than the number of cars present in one station. We have to decide when the recharging process can be stopped in order to remove the car and recharge the next one. Finally we present the simulator that groups together all the techniques developed previously.*

# Première partie

# Introduction et contribution de la thèse

# *Introduction et contribution de la thèse*

La thèse, qui s'intitule *Conditions d'équilibre et gestion d'unités de transport en libre service avec demandes aléatoires*, étudie la gestion d'un nouveau système de transport public. Il s'agit de voitures électriques mises en libre service en complément des transports en commun. Ce système comprend plusieurs stations ou sites. Chaque site a un nombre limité de places de parking, et chaque emplacement peut être muni ou non d'un système de recharge des batteries. Les voitures sont réparties sur ces différents sites. Un client qui veut utiliser ce système se présente sur un site, prend une voiture disponible, effectue sa course, puis finalement dépose la voiture dans un site de son choix. Celui-ci peut être le même que celui du départ. En règle générale, les clients monopolisent une voiture pour de petites durées parce qu'ils sont facturés en fonction du temps d'utilisation. Ce système n'est donc pas adéquat pour les courses comprenant de longues périodes d'arrêt.

Praxitèle, consortium industriel regroupant la C.G.E.A., Dassault, l'E.D.F., l'I.N.R.E.T.S., l'I.N.R.I.A. et Renault, est un programme de recherche - développement qui tend à démontrer l'intérêt social et économique de ce nouveau système de transport. Une première expérimentation dans la ville de St. Quentin en Yvelines a débuté en octobre 97 pour une période d'essai d'un an. Celle-ci est prolongée jusqu'à fin juin 99. Au début de l'expérimentation, le système comprenait cinq sites et cinquante voitures Renault (Clios) électriques. Dix nouveaux sites ont été mis en service durant l'expérimentation.

DESCRIPTION DÉTAILLÉE DU SYSTÈME.

Le système est constitué de sites qui sont localisés à différents endroits de la ville, comme par exemple, les zones résidentielles, les zones industrielles, les zones proches des gares, etc. Un site comprend un nombre limité de places de parking. Toutes les places de parking ne sont pas munies d'un système de recharge pour des raisons techniques et économiques. Lorsqu'une voiture se gare sur une place de parking munie d'un système de recharge, si le système de gestion le permet, le processus de recharge est déclenché. Lorsqu'un client arrive dans un site et qu'il trouve une voiture disponible - disponible signifie que la voiture est autorisée à quitter un site - il la prend, effectue sa course et ensuite la dépose dans un site de son choix. Un client est facturé en fonction du temps d'utilisation de la voiture. Ce système incite donc les clients à rendre les voitures le plus rapidement possible. Autrement dit, ce système n'est pas adapté aux courses comprenant de longues périodes d'arrêt. L'intérêt d'un tel système est l'utilisation d'une voiture par différents clients dans la même journée, ce qui a comme conséquence d'alléger le trafic routier et aussi de réduire les espaces nécessaires pour les places de parking dans les centres villes. L'intérêt d'utiliser des voitures électriques

est aussi de réduire la pollution dans les villes.

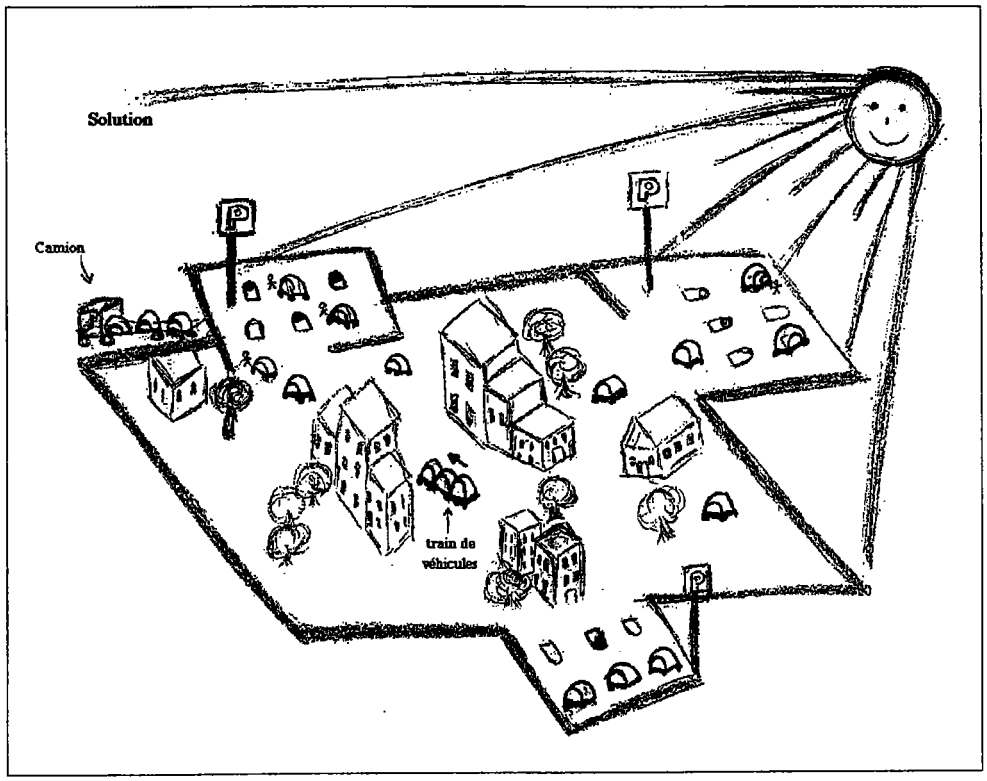



FIG. 1: *Illustration de l'objectif de la thèse qui est le bon fonctionnement du système.*

Les batteries des voitures électriques ont besoin de huit heures pour se recharger complètement. Périodiquement, il est nécessaire de recharger et de décharger complètement les batteries d'une voiture afin d'optimiser leur durée de vie. Une voiture est donc périodiquement immobilisée. Pour des raisons de coût, il n'est pas envisageable d'enlever les batteries et de les remplacer par des batteries chargées lorsqu'une voiture arrive dans un site.

Afin de faciliter l'utilisation d'un tel système, les voitures disponibles sont repérables grâce à une signalisation. Le client présente sa carte d'abonnement au lecteur qui se trouve sur le côté de la voiture. Si la carte d'abonnement est valide, la voiture est attribuée au client et la facturation est mise en route.

Certains problèmes se posent dans la gestion de ce système de transport. Après un certain nombre de courses, en fonction des demandes formulées dans les différents sites, nous pouvons nous trouver dans la situation où un ou plusieurs sites sont incapables, à cause de leur nombre limité de places de parking, d'accueillir de nouvelles voitures. Dans ce cas, nous dirons qu'il y a un 'encombrement' dans ce ou ces sites. De même, nous pouvons nous trouver dans la situation inverse où un site est dans l'incapacité de répondre aux demandes des clients et de leur fournir des voitures. Dans ce cas, nous dirons que ce site est en 'rupture' de voitures. Un site peut être en rupture pour deux raisons différentes : ou bien aucune voiture ne se trouve effectivement dans le site, ou bien les voitures qui se trouvent dans le site ne

sont pas autorisées à le quitter soit parce qu'elles sont insuffisamment chargées, soit parce qu'elles sont en maintenance. Notre rôle est donc de déplacer quelques voitures à vide (sans clients) en prévision des demandes qui seront formulées dans les différents sites, autrement dit, de déplacer des voitures des sites excédentaires vers les sites déficitaires afin que les clients aient la possibilité de se garer ou de trouver une voiture disponible dans n'importe quel site. Il existe différents moyens pour déplacer des voitures d'un site vers un autre, par exemple :

- *Utilisation de jockeys :* le jockey est une personne qui déplace une voiture d'un site vers un autre. Il faut remarquer que, si on utilise ce moyen, la voiture déplacée consomme de l'énergie. L'expérimentation de Saint Quentin en Yvelines a opté pour cette méthode de redistribution des voitures.

- *Utilisation de camions :* les voitures que l'on veut déplacer sont transportées dans un camion. Dans ce cas, il faut prendre en considération les temps de chargement et de déchargement des voitures.

- *Constitution de trains de véhicules :* cette technique est développée au sein de notre équipe de l'I.N.R.I.A. Une voiture peut en suivre une autre sans lien physique grâce à des capteurs et des caméras. Le train peut comprendre au plus 6 voitures. Cette technique élimine le temps de chargement et de déchargement dans chaque site.

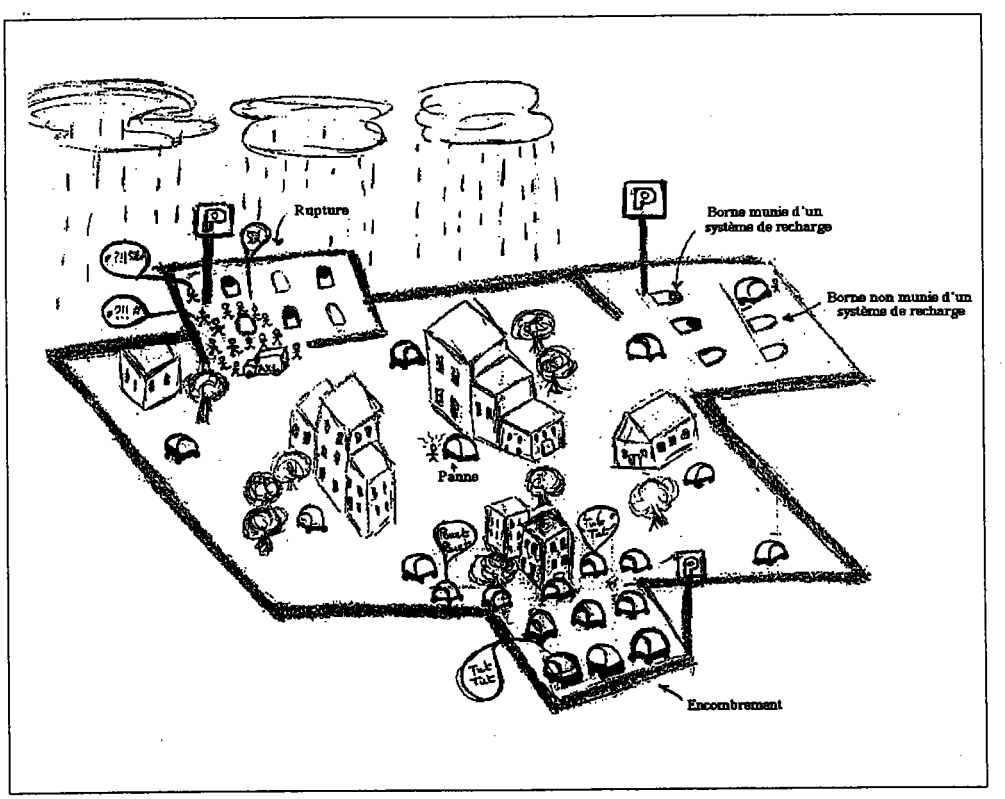



FIG. 2: *Présentation du système lors d'un disfonctionnement.*

Les problèmes qu'un client peut rencontrer lors de l'utilisation d'un tel système sont les suivants :

⋆ Une attente plus ou moins longue dans un site avant qu'une voiture soit disponible.
⋆ Une attente avant de trouver une place disponible dans son site d'arrivée.
⋆ Une panne lors d'une course.

Pour limiter les temps d'attente, nous avons recours à la redistribution des voitures sur les sites. Nous connaissons le nombre de voitures présentes dans les différents sites, les probabilités d'arrivée et de départ des clients dans chacun des sites, les temps moyens pour parcourir la distance entre chaque paire de sites, le nombre de camions chargés de la redistribution des voitures, leur emplacement et leur capacité. Grâce à ces paramètres, nous déterminons le moment du lancement du processus de rééquilibrage, le nombre de voitures à déplacer, et les différentes tournées des camions afin de terminer la redistribution dans les plus brefs délais.

Afin de limiter le nombre de voitures qui tombent en panne d'énergie lors d'une course, nous calculons un niveau de charge appelé *seuil de disponibilité.* Ainsi, les voitures qui ont un niveau de charge inférieur au seuil de disponibilité ne sont pas autorisées à quitter un site. Ce seuil est calculé en fonction d'un critère détaillé dans la thèse.

Différents méthodes de gestion du système de recharge sont considérées dans cette thèse. Comme toutes les places de parking ne sont pas munies d'un plot, nous avons recours au rééquilibrage intra-sites. Des voitures sont permutées afin de permettre à certaines voitures de se recharger dans un même site. Lors de l'implémentation des différentes politiques, nous prenons en considération le nombre de voitures présentes dans le site, le niveau de charge des différentes voitures présentes, le nombre de plots, et le personnel - le personnel peut ne pas être présent en permanence sur les différents sites.

# Contenu de la thèse.

Dans la première partie de la thèse, nous examinons la littérature et rappelons les différentes méthodes déjà développées.

Considérons la seconde partie de la thèse. Dans le premier chapitre, nous montrons comment définir le moment où un processus de rééquilibrage doit être lancé. Le processus de rééquilibrage consiste à déplacer les voitures à vide d'un site excédentaire vers des sites déficitaires. Les termes 'états favorables' et 'états défavorables' sont introduits et nous avons recours à la simulation afin de les déterminer.

Dans le deuxième chapitre, nous calculons le nombre de voitures qu'il faut déplacer, et nous déterminons les sites d'origine et de destination de ces voitures.

Dans le troisième chapitre, nous déterminons les trajets que les camions (qui transportent les voitures) doivent effectuer afin de terminer le rééquilibrage dans les temps. Nous proposons

plusieurs techniques qui reposent sur des heuristiques puisque les problèmes à résoudre sont NP-difficiles.

Finalement nous terminons cette partie en exposant une méthode différente pour remédier à ce problème de redistribution de voitures dans les sites. Cette méthode consiste à faire passer un camion dans les différents sites de manière cyclique. A chaque passage d'un camion dans un site, une décision est prise. Cette décision est soit de déposer des voitures dans le site visité, soit au contraire d'en retirer, ou tout simplement de ne faire.

Passons maintenant à la troisième partie. Dans le premier chapitre, nous examinons le problème de la recharge. Il s'agit de définir à quel niveau de charge une voiture est autorisée à quitter un site, afin de ne pas tomber en panne d'énergie lors d'une course. Nous avons décidé de déterminer un seuil (*seuil de disponibilité*) au dessous duquel les voitures ne seront pas autorisées à quitter le site.

Dans le second chapitre, nous examinons le problème qui apparaît quand le nombre de systèmes de recharge est limité, c'est à dire inférieur au nombre de places de parking. Lorsque le niveau de charge d'une voiture atteint le seuil de disponibilité, le processus de recharge peut être interrompu. La voiture peut alors être déplacée afin de laisser la place munie d'un plot de recharge à une autre voiture. Ce chapitre expose trois politiques qui ont pour rôle de choisir les voitures qui vont être déplacées, soit pour les placer sur des places munies de plots, soit au contraire pour libérer les places munies de plots.

La dernière partie de ce travail consiste essentiellement à regrouper les différents logiciels développés précédemment et de fournir un logiciel de simulation intégrant tous les aspects du programme.

# Table des matières

# Deuxième partie

# Etat de l'art

# Chapitre 1

# *L'optimisation combinatoire*

## Résumé

*Ce chapitre a pour objet d'effectuer une synthèse sur l'optimisation combinatoire et ses différentes méthodes de résolution, en s'appuyant sur les publications disponibles.*

*Mots-clés : Optimisation combinatoire, Méthodes de résolution exacte, Méthodes de résolution approchée.*

## Sommaire

## 1.1 Introduction

Les problèmes d'ordre combinatoire sont des problèmes qui, à partir d'un nombre limité d'éléments, conduisent à une quantité de solutions qui croît exponentiellement [Sak84b]. L'optimisation combinatoire a pour objet de rechercher la solution qui conduit au maximum ou au minimum d'un critère donné. Compte tenu du grand nombre de solutions à considérer, différents outils mathématiques et algorithmiques ont été développés. Ces outils vont être présentés dans les sections qui suivent et nous allons procéder comme le montre la figure 1.1.

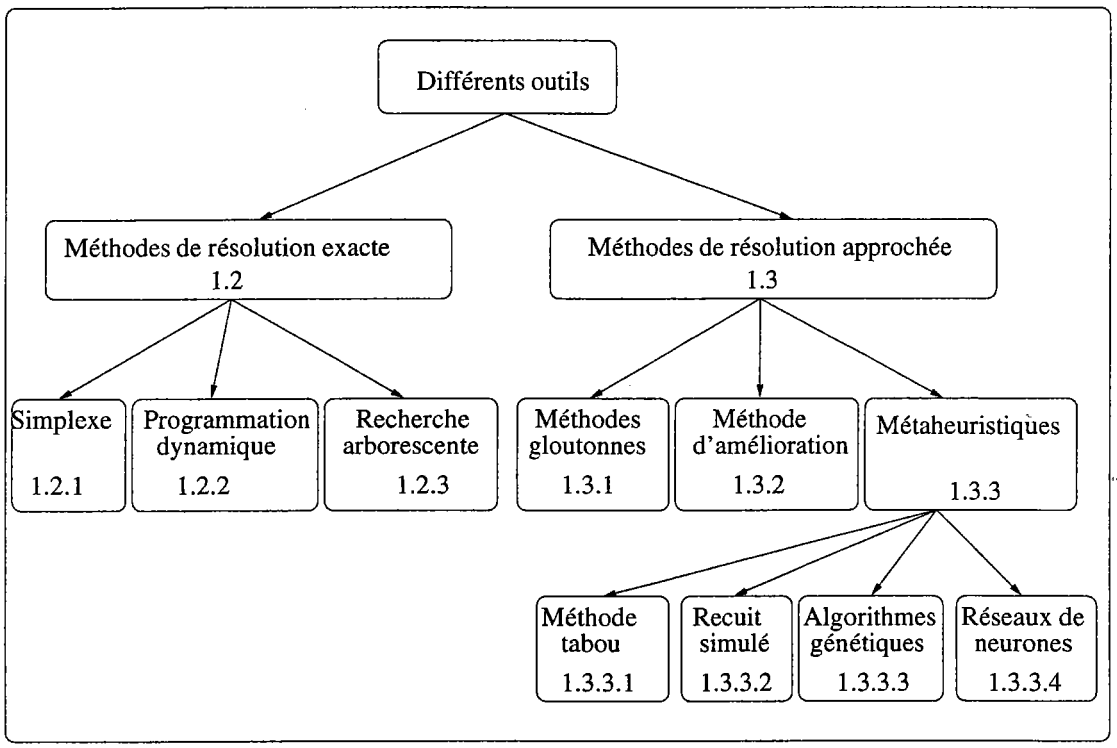



FIG. 1.1: *Classification des sections suivantes*

## 1.2 Méthodes de résolution exacte

L'intérêt de ces méthodes est d'aboutir à la solution optimale, mais l'inconvénient majeur est le temps de calcul qui croît souvent exponentiellement avec la taille du problème.

**Les outils à la base des différentes méthodes sont :**

- la méthode du Simplexe.
- la programmation dynamique.
- la méthode de recherche arborescente, par séparation et évaluation.

Les paragraphes suivants nous donnent un aperçu de ces différentes méthodes.

### 1.2.1 La méthode du Simplexe

Avant d'expliquer cette méthode, nous allons introduire la programmation linéaire. Ce terme, programme linéaire, a été introduit par Dantzig au lendemain de la guerre mondiale en 1947 [Sak84a].
Un problème linéaire se caractérise par :

- une fonction objectif linéaire qu'il faut minimiser (ou maximiser),
- des contraintes linéaires [Thi78].

Voici la formulation générale d'un programme linéaire où l'objectif est de minimiser la fonction (1), sous les contraintes (2) et (3).

$$\text{minimiser } z = cx \quad \ldots\ldots(1)$$
$$Ax \geq b \quad \ldots\ldots(2)$$
$$x \geq 0 \quad \ldots\ldots(3)$$

L'ensemble des solutions réalisables d'un programme linéaire correspond à un polyèdre convexe et il existe toujours un sommet du polyèdre qui coïncide avec une solution optimale, si toutefois une telle solution existe [Sak84b].

Ci-dessous, nous donnons quelques problèmes d'optimisation combinatoire connus qui peuvent être formulés comme des programmes linéaires :

- ⋆ Dans le domaine du *transport*: déterminer le programme d'expédition des marchandises qui minimise le coût total de transport des usines aux dépôts.
- ⋆ Problème de *mélange* : déterminer les quantités de produits qui constituent un certain mélange dans des pourcentages qui restent entre des limites données, tout en minimisant le coût total d'approvisionnement.
- ⋆ Dans le secteur de *production* : déterminer le programme de production optimum d'un fabricant et les quantités de chaque type de produit à fabriquer, en ayant comme but d'obtenir la marge bénéficiaire maximale [Dan66].
- ⋆ Problème du *sac à dos* : effectuer une sélection parmi n produits de différents poids et de différentes valeurs nutritives, de façon à minimiser le poids du sac à dos tout en maximisant les valeurs transportées.

Le Simplexe est une méthode qui a été développée par Dantzig [Dan66]. Elle est basée sur le principe du pivotage. Cet algorithme examine les solutions extrémales du polyèdre convexe et s'arrête quand il atteint l'optimum (s'il existe).

Nous allons expliquer l'algorithme en nous aidant d'un exemple simplifié de programme linéaire [Loo64], qui est le suivant :

$$\text{Maximiser } z = 10x + 15y + 20z.$$
$$10.7x + 5y + 2z \geq 2705$$
$$5.4x + 10y + 4z \geq 2210$$
$$0.7x + y + 2z \geq 445$$

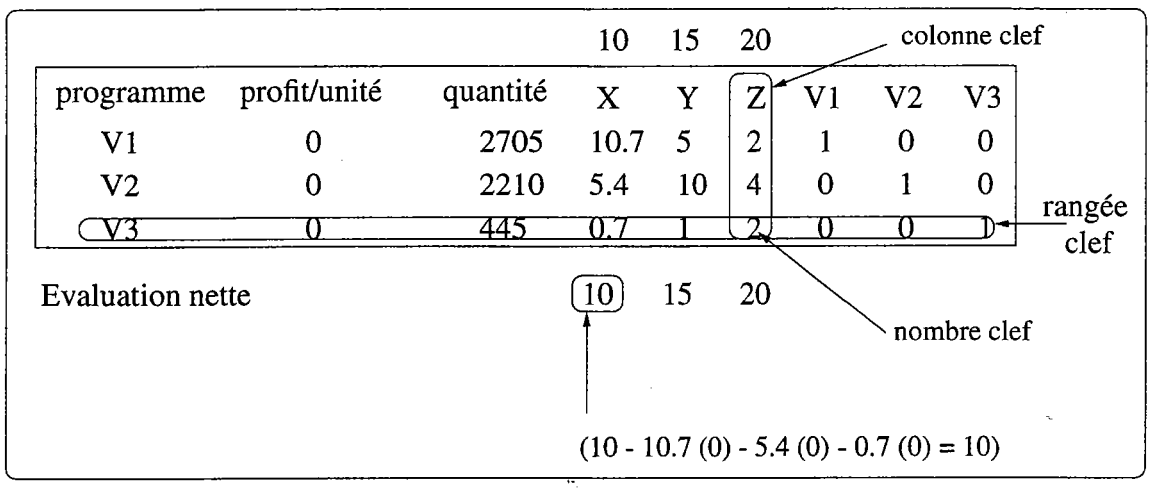



FIG. 1.2: *Premier tableau de l'algorithme du Simplexe*

Les étapes de l'algorithme, lorsque l'objectif est de maximiser une fonction, sont :

1. Transformer les inégalités en égalités et ceci en ajoutant ou en soustrayant des variables d'écart.

2. Calculer l'évaluation nette, (voir tableau 1.2). Ces nombres représentent l'amélioration potentielle de la fonction objectif.

   La présence de nombres positifs dans cette rangée indique que la solution optimale n'a pas été encore obtenue et nous passons à l'étape 3. Si au contraire nous avons uniquement des nombres négatifs ou des zéros, c'est que nous avons obtenu la solution optimale et notre algorithme se termine.

3. Désigner la colonne clef. Celle-ci correspond à la colonne où se situe le plus grand nombre positif dans la rangée de l'évaluation nette.

4. Désigner la rangée clef. Celle-ci correspond à la rangée où se situe le plus petit nombre résultant de la division du nombre se trouvant sous la colonne quantité par le nombre se situant dans la colonne clef de la même rangée.

5. Le nombre clef est celui qui se trouve à l'intersection de la colonne et de la rangée clef.

6. Transformer le tableau par une manipulation de pivotage.

   ⋆ Pour les termes se trouvant dans la rangée clef : $N = \frac{A}{C}$

   où ◇ A = nombre initial se situant à cette position,

   ◇ C = nombre clef.

   ⋆ Pour les termes autres que ceux de la rangée clef :

   Etape 1 : calculer $F = \frac{B}{C}$, où B est le nombre de la même rangée se trouvant dans la colonne clef avant toute modification.

   Etape 2 : $N = A - (R * F)$, où R est le nombre de la même rangée se trouvant dans la colonne clef.

7. Recommencer l'étape 2 [Loo64].

### 1.2.2 La programmation dynamique

Le principe d'optimalité est le principe de base de cette approche. Ce principe est le suivant "dans une séquence optimale de décisions, quelle que soit la première décision prise, les décisions suivantes forment une sous-séquence optimale, compte tenu des résultats de la première décision" [Sak84b]. Autrement dit, afin d'obtenir le résultat final, il faut passer par une séquence d'états intermédiaires, dirigés par une séquence de décisions [PH90].

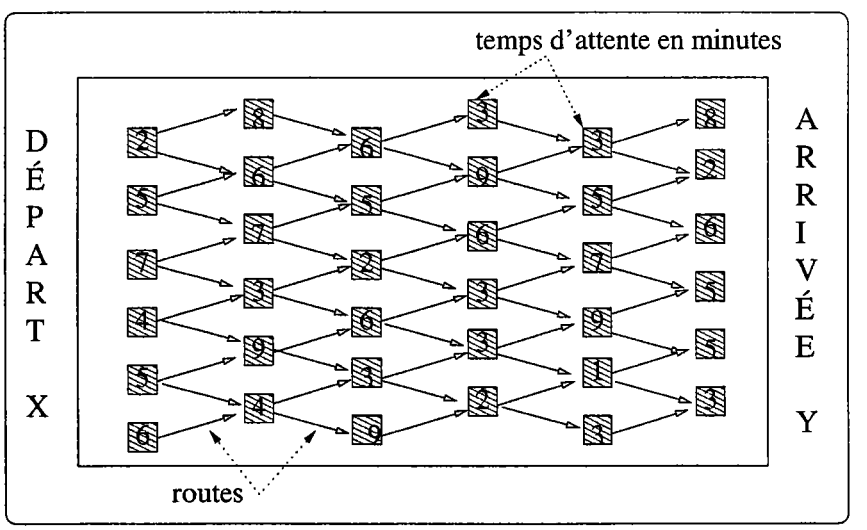



FIG. 1.3: *Présentation d'un problème qui peut être résolu par l'approche dynamique.*

La figure 1.3 qui suit, est une représentation graphique d'un problème qui peut être résolu par l'approche de programmation dynamique. Le but du problème est d'arriver à un point Y en partant d'un point X, ceci en perdant le moins de temps possible dans les intersections de routes. Les temps d'attente sont représentés graphiquement par les chiffres dans les cercles [BHM77].

Les points suivants résument les différentes caractéristiques de cette approche :

- ⋆ Un problème est structuré en différentes étapes. Celles-ci sont traitées séquentiellement comme des problèmes ordinaires d'optimisation, tout en remarquant que chaque étape a un impact sur la décision prise dans l'étape suivante [BHM77]. Comme exemple de structuration en différentes étapes, nous pouvons considérer un problème d'optimisation quelconque qui a lieu sur une année, lorsque celui-ci peut être divisé en douze étapes qui représentent les 12 mois de l'année.

- ⋆ La détermination d'un critère d'évaluation est très importante et difficile à établir. Ce critère doit refléter une information nécessaire afin d'évaluer les conséquences futures d'une décision prise à un instant donné. Le nombre de critères doit être limité afin de diminuer les efforts de calcul, qui sont coûteux [BHM77].

- ⋆ La récursion, essentielle à la programmation dynamique, peut être :

  - ◇ En avant. La première étape considérée est aussi l'étape initiale du problème. On progresse étape par étape jusqu'à l'étape finale.
  - ◇ En arrière. La première étape considérée est l'étape finale du problème, et ensuite on revient en arrière et on évolue étape par étape jusqu'à arriver à l'étape initiale du problème [PH90].

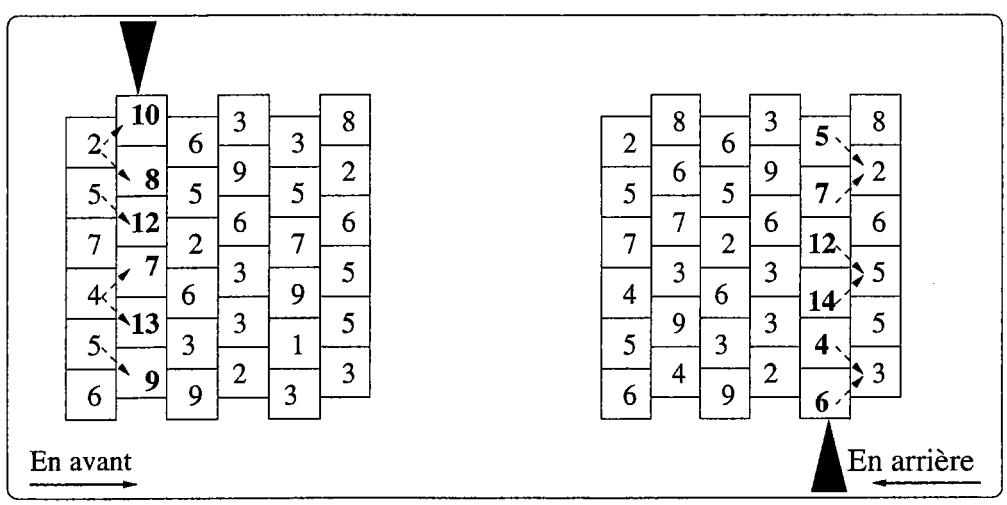



FIG. 1.4: *Démonstration graphique de la première étape de la programmation dynamique (avant et arrière).*

La figure 1.4 qui précède est le calcul de la première étape du problème cité précédemment, une fois en avant et l'autre en arrière.

La difficulté de l'approche dynamique est de reconnaître si un problème peut être structuré en différentes étapes séquentielles ou non, et si le principe d'optimalité peut être appliqué. Comme il n'existe pas de méthode qui permette de vérifier cela, il faut se baser sur l'intuition et l'expérience [Sak84b].

### 1.2.3 La méthode de recherche arborescente

La principale idée de cette méthode est de décomposer un problème, trop important et trop difficile à résoudre directement, en sous-problèmes. Elle est souvent appliquée pour résoudre les problèmes linéaires en nombres entiers, comme par exemple :

$$\begin{aligned}&\text{Maximiser } z = 5x + 8y.\\&5x + 9y \leq 45\\&x + y \leq 6\\&x, y \geq 0 \text{ et x, y sont des nombres entiers.}\end{aligned}$$

Si nous éliminons la contrainte d'intégrité, et si nous résolvons ce problème en appliquant l'algorithme du Simplexe, la solution optimale sera x = 2.25 et y = 3.75. Comme nous voulons des nombres entiers, la première idée est d'arrondir la solution obtenue, mais dans notre cas précis, cette solution arrondie est non admissible. Une solution naïve serait donc d'énumérer toutes les solutions extrémales possibles et d'opter pour une solution entière. Une autre solution serait d'avoir recours à la méthode de recherche arborescente qui cherche une solution optimale possible en faisant uniquement une énumération partielle [Mur76]. Autrement dit, la recherche de la solution optimale s'effectue par une exploration implicite des solutions admissibles entières [BHM77].

Le principe de cette méthode de recherche arborescente constitue un cadre général à l'intérieur duquel les différents algorithmes de séparation et d'évaluation → (BRANCH AND BOUND), qui permettent d'éliminer des branches complètes de l'arborescence et de localiser une solution optimale, sont spécifiés [Sak84b].
Ce principe présente trois caractéristiques principales (nous supposons que nous sommes en présence d'un problème de minimisation) :

1. **Une règle de séparation des solutions.** Cette règle doit respecter la condition suivante :

   $$S_1 \cup S_2 \cup ...S_k \supseteq S$$

   où S est l'espace des solutions et $S_1, \cdots, S_k$ des solutions admissibles ou non.

2. **Une procédure d'évaluation des solutions.** Celle-ci permet d'éviter la construction de l'arborescence complète. Elle comprend deux actions :

   ⋆ *Détermination d'une borne supérieure* **a**. Quand cette borne est proche de la solution optimale, l'algorithme est efficace car il converge rapidement vers la solution optimale. A chaque branchement, une nouvelle borne supérieure peut être calculée en prenant en considération les calculs déjà effectués.

   Les méthodes les plus répandues afin de déterminer une borne supérieure **a** sont :

   - $a = F(x)$ pour x choisi aléatoirement, sachant que F(x) est la fonction objectif que l'on veut minimiser. Afin d'améliorer cette évaluation, on pourrait choisir aléatoirement plusieurs solutions admissibles, calculer les valeurs de

la fonction objectif pour chacune de ces solutions, et finalement opter pour la meilleure solution parmi celles-ci [PH90].

- Une autre méthode consiste à choisir une solution localement optimale [PH90].

⋆ *Calcul d'une borne inférieure* $b_i$ de la fonction objectif pour la meilleure solution de $S_i$. Si $b_i > a$, il est inutile de continuer à explorer cette branche et, par conséquent, ce noeud est tué. Souvent, une borne inférieure est calculée en relaxant les contraintes qui ont défini le sous-ensemble de solutions.

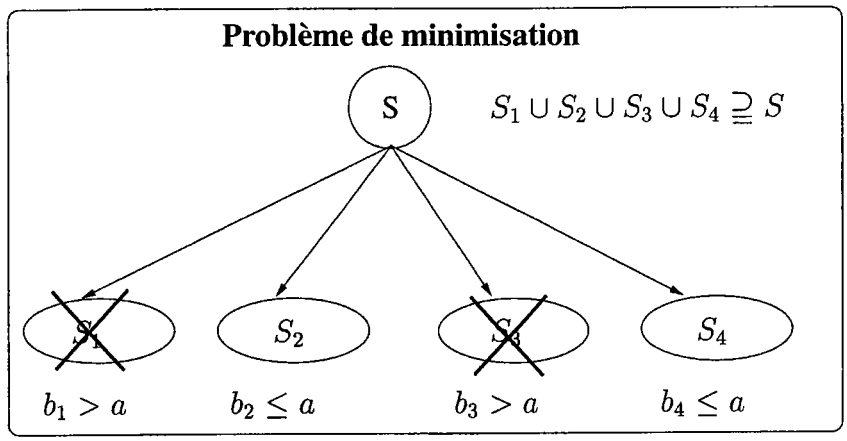



FIG. 1.5: *Illustration de la procédure d'évaluation*

3. **Une stratégie d'exploration des noeuds.** Il faut choisir un noeud auquel seront appliquées les règles de séparation et évaluation. Ici encore, plusieurs approches sont possibles :

   ⋆ *Meilleure limite d'abord* ⟷ BEST BOUND RULE.

   Cette méthode consiste à choisir à chaque étape le sous-ensemble (ou noeud) qui a la plus petite borne inférieure [BG81].

   ⋆ *Premier noeud créé - premier partitionné* ⟷ FIRST CREATED - FIRST PARTITIONED.

   Cette méthode consiste à partitionner les noeuds dans un ordre chronologique [PH90].

   ⋆ *Priorité au noeud le plus récent* ⟷ NEWEST BOUND RULE.

   Cette méthode consiste à choisir parmi les sous-ensembles celui qui a été créé le plus récemment [PH90]. Cette méthode est la plus économe en mémoire.

   ⋆ *Recherche en profondeur* ⟷ IN DEPTH STRATEGY, ou BRANCH AND BOUND WITH BACKTRACKING.

   L'arbre est partagé en sous-ensembles. Un ensemble est choisi, et repartagé, et ainsi de suite jusqu'à obtenir un sous-ensemble contenant un seul élément. Pour déterminer quel sous-ensemble choisir, les règles précédentes, telles que 'meilleure limite d'abord', peuvent être utiles [PH90]. Quand on se trouve en bas

de l'arbre, on doit remonter afin de visiter des noeuds qui n'ont pas été partagés. Le fait de remonter dans l'arbre est le BACKTRACKING [Bak74]. D'après [Rou87], cette méthode est lourde à implémenter et a besoin de beaucoup de mémoire. En fait l'intérêt de cette méthode est qu'elle s'enfonce rapidement dans l'arborescence, et qu'elle détecte très tôt une première solution admissible.

⋆ *Plus mauvais d'abord*

Cette méthode a pour avantage d'écarter rapidement les noeuds qui apporteraient un mauvais résultat. Ainsi, la taille de l'ensemble des noeuds restants est restreinte. Par contre, le calcul de la borne inférieure est la difficulté majeure de cette approche [Mau93].

**Une méthode par séparation et évaluation** est désignée comme **bonne** si elle :

◇ Crée peu de successeurs pour chaque noeud.

◇ Génère des sous problèmes fortement contraints [La87].

Il existe des variantes du BRANCH AND BOUND telles que le BRANCH AND CUT ainsi que le BRANCH AND PRICE.

## 1.3 Méthodes de résolution approchée

Comme les méthodes de résolution exacte sont des problèmes non polynomiaux, nous avons recours aux méthodes de résolution approchée [HKS88]. Contrairement aux méthodes exactes, les méthodes approchées ne garantissent pas l'optimalité des solutions obtenues, mais ont pour but de trouver une bonne solution réalisable en un temps de calcul raisonnable [Mau93]. Ce sont des algorithmes d'approximation ou heuristiques [HKS88]. La qualité d'une heuristique dépend essentiellement du compromis entre la qualité de la solution et le temps de calcul.

### 1.3.1 Les méthodes par construction ou gloutonnes

Ces méthodes consistent à construire progressivement une solution, en se basant sur la fonction économique, sans jamais remettre en question les décisions prises précédemment, autrement dit 'sans retour arrière' [Sak84b].

### 1.3.2 Les méthodes d'amélioration ou d'exploration locale

Ces méthodes consistent à essayer d'améliorer une solution réalisable, en faisant des changements locaux, dans un certain voisinage. L'algorithme s'arrête quand il n'y a plus d'améliorations possibles. Nous obtenons ainsi un optimum local [VDBKS93]. Notons que plus le voisinage considéré est grand, plus la chance d'obtenir une solution optimale est grande mais, par contre, plus les calculs sont importants. L'expérience à prouvé qu'il est beaucoup plus avantageux de ne considérer que des voisinages très restreints quitte à recommencer plusieurs fois la procédure à partir de solutions initiales différentes [Sak84b]. Les solutions

initiales peuvent être obtenues par l'intermédiaire d'heuristiques gloutonnes, mais pour avoir l'assurance de ne pas oublier des solutions ayant des structures particulières, il est préférable de considérer un ensemble de solutions initiales, obtenues par tirage au sort. Un exemple de résultat obtenu pour le problème du voyageur de commerce par Lin et Kerningham est cité ci-dessous. Pour 48 villes différentes, ils ont lancé la procédure 200 fois, avec des solutions initiales différentes. Parmi les 4 solutions réalisables possibles, une solution optimale (l'une d'entre elles) a été trouvée dans 30 % des cas [Sak84b].
Pour le problème du voyageur de commerce, représenté par un graphe où les noeuds représentent les villes et les arcs représentent les routes, plusieurs algorithmes ont été développés tels que :

- 2-échanges ⟷ 2-EXCHANGES.

  Celui-ci consiste à échanger deux arcs du tour initial contre deux autres du voisinage afin d'obtenir un nouveau tour, (voir figure 1.6). Pour calculer s'il y a une amélioration à la suite de cet échange, il suffit de calculer la différence entre les arcs enlevés et les arcs ajoutés [VDBKS93].

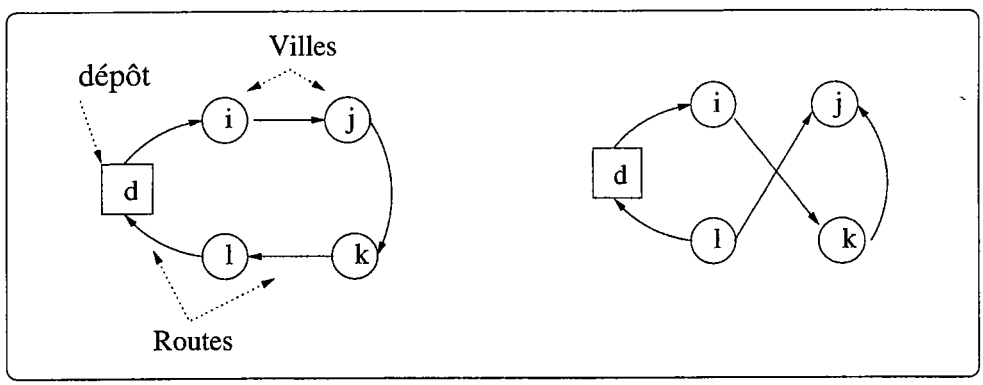



FIG. 1.6: *Illustration de l'algorithme 2-échanges*

- Or-échanges ⟷ OR-EXCHANGE.

  Celui-ci consiste à établir trois modifications dans le tour afin de déplacer un groupe de noeuds. Cet algorithme prend donc en considération les contraintes de précédence [VDBKS93].

- Echange de profondeur variable ⟷ VARIABLE DEPTH EXCHANGE.

  Cet algorithme a été développé par Lin et Kernighan, et son originalité se trouve dans la détermination dynamique du nombre d'arcs à échanger, de sorte qu'à chaque étape une solution peut être trouvée [VDBKS93].

- Recherche du voisin dans un ordre lexicographique ⟷ LEXICOGRAPHIC NEIGHBORHOOD SEARCH STRATEGY.

  Cet algorithme examine tous les échanges possibles d'une façon systématique [VDBKS93].

### 1.3.3 Les métaheuristiques ou recherches globales

Les méthodes métaheuristiques ont pour but de se dégager optima locaux en espérant atteindre l'optimum global.

#### 1.3.3.1 La méthode tabou

Glover est le premier à décrire la méthode tabou [HTdW97]. C'est une méthode itérative qui, partant d'une solution initiale, tente d'atteindre la solution optimale. Elle exécute à chaque pas des mouvements dans un graphe d'espace d'états. En acceptant de détériorer la valeur de la solution courante, elle permet de s'éloigner d'un optimum local. Le risque de cycler est évité en utilisant une liste tabou qui garde en mémoire les derniers mouvements et ceci pendant un nombre limité d'itérations.

Pour un problème de minimisation, l'algorithme général de la méthode tabou est le suivant :

1. Choisir une solution initiale i parmi l'ensemble des solutions S $\rightarrow i \in S$.
2. Initialiser la liste tabou : $L = \phi$
3. Poser $i^* = i$, où $i^*$ est la meilleure solution trouvée jusqu'à maintenant.
4. Initialiser k à zéro, où k est le compteur du nombre d'itérations.
5. Incrémenter k.
6. Générer un sous-ensemble de solutions $V^*$ dans le voisinage de i sans violer la liste tabou (sachant que $\mathcal{S}_i$ est le voisinage de i) : $V^* \subseteq \mathcal{S}_i / L$.
7. Choisir la meilleure solution i parmi les solutions $V^*$ en se basant sur la fonction objectif.
8. Remplacer $i^*$ par i si $\mathcal{F}(i) \leq \mathcal{F}(i^*)$.
9. Ajouter $i$ à la liste Tabou : $L = L + \{i\}$.
10. Retourner à l'étape 4 si aucune condition d'arrêt n'est rencontrée, sinon stopper la procédure.

    Les conditions d'arrêt peuvent être une des conditions listées ci-dessous :

    - $\mathcal{S}_i = \varnothing$.
    - k a dépassé le nombre limite d'itérations.
    - Le nombre d'itérations depuis la dernière amélioration a dépassé un nombre fixé au début de la procédure.

**Description de l'algorithme 1.1:** Tabou.

Quelques astuces pour ne pas oublier certaines solutions consistent à :

⋆ Relancer plusieurs fois la procédure avec une solution initiale différente.

⋆ Pénaliser les solutions les plus fréquemment visitées.

⋆ Relaxer les contraintes en pénalisant leur violation, ce qui est équivalent à réduire les hauteurs des montagnes afin d'atteindre d'autres vallées inexplorées. Dans ces cas là, les solutions visitées peuvent être irréalisables, et il faut renforcer les contraintes graduellement [HTdW97].

#### 1.3.3.2 Le recuit simulé

Le recuit simulé est une méthode issue de la mécanique statistique. L'application à l'optimisation combinatoire a été proposé par Kirkpatrick et al., 1983. En métallurgie, l'obtention d'un cristal parfait se fait grâce à la méthode du recuit. On porte un métal à une température suffisamment élevée pour qu'il soit à l'état liquide. Puis, à partir de cet état, on abaisse la température, et de ce fait les atomes se réorganisent en une nouvelle structure. Une même structure initiale peut donner différentes structures finales selon la façon dont on baisse la température. Pour faire disparaître d'éventuels défauts dans la structure du cristal, on réchauffe doucement le métal, afin que les atomes aient plus de liberté de mouvement. En chauffant suffisamment, on donne assez d'énergie pour qu'ils sortent de l'optimum local. En abaissant à nouveau la température régulièrement, ils pourront atteindre l'optimum global. C'est en s'inspirant de ce procédé que le recuit simulé a été développé. Il s'agit d'un algorithme stochastique permettant une détérioration de la valeur courante selon une certaine probabilité. Cette probabilité dépend de l'importance de la détérioration et du degré d'avancement de la recherche. Cette méthode n'exclut donc pas entièrement le cyclage, mais l'autorise selon une certaine probabilité. Une différence fondamentale apparaît : il n'y a plus ici la notion de température, elle deviendra simplement un paramètre de contrôle lorsqu'on simule un recuit (d'où le nom *Recuit Simulé*, ou SIMULATED ANNEALING) [RR94].
Pour un problème de minimisation, l'algorithme général du recuit simulé est le suivant :

1. Choisir une solution initiale i parmi l'ensemble des solutions S.
2. Choisir les paramètres $T$, $\mu$, et $\xi$.
3. Générer aléatoirement une solution j dans le voisinage de i.
4. Si $\mathcal{F}(j) < \mathcal{F}(i)$
   → alors $i = j$
   → sinon

   Tester si la probabilité de $e^{\frac{\mathcal{F}(i)-\mathcal{F}(j)}{T}} \leq \zeta$,
   où $\zeta \in [0, 1]$ est un nombre tiré aléatoirement.
   Si cette probabilité est effectivement inférieure à $\zeta$
   → alors $i = j$.
   → sinon cette solution est rejetée [ea97]
5. Calculer T tel que $T = T * \mu$
6. Si $T < \xi$
   → alors stopper
   → sinon retourner à l'étape 3.

**Description de l'algorithme 1.2:** Recuit simulé.

Remarques sur l'algorithme du recuit simulé :

- T est notre paramètre de refroidissement. Au cours de la procédure, T va tendre graduellement vers zéro. Ceci veut dire qu'au cours de la procédure, la probabilité d'accepter une détérioration diminue. Cela force le système à converger vers un optimum local [VDBKS93].
- Cet algorithme s'est avéré très efficace pour des problèmes tels que celui du voyageur de commerce, des tournées de véhicules, de la coloration de graphes, etc.

#### 1.3.3.3 Les algorithmes génétiques

Ce type d'algorithme a été proposé par Holland. Il examine à chaque pas une population de solutions. Chaque population (enfants) est dérivée de celle qui la précède (parents) en combinant ses meilleurs éléments et en mettant de côté les moins bons. Des perturbations aléatoires sont appliquées aux enfants de façon à assurer une grande diversité de solutions dans une population : ce sont les mutations [GLP94]. L'inconvénient de ces algorithmes est qu'ils ne garantissent pas la qualité de la solution obtenue.

Description générale de l'algorithme génétique décrit dans [Dav91] :

1. Initialiser une population d'individus par leurs chromosomes qui nous donnent une représentation génétique des individus (du problème). Chaque chromosome est représenté par une série de bits.
2. Evaluer chaque individu de la population.
3. Créer de nouveaux gènes (enfants) à partir des gènes anciens : c'est le CrossOver (voir exemple 1.1 ci-dessous).
4. Supprimer les mauvais individus de la population pour laisser la place aux nouveaux.
5. Evaluer les nouveaux individus et les insérer dans la population.
6. Si une condition d'arrêt est rencontrée
   → alors retourner la meilleure solution trouvée jusqu'à maintenant et stopper la procédure,
   → sinon recommencer à partir de l'étape 3.

**Description de l'algorithme 1.3:** Génétique.

Prenons un exemple simple afin d'illustrer facilement le concept de cet algorithme. Nous avons deux parents dont les chromosomes sont représentés par 5 bits chacun. Un nombre entre 1 et 5 est choisi aléatoirement, disons 2. Pour créer les enfants, les bits des parents sont recopiés sur une longueur aléatoire, et au-delà de celui-ci, les gènes des parents sont intervertis.
Exemple :
Parent 1 = 1 0 1 0 0, et Parent 2 = **0 0 1 1 1**
Comme le nombre aléatoire choisi est 2, les enfants issus seront donc :
Enfant 1 = 1 0 **1 1 1,** et enfant 2 = **0 0** 1 0 0 [GLP94].

**Exemple 1.1:** One Point Crossover pour la création de nouvelles populations

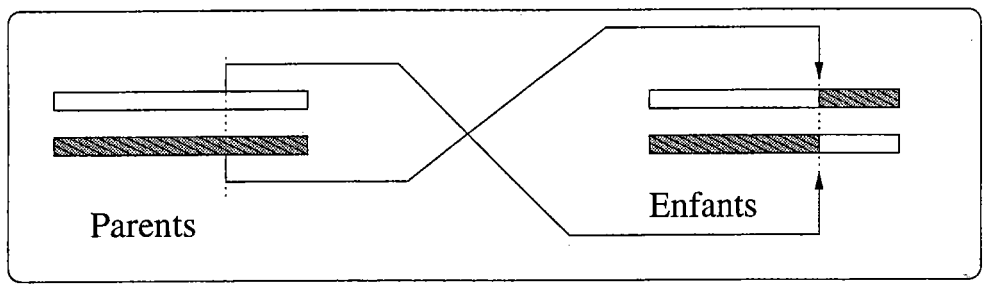



FIG. 1.7: *Illustration de l'algorithme* ONE POINT CROSSOVER.

#### 1.3.3.4 Les réseaux de neurones

Un réseau de neurones est représenté par un graphe orienté et pondéré, les noeuds (les neurones) sont des automates simples qui sont dotés d'un état interne (l'activation) par lequel ils influencent les autres neurones du réseau. Cette activité se propage dans le graphe le long d'arcs pondérés [Jod94].

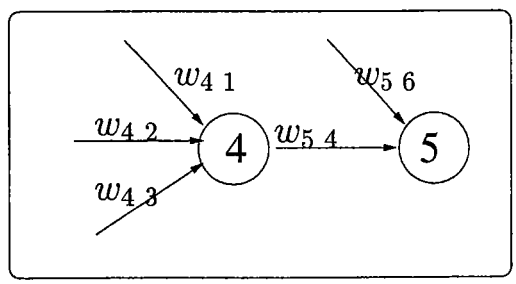



FIG. 1.8: *Un réseau de neurones*

Un modèle général de neurones [Jod94] est essentiellement constitué :

- d'un ensemble d'influences (entrées) provenant d'autres neurones par l'entremise de liens synaptiques pondérés.

- d'une étape interne de calcul où ces influences sont transformées en une valeur unique que l'on nomme l'activation pondérée. Elle est égale à

$$Net_i^t = \sum_{j \in N} w_{ij} a_j^t$$

$N$ est l'ensemble des neurones.

$w_{ij}$ est le poids de l'arc ij.

$a_j^t$ est l'activation du neurone j au moment t.

- d'un seuil d'activation du neurone au dessous duquel le neurone ne répondra pas.

- d'une fonction d'activation (fonction de seuillage) qui permet de déterminer l'activation réelle du neurone. Celle-ci peut être à valeurs continues ou discrètes, ou bien déterministe ou stochastique.

Les domaines d'utilisation sont :

⋆ la reconnaissance des formes et la classification : les données représentent l'information recueillie par un ou plusieurs capteurs. Le but est de reconnaître le ou les objets perçus par le capteur.

⋆ le traitement d'images : il existe des modèles connexionnistes capables de calculer les contours d'objets, la correspondance temporelle d'objets en mouvement et la compression de données.

⋆ la prédiction et le contrôle de processus : ce problème consiste à estimer l'état futur d'un processus à partir d'un historique de son comportement et des variables environnementales attenantes [Jod94].

### 1.3.4 Les critères d'évaluation pour une heuristique

L'analyse du plus mauvais cas peut garantir les performances d'un algorithme. Mais ce sont souvent des résultats pessimistes [Sak84b].

Les principaux critères pour évaluer une heuristique sont :

⋆ La rapidité et la complexité.

⋆ La qualité de la solution.

Les deux critères pour l'évaluer sont :

- La capacité à l'algorithme de fournir une solution réalisable quand celle-ci existe.
- La proximité de la valeur de la fonction objectif par rapport à la valeur optimale.

⋆ La robustesse et la stabilité.

Les résultats devraient varier peu lorsque les données changent légèrement. Cette propriété est désirable car souvent, les problèmes réels présentent des imprécisions dans les données qui sont difficiles à prévoir à l'avance. En calculant le rapport en pourcentage des résultats de cette heuristique à l'optimum (si nous le connaissons) ou bien à une borne inférieure, ou encore au résultat de la meilleure heuristique, nous pouvons comparer plusieurs heuristiques. L'heuristique la plus stable est celle dont le rapport présente le plus petit écart-type sur la série de données.

# Chapitre 2

# *Les problèmes de tournées de véhicules et leurs solutions.*

## Résumé

*Ce chapitre a pour objet d'effectuer une synthèse des problèmes de tournées de véhicules, ainsi que des solutions qui ont été proposées dans la littérature.*

*Mots-clés : Tournées de véhicules* (VRP), le problème du voyageur de commerce (TSP)

## Sommaire

## 2.1 Introduction générale

### 2.1.1 Problème standard

Un VRP (VEHICLE ROUTING PROBLEM) standard se caractérise par :

- ⋆ un nombre de clients qui sont géographiquement dispersés et qui demandent certaines quantités de produits,
- ⋆ un magasin central (ou dépôt),
- ⋆ un nombre limité de véhicules identiques.

Le problème du VRP consiste à déterminer un ensemble de tournées pour ces véhicules, tout en essayant de minimiser le coût total de ces tournées. Celles-ci doivent commencer et finir au dépôt, et chaque client doit être visité par un véhicule [GLP94].

### 2.1.2 Variantes

Quelquefois, de nouvelles données s'ajoutent au problème standard de VRP, comme par exemple :

- *La capacité :* la flotte des véhicules n'est plus homogène mais hétérogène. Les véhicules sont de capacités différentes [Ass88].
- *Les demandes :* dans le problème de collectes et livraisons (PICKUP AND DELIVERY PROBLEM), des marchandises sont ramassées chez un client (source) pour être livrées chez un autre client (destinataire). Si les demandes des clients sont identiques, ce problème est connu sous le nom de DIAL-A-RIDE PROBLEM [DLSS88].

  Deux cas se présentent :

  - ◇ *le cas statique :* les demandes sont connues au moment où les tournées sont construites,
  - ◇ *le cas dynamique :* au moment de la construction des tournées, seule une partie des demandes est connue, d'autres demandes s'ajoutent au fur et à mesure que le temps s'écoule. Alors, on est souvent obligé de modifier quelques tournées de façon à prendre en considération les nouvelles demandes [Psa88].
- *Le temps :* la durée d'une tournée ne doit pas dépasser une certaine valeur limite [Ass88].
- *Les fenêtres de temps* (VRPTW → VEHICLE ROUTING PROBLEM WITH TIME WINDOWS, ou bien PTVFT → PROBLÈMES DE TOURNÉES DE VÉHICULES AVEC FENÊTRES DE TEMPS) : les fenêtres de temps sont des périodes durant lesquelles les véhicules doivent passer chez les différents clients [Bod90]. A chaque client on associe une fenêtre de temps [e, l]. Si le véhicule arrive avant $e$, il est obligé d'attendre

avant d'effectuer sa livraison [DDS92], et s'il arrive après $l$, la livraison est impossible. Cette contrainte oblige à mixer les problèmes de routage et d'ordonnancement. Ce problème est fréquent dans l'industrie. Prenons l'exemple des supermarchés, ceux là ont intérêt à imposer des fenêtres de temps de livraison à leurs fournisseurs afin de gérer correctement leurs marchandises [HDD90].

- *La contrainte véhicule/lieu ou* VEHICLE/SITE DEPENDENCY: souvent, pour des raisons géographiques, seuls quelques types de véhicules de la flotte hétérogène peuvent atteindre certains clients. Dans ce cas, chaque client indique les types de véhicules qui peuvent le desservir [Nag86].

- *Le dépôt :* dans un problème donné, nous pouvons avoir un ou plusieurs dépôts. Dans le cas de plusieurs dépôts, on connaîtra, pour chaque type de véhicules pouvant s'y garer, le nombre maximal admis [Bod90].

- *L'antériorité ou la précédence :* quelquefois, le véhicule de service doit passer par un client avant de passer par un autre. Autrement dit, quelques clients doivent être visités dans un certain ordre (ou pré-ordre) [KS97].

Habituellement, les clients et les routes sont représentés par un graphe orienté, où les noeuds sont les clients, et les arcs sont les routes qui constituent les tournées comme le montre la figure 2.1

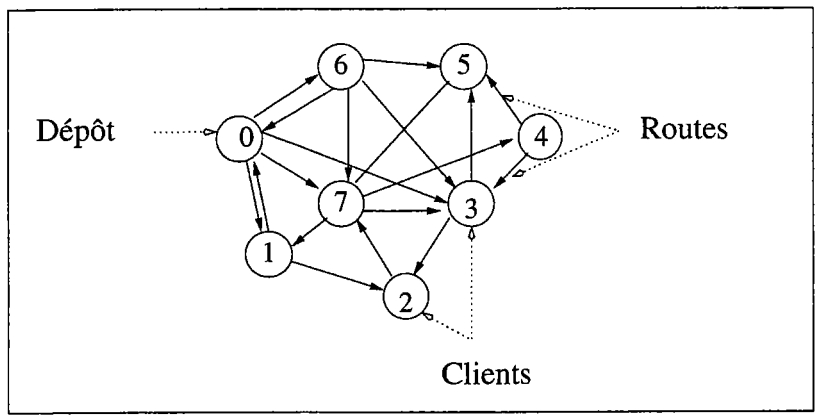



FIG. 2.1: *Un réseau formé de clients et de routes.*

Les tournées sont alors formées d'une série d'arcs ou bien d'une succession de noeuds.

## 2.2 Les solutions proposées

### 2.2.1 La programmation dynamique

Souvent la programmation dynamique est utilisée pour résoudre le problème du plus court chemin. Nous avons un graphe $G = (V, A)$ où $V$ représente les noeuds et $A$ représente l'ensemble des arcs, comme le montre la figure 2.1. $t_{ij}$ est le temps nécessaire pour traverser

l'arc ij, et $d(j)$ est la durée minimale pour atteindre le noeud j en partant d'un noeud initial 0.

Les équations de la programmation dynamique sont les suivantes :

$$d(0) = 0$$
$$d(j) = min_{(i,j)\in A}\{d(i) + t_{ij}\} \text{ pour } j \in V|0$$

Cette approche est souvent utilisée quand le nombre de noeuds est relativement restreint [DLSS88].

#### 2.2.1.1 Algorithme de Christophides, Mingozzi et Toth

Christophides, Mingozzi et Toth ont proposé un algorithme pour résoudre un TSPTW (TRAVELING SALESMAN PROBLEM WITH TIME WINDOWS) qui est le problème du voyageur de commerce avec des fenêtres de temps. Le problème du voyageur de commerce standard (TSP → TRAVELING SALESMAN PROBLEM), consiste à trouver une tournée pour le voyageur de commerce de façon à ce qu'il passe par toutes les villes qu'il doit visiter une seule fois, et puis qu'il revienne à son point de départ. Du point de vue de la théorie des graphes, le TSP consiste à trouver le cycle hamiltonien le plus court [La87].

Les données du problème sont :

Un graphe $G = (V, A)$

où :

$V$ est l'ensemble des noeuds représentant les différentes villes à visiter.

$V = \{0\} \cup N$, où $\{0\}$ est le site de départ et $N = 1..n$, l'ensemble des villes à visiter.

$A$ est l'ensemble des arcs représentant les différentes routes.

A chaque ville $i \in N$, nous associons une fenêtre de temps $[e_i, l_i]$.

Les auteurs cités plus haut ont ramené le problème du TSPTW au problème de la recherche du plus court chemin reliant le noeud source 0 au noeud destination $n + 1$, en visitant tous les sites tout en respectant les fenêtres de temps. Pour cela, ils ont défini des états $(S, j)$ où $S \subset N$ et $j \in S$. De même que $d(S, j)$ est la durée minimale pour parcourir tous les noeuds se trouvant dans $S$ en partant du noeud 0 et en arrivant au noeud $j$. Le but de l'algorithme est d'obtenir la plus petite valeur de $d(N \cup \{n + 1\}, n + 1)$.

Ils ont utilisé les équations de la programmation dynamique suivantes :

$$d(\{0\}, 0) = e_0.$$
$$d(S, j) = min_{i\in S-\{j\},\ (i,j)\in A}\{d(S - \{j\},\ i) + t_{ij}\} \text{ pour } j \in N \cup \{n + 1\}$$

Ensuite, ils redéfinissent $d(S, j)$ de la façon suivante :

Si $d(S, j) < e_j$

alors : $d(S, j) = e_j$.

Si $d(S, j) > l_j$

alors : $d(S, j) = \infty$.

## 2.2.2 La méthode gloutonne

### 2.2.2.1 Algorithme de Clarke et Wright

Cette section décrit un nouvel algorithme développé pour résoudre le problème du TSP. La solution est construite progressivement en se basant sur l'objectif qui consiste à minimiser le coût total de la tournée. Au début, la tournée ne comprend que deux noeuds du graphe : le premier noeud représente la ville de départ (ou le dépôt), et l'autre est un noeud quelconque. Ensuite, à chaque étape, le noeud dont l'ajout dans la tournée entraîne le plus petit coût est fusionné dans la tournée initiale.

Notre graphe G = (V, A) où V = {1..n} est l'ensemble des noeuds (ou sommets) qui représentent les villes (ou les clients), et A est l'ensemble des routes. $c_{ij}$ est le coût lorsque le voyageur de commerce passe par l'arc (i, j).

1. Construire n-1 arcs reliant le dépôt aux n-1 sommets.
2. Calculer les gains $s_{ij}$ :
   Pour i = 2 à n
   Pour j = 2 à n
   Si i $\neq$ j
   $s_{ij} = c_{i1} + c_{1j} - c_{ij}$.
3. Trier les gains dans un ordre décroissant.
4. Si $s_{ij} > 0$
   Tester si, en introduisant la route (i, j) à la place des arcs (i, 1) et (1, j), les contraintes sont respectées.
   Si la réponse est affirmative
   → alors effectuer le changement
   → sinon considérer le gain suivant de la liste établie à l'étape 3.

**Description de l'algorithme 2.1:** Clarke et Wright pour le problème du TSP

## 2.2.3 La méthode en deux phases

### 2.2.3.1 Partitionnement puis construction de la tournée

Le terme anglais est CLUSTER FIRST - ROUTE SECOND. C'est une méthode utilisée pour résoudre le problème standard des tournées de véhicules. Elle est divisée en deux phases : la première consiste à former des sous-ensembles de noeuds du graphe (V), et la seconde phase construit les différentes tournées. La deuxième phase peut avoir recours à des algorithmes connus de résolution du TSP tel que celui décrit dans la section 2.2.1 et d'autres qui seront décrits ultérieurement [BG81].

#### 2.2.3.2 Construction d'une tournée puis partitionnement

Le terme anglais est ROUTE FIRST - CLUSTER SECOND. Cette méthode proposée par Beasley, Haimovich, et Rinnoy Kan sert à résoudre un problème standard de VRP. La méthode commence par construire une tournée passant par tous les noeuds. Cette tournée peut être non admissible. Comme exemple de tournées non admissibles, nous pouvons citer une tournée qui passe plus d'une fois par le même noeud. Ensuite cette tournée est divisée en plusieurs tournées admissibles [BG81].
Pour le partitionnement de la tournée, plusieurs méthodes sont proposées. Ces méthodes consistent à diviser la région planaire, où les différentes villes ou clients sont répartis, en plusieures régions qui se touchent de façon à ce que chaque région ne contienne pas plus de $q$ clients. Ces méthodes (voir figure 2.2) sont :

- Le partitionnement en régions rectangulaires, ⟷ RECTANGULAR REGION PARTITIONING (RRP). Cette méthode est proposée par Karp. Elle consiste à superposer un rectangle sur l'ensemble des noeuds, et ensuite partitionner ce rectangle en plusieurs autres rectangles.

- Le partitionnement en régions polaires, ⟷ POLAR REGION PARTITIONING (PRP). Au lieu du rectangle décrit dans le paragraphe précédent, cette méthode utilise un cercle. Des coupes radiales et des cercles concentriques sont utilisés pour le partitionnement. Si seules les coupes radiales sont utilisées pour le partitionnement, alors cet algorithme se nomme 'partitionnement en régions sectorielles', ⟷ SECTORIAL REGION PARTITIONING (SRP) SCHEME [HKS88].

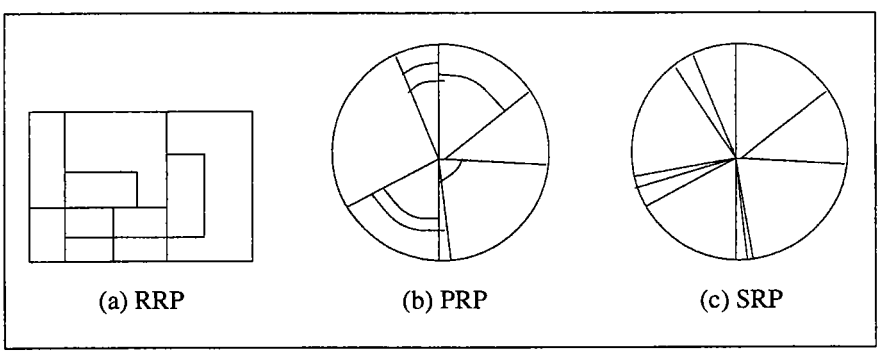



FIG. 2.2: *Les différentes méthodes de partitionnement.*

### 2.2.4 La méthode d'amélioration

#### 2.2.4.1 Les échanges de profondeur variable pour le problème du TSP

La méthode, développée ci-dessous, comporte deux phases :

1. *La phase de construction :* Cette phase, à partir d'une solution quelconque, éventuellement non admissible, tente de construire une tournée admissible en s'aidant d'une

fonction objectif qui pénalise les violations des contraintes.

2. *La phase d'amélioration :* Cette phase a pour rôle de minimiser le coût de la tournée construite dans la phase précédente.

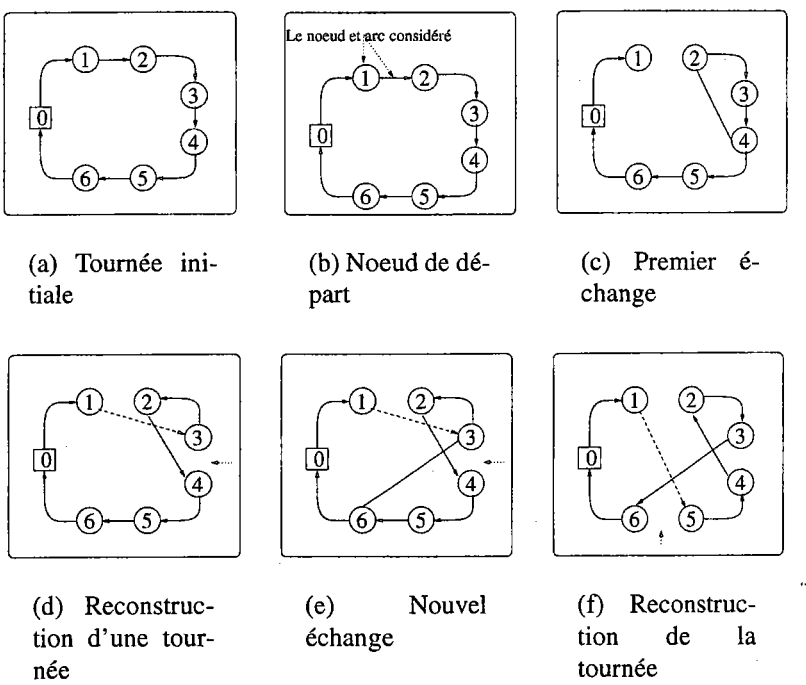
La phase d'amélioration utilise l'algorithme de Lin et Kernighan, appelé l'échange de profondeur variable, que nous détaillons ci-dessous (voir algorithme 2.2). La figure 2.3 [a] représente une tournée admissible issue de la phase de construction. Chaque fois qu'un échange d'arcs est effectué, celui-ci entraine soit une amélioration, soit, au contraire, une détérioration du coût de la tournée. De plus, si $G_s^*$ est le gain quand on change s arcs, et si $G_s^* < 0$, il se peut que $G_{s+1}^*$ soit supérieur à zéro. La traversée d'un arc (i, j) a un coût $c_{ij}$. [VDBKS93]

Les arcs de la tournée initiale sont parcourus dans l'ordre 0, 1, 2, ... , n.

1. Initialiser :
   → $G_0^* = 0$ (gain),
   → s = 0 (compteur).
2. Choisir un noeud $i$ au hasard tel que $i \in \{0, 1, 2, ..., n\}$.
3. Supprimer l'arc $(i,\ i+1)$ si $i < n$ et (i, 0) si $i = n$.
4. Choisir $k^* \neq i$ tel que $g_i = c_{(i,i+1)} - c_{(i+1,k^*)} = c_{(i,i+1)} - Min_{k \neq i}\, c(i+1, k)$.
   Si $g_i > 0$
   → introduire l'arc $(i+1, k^*)$
   → sinon choisir un autre noeud i de départ.
   Si tous les noeuds ont été explorés
   → la procédure s'arrête : la dernière tournée est optimale.
5. Supprimer l'arc $(k^* - 1, k^*)$ si $k^* > 0$
   ou $(n,\ 0)$ si $k^* = 0$.
6. Introduire l'arc $(i, k^* - 1)$.
7. Calculer $G_i^*$ où $G_i^* = g_i + c_{(k^*,k^*-1)} - c_{(k^*-1,i)}$.
8. Inverser le sens de la succession d'arcs appartenant à la tournée initiale et qui joignent $k^* - 1$ à $i + 1$.
9. Recommencer la procédure de 3 à 8 en initialisant $i$ à $k^* - 1$.
10. Renuméroter les sommets dans l'ordre de la nouvelle tournée obtenue.
11. Retour en 2.

**Description de l'algorithme 2.2:** Echanges de profondeur variable pour le problème du TSP

Considérons l'exemple illustré par la figure 2.3. Au départ, on a choisi i = 1. L'arc (1, 2) est donc supprimé. Les calculs ont donné $k^* = 4$. On introduit l'arc (2, 4). On supprime alors l'arc (3, 4). Ensuite, on introduit l'arc (1, 3). On réoriente l'arc (2, 3) qui devient (3, 2). Cela nous mène à la figure 2.3 [d]. Le point 9 de l'algorithme est illustré par les deux dernières figures 2.3 [e et f].




(a) Tournée initiale

(b) Noeud de départ

(c) Premier échange

(d) Reconstruction d'une tournée

(e) Nouvel échange

(f) Reconstruction de la tournée

FIG. 2.3: *Illustration de l'algorithme 'échanges de profondeur variable.'*

## 2.2.5 La méthode tabou

### 2.2.5.1 Algorithme tabou de Williard

Williard est un des premiers à avoir adapté la méthode tabou (voir 1.3.3.1) à la résolution du TSP standard. Une solution initiale est générée au hasard. Cette solution consiste en une tournée qui relie les différents clients au dépôt. Le voisinage de cette solution est l'ensemble des solutions admissibles obtenues grâce à des échanges locaux tel que l'algorithme des 2-échanges (voir section 1.3.2). La nouvelle solution est celle qui, parmi le voisinage ne se trouvant pas dans la liste tabou, mène à la meilleure solution . [GLP97]

### 2.2.5.2 Algorithme tabou de Pureza et França

Ces auteurs utilisent la méthode tabou décrite précédemment (voir section 1.3.3.1) afin de résoudre un problème de VRP standard. La solution initiale est constituée de différentes routes réunissant plusieurs clients. Pour construire l'ensemble des solutions voisines de la

solution initiale, ils autorisent un client à changer de route, de même qu'un échange de clients situés sur différentes routes est possible [GLP94].

#### 2.2.5.3 Algorithme tabou d'Osman

L'algorithme tabou d'Osman utilise la méthode tabou détaillée dans le chapitre précédent (voir section 1.3.3.1) pour résoudre le problème du VRP standard. Afin de définir l'ensemble des solutions voisines de la solution initiale, Osman a recours à l'algorithme de $\lambda$-INTERCHANGE GÉNÉRATION qui est décrit ultérieurement (voir algorithme 2.3), dans lequel il fixe $\lambda$ à 2. Une fois le voisinage construit, Osman propose deux façons de procéder :

1. *Meilleure admissible* ou BEST ADMISSIBLE : tout le voisinage est inspecté et la meilleure solution admissible non taboue est choisie.
2. *Première meilleure admissible* ou FIRST BEST ADMISSIBLE : la première solution admissible qui amène une amélioration à la solution est choisie [GLP94].

On génère plusieurs tournées partielles.

1. Choisir deux tournées $p$, et $q$ parmi les tournées initiales.
2. Choisir un sous-ensemble de clients, $S_p$ et $S_q$ de chacune des tournées sélectionnées en satisfaisant la condition $|S_p| \leq \lambda$ et $|S_q| \leq \lambda$.
3. Tant que la solution est admissible
   $\rightarrow$ échanger le sous-ensemble $S_p$ avec celui de $S_q$.
   *Remarque : Si un de ces sous-ensembles est vide, cette opération s'appelle transfert de clients d'une tournée vers une autre.*

**Description de l'algorithme 2.3:** $\lambda$-INTERCHANGE GÉNÉRATION pour un VRP standard

#### 2.2.5.4 Taburoute de Gendreau, Hertz et Laporte

Taburoute est un algorithme qui utilise la recherche tabou et qui est destiné à résoudre un problème de VRP standard. Pour la création du voisinage d'une solution initiale, celui-ci a recours à l'algorithme GENI.

GENI ou bien GENERALIZED INSERTION est développé par Gendreau, Hertz et Laporte pour résoudre un problème de TSP standard. Il consiste à insérer un client entre ses $p$ plus proches voisins en effectuant une réoptimisation locale de la route. [GLP94]

Description de l'algorithme GENI :

comme données de l'algorithme, nous avons :

a. Une tournée qui est une séquence de noeuds $(n_0, n_1, n_2, ..n_t, n_0)$.

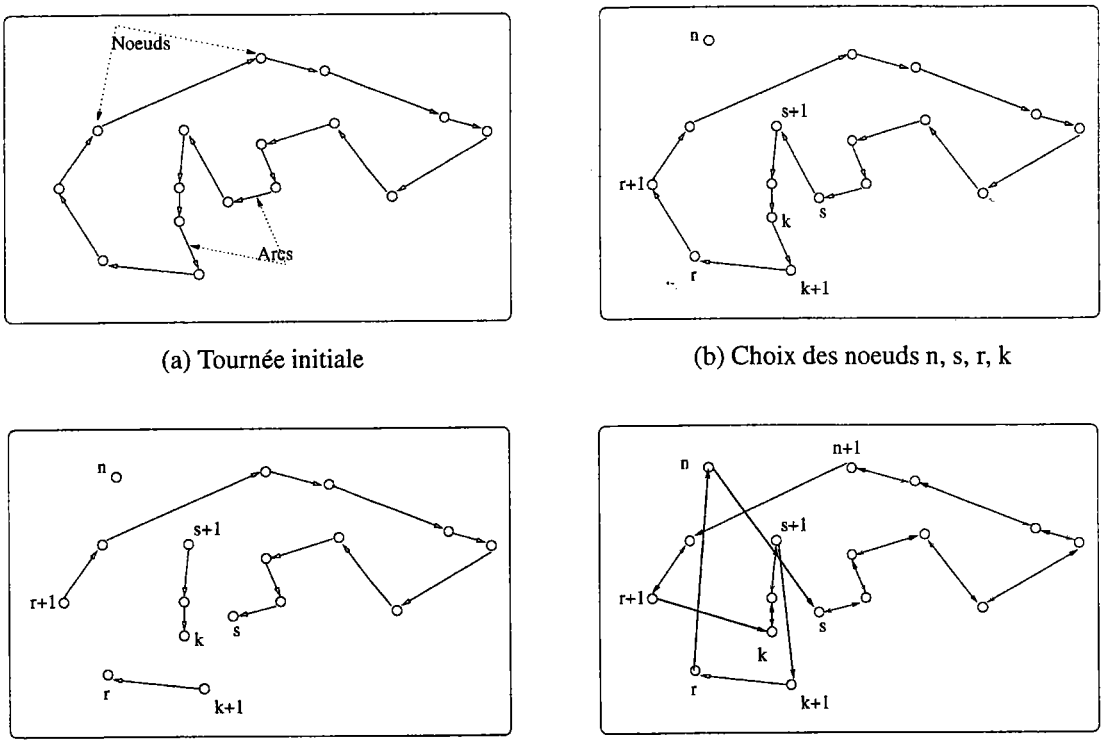
b. Le p-voisinage d'un noeud $n$, qui est désigné par $\mathcal{N}_p(n)$.

c. $P_{ij}$, qui est l'ensemble des noeuds qui se situent entre le noeud i et le noeud j.

Ensuite deux types d'insertions existent :

⋆ Type 1 (voir illustration dans la figure 2.4) :

1. Choisir deux noeuds $n_r$ et $n_s$ parmi le voisinage de $\mathcal{N}_p(n)$.
2. Choisir un noeud $n_k$ tel que $n_k \in \mathcal{N}_p(n+1) \cap P_{sr} | \{n_r, n_s\}$.
3. Supprimer les arcs $(n_r, n_{r+1})$, $(n_s, n_{s+1})$, et $(n_k, n_{k+1})$.
4. Insérer les arcs $(n_r, n)$, $(n, n_s)$, $(n_{r+1}, n_k)$ et $(n_{s+1}, n_{k+1})$.
5. Les arcs se situant entre les noeuds $n_{r+1}$ et $n_s$ sont inversés, de même que les arcs se situant entre les noeuds $n_{s+1}$ et $n_k$.




(a) Tournée initiale

(b) Choix des noeuds n, s, r, k

(c) Suppression des arcs

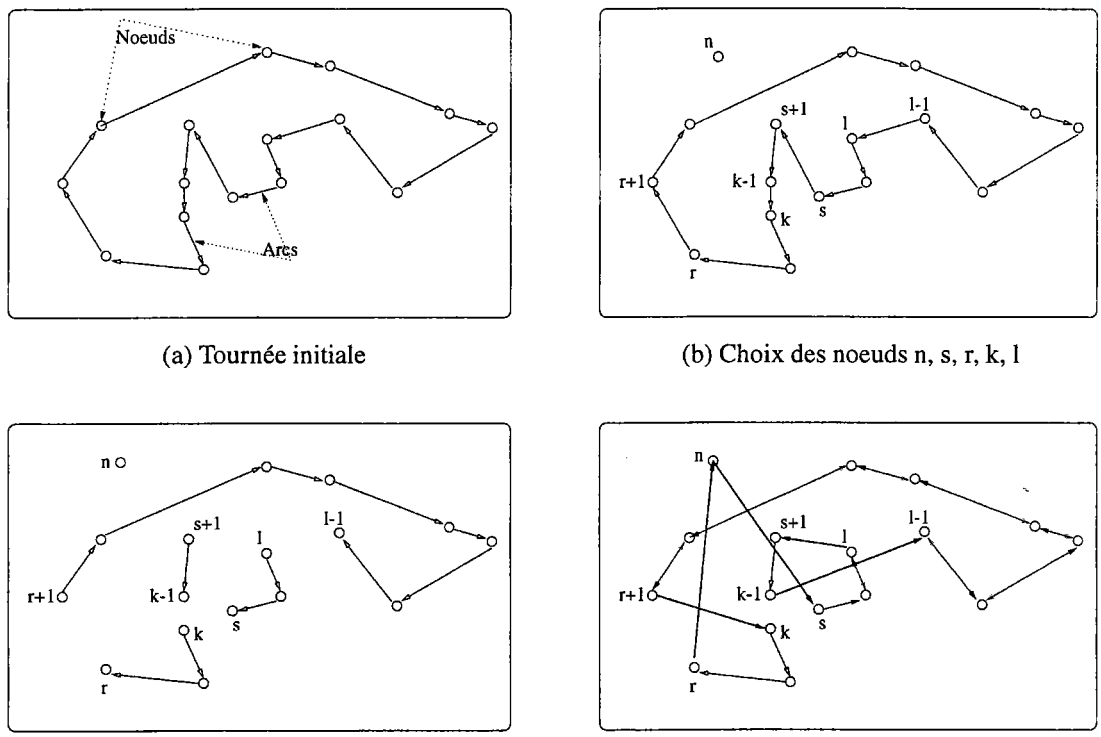
(d) Insertion des nouveaux arcs

FIG. 2.4: *Illustration de l'insertion de type 1.*

⋆ Type 2 (voir illustration dans la figure 2.5) :

1. Choisir deux noeuds $n_r$ et $n_s$ parmi le voisinage de $\mathcal{N}_p(n)$.
2. Choisir un noeud $n_k$ tel que $n_k \in \mathcal{N}_p(n+1) \cap P_{sr} | \{n_s, n_{s+1}\}$.
3. Choisir un noeud $n_l$ tel que $n_l \in \mathcal{N}_p(n_{s+1}) \cap P_{rs} | \{n_r, n_{r+1}\}$.

4. Supprimer les arcs $(n_r, n_{r+1})$, $(n_{l-1}, n_l)$, $(n_s, n_{s+1})$, et $(n_{k-1}, n_k)$.
5. Insérer les arcs $(n_r, n)$, $(n, n_s)$, $(n_{r+1}, n_k)$, $(n_l, n_{s+1})$, et $(n_{k-1}, n_{l-1})$.
6. Les arcs se situant entre les noeuds $n_{r+1}$ et $n_{l-1}$ sont inversés, de même que les arcs se situant entre les noeuds $n_l$ et $n_s$.




(a) Tournée initiale

(b) Choix des noeuds n, s, r, k, l

(c) Suppression des arcs

(d) Insertion des nouveaux arcs

FIG. 2.5: *Illustration de l'insertion de type 2.*

L'algorithme Taburoute n'utilise pas une liste tabou mais plutôt des onglets tabous aléatoires. Quand un sommet migre de la tournée $a$ vers la tournée $b$ à l'itération $t$, sa réinsertion dans la tournée $a$ n'est autorisée qu'au bout de $t + \theta$ itérations. $\theta$ est un nombre aléatoire qui se situe dans l'intervalle prédéfini $[\overline{\theta}, \underline{\theta}]$. De même cet algorithme pénalise les mouvements fréquents afin d'augmenter la probabilité de considérer les autres mouvements possible[GLP97].

#### 2.2.5.5 Algorithme tabou de Taillard

Cet algorithme destiné à résoudre un problème de VRP standard, utilise la méthode tabou (voir section 1.3.3.1). Afin de créer le voisinage de la solution initiale, cet algorithme a recours à l'algorithme d'insertion $\lambda$-INTERCHANGE GÉNÉRATION développé par Osman (voir algorithme 2.3) en égalant $\lambda$ à 1. La nouveauté de l'algorithme de Taillard par rapport à celui d'Osman est la décomposition du problème initial en sous-problèmes. Chaque

sous-problème est résolu individuellement tout en autorisant des déplacements périodiques de noeuds entre les différents sous-problèmes. Cet algorithme est bien adapté à la programmation en parallèle [GLP94].

## 2.2.6 Le recuit simulé

### 2.2.6.1 Algorithme d'Alfa, Heragu et Chen

Cet algorithme utilise la méthode du recuit simulé décrite dans la section 1.3.3.2 afin de résoudre un problème de VRP. Comme donnée supplémentaire, les véhicules ont des capacités limitées.

Les différentes étapes de l'algorithme d'Alfa, Heragu et Chen sont :

1. Construire une solution initiale comprenant une tournée passant par tous les noeuds et ne prenant pas en considération la donnée concernant la capacité des véhicules.
2. Créer le voisinage de la solution initiale :
   a. Supprimer trois arcs aléatoirement de la tournée initiale.
   b. Reconstruire la tournée de huit différentes manières, ce qui constitue le voisinage de la solution initiale.
3. Utiliser le recuit simulé afin de déterminer la meilleure solution parmi le voisinage de la solution initiale.
4. Partitionner cette tournée en plusieurs séries de noeuds de façon à ce que les demandes, dans chaque tournée individuellement, ne dépassent pas Q.

D'après [GLP94], les résultats obtenus grâce à cet algorithme sont mauvais, et non comparables avec ceux obtenus par d'autres algorithmes.

### 2.2.6.2 Algorithme du recuit simulé d'Osman

Osman a proposé un algorithme basé sur le recuit simulé afin de résoudre le problème standard du VRP.

Les étapes de l'algorithme sont :

1. Générer une solution initiale en s'aidant de l'algorithme 2.1 de Clarke et Wright.
2. Créer des solutions voisines de la solution initiale grâce à l'algorithme 2.3 $\lambda$-INTERCHANGE GÉNÉRATION .
3. Utiliser l'algorithme du recuit simulé.

L'auteur de cet algorithme a légèrement modifié le recuit simulé standard décrit dans la section 1.3.3.2. Cette modification consiste à diminuer le paramètre T d'une manière monotone uniquement lorsque la solution courante est modifiée. Lorsque la solution courante est égale à la solution précédente, le paramètre T est diminué de moitié. Pour conclure, cet algorithme n'est pas robuste [GLP94].

## 2.2.7 Les algorithmes Génétiques

### 2.2.7.1 Algorithme du ONE POINT CROSSOVER

Afin de coder les chromosomes qui représentent les tournées dans un problème de TSP ou de VRP, il suffit de lister les noeuds dans l'ordre de visite. Prenons l'exemple ci-dessous :
La tournée = chromosome : 2 3 5 6 4 1 7 8.
La tournée commencera par la visite du noeud 2, sera suivie par la visite du noeud 3 et ainsi de suite jusqu'à arriver au noeud 8. Implicitement, nous comprenons qu'une fois le noeud 8 visité, on recommence la visite du noeud 2 afin d'effectuer une tournée.
L'algorithme du ONE POINT CROSSOVER décrit dans le chapitre précédent (voir algorithme 1.1) est utilisé dans les problèmes de TSP ou de VRP afin de créer des nouvelles populations. La difficulté, lors de son utilisation, est la possibilité de créer des tournées non admissibles telle que celle de l'exemple suivant :
Nous avons deux tournées initiales qui sont :
Tournée 1 : 2 3 5 6 4 1 7 8
Tournée 2 :**1 4 2 3 6 5 8 7**

si le nombre aléatoire est deux, nous obtenons alors :
Nouvelle tournée 1 : 2 3 **2 3 6 5 8 7**
Nouvelle tournée 2 : **1 4** 5 6 4 1 7 8

Comme la nouvelle tournée 1 créée, indique la visite du noeud 3 plus d'une fois, nous constatons ces deux tournées sont non admissibles pour un problème de TSP ou de VRP.

### 2.2.7.2 Algorithme du CYCLE CROSSOVER

Pour la raison citée dans la section précédente, un autre algorithme nommé CYCLE CROSSOVER, développé par Oliver, Smith et Holland en 1987, est présenté ci-dessous. Celui-ci est développé spécialement pour la création de nouvelles populations pour les problèmes de TSP et VRP standards.

1. Sélectionner un sous-ensemble de positions dans les chromosomes des parents. Ce sous-ensemble doit contenir le même sous-ensemble de noeuds.
2. Effectuer un échange entre les sommets des deux parents de ce sous-ensemble.

Exemple :
Les parents, les deux tournées initiales, sont :

```
        *   * *
```

- Tournée 1 : 2 3 5 6 4 1 7 8
- Tournée 2 : **1 4 2 3 6 5 8 7**

Les sous-ensembles de positions 2,4,5 (marquées par les asterisques) renferment le même sous-ensemble de noeuds 3, 4, 6. Après l'échange, nous obtenons :

```
        *   * *
```

- Enfant 1 : 2 **4 5 3 6** 1 7 8
- Enfant 2 : **1** 3 **2** 6 4 **5 8 7**

**Description de l'algorithme 2.4:** CYCLE CROSSOVER

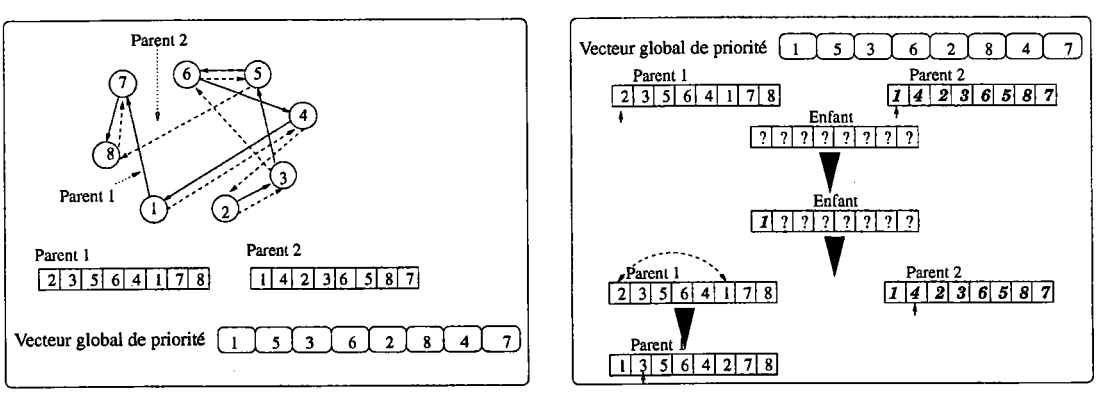
La limite de cet algorithme est qu'il ne peut pas gérer les problèmes de TSP ou VRP avec les contraintes de fenêtres de temps, de capacités ou de distances [GLP94].
Afin de prendre en considération ces contraintes, plusieurs autres algorithmes ont été développés, tel que l'algorithme du MERGE CROSS 1 expliqué ci-dessous.

### 2.2.7.3 Algorithme du MERGE CROSS 1

Cet algorithme a pour but de créer de nouvelles tournées pour les problèmes de TSP et de VRP avec fenêtres de temps. Chaque client a une fenêtre de temps [e, l], et le véhicule est contraint de visiter chaque client suivant la fenêtre de temps qui lui est associée. Une tournée est inadmissible si cette contrainte n'est pas respectée.
Cet algorithme a comme entrée un vecteur global de priorités. Dans ce vecteur, dans un ordre croissant, les noeuds sont rangés en fonction de leur fenêtre de temps.
Avant de rentrer dans les détails de l'algorithme (voir l'algorithme 2.5), nous allons lister les entrées qui sont :
- n noeuds dans le graphe.
- Le vecteur V[x] global de priorité qui est de longeur n.
- Deux vecteurs parents P1[x] et P2[x] qui représentent des tournées de longueur n.
- Un vecteur Enfant E[x] qui est la nouvelle tournée créée.

```
1. Pour i = 1 à n
2.     Si P1[i] est placé avant P2[i] dans V.
3.         Alors
4.             E[i] = P1[i]
5.             Inverser les noeuds dans P2, de façon à ce que le noeud qui était initialement
               égal à E[i] prenne la valeur de P2[i] et P2[i] = E[i].
6.         Inverser les parents et refaire les étapes 4 et 5.
```

**Description de l'algorithme 2.5:** MERGE CROSS 1




(a) Deux tournées parents

(b) Première étape

FIG. 2.6: *Illustration de la première étape de l'algorithme* MERGE CROSS 1.

#### 2.2.7.4 GIDEON

En 1974, Gillet et Miller ont proposé un algorithme génétique pour la méthode en deux phases (voir section 2.2.3) afin de résoudre des problèmes de TSP et de VRP. L'algorithme génétique est utilisé dans la première phase de cette méthode 'partitionnement puis construction de la tournée'. A l'aide du partitionnement en régions rectangulaire ou RECTANGULAR REGION introduit dans la section 2.2.3, les noeuds sont répartis dans plusieurs sous-ensemble qui ont la forme de rectangles. Chaque chromosome, binairement codé, représente un angle de ces régions. En s'aidant de l'algorithme ONE POINT CROSSOVER , et en effectuant plusieurs mutations, ils obtiennent de nouvelles populations qui représentent de nouveaux angles [GLP94].

#### 2.2.7.5 Algorithme de l'opérateur de recombinaison des arcs

L'opérateur décrit dans cette section est développé dans le but de créer de nouvelles populations pour un problème de TSP ou de CTSP qui est le CLUSTERED TRAVELING SALESMAN PROBLEM. Le CTSP est une extension du problème classique de TSP. Les noeuds sont regroupées en sous-ensembles, et le voyageur de commerce doit visiter tous les noeuds d'un sous-ensemble avant de pouvoir visiter un autre sous-ensemble [PG96].

L'opérateur de recombinaison des arcs, EDGE RECOMBINATION OPERATOR, à partir de deux parents et du tableau des arcs, créé une nouvelle tournée.

Le tableau des arcs est construit à partir de deux parents. Afin de l'expliquer nous allons prendre l'exemple suivant. Nous avons deux tournées qui sont :

- Tournée 1 : 1 2 3 4 5 6
- Tournée 2 : 1 5 6 3 2 4

A partir de ces tournées (parents), nous construisons le tableau des arcs comme le montre la figure 2.7. Nous signalons les arcs qui font partie des tournées initiales. Exemple : si l'arc (0, 1) existe dans une des tournées des parents, la case qui se trouve dans l'intersection de 0 et 1 est marquée d'un point noir dans le tableau, et nous l'appelons arc actif [PG96].

Noeuds

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | | ● | | ● | ● | ● |
| 2 | ● | | ● | ● | | |
| 3 | | ● | | ● | | ● |
| 4 | ● | ● | ● | | ● | |
| 5 | ● | | | ● | | ● |
| 6 | ● | | ● | | ● | |

FIG. 2.7: *Le tableau des arcs.*

Les étapes de l'algorithme sont :

1. Choisir un noeud initial $n_0$ aléatoirement. Pour l'exemple, $n_0 = 1$.

2. Supprimer les arcs qui sont incidents à ce noeud $n_0$, ceci en supprimant la colonne du tableau1 dans la figure 2.7.

3. Choisir le noeud suivant de la tournée. Celui-ci ne doit pas être déjà choisi dans une étape antérieure.

   ⋆ Considérer les noeuds qui se trouvent dans la rangée du noeud considéré. Dans l'exemple, les noeuds sont 1, 3, 4 et 5.

   ⋆ Parmi ces noeuds, choisir celui qui referme le moins d'arcs actifs. Dans la figure 2.7, le noeud 3 est donc éliminé.

   ⋆ Si plusieurs noeuds sont équivalents (ont le même nombre d'arcs actifs), le choix se fait aléatoirement entre ces noeuds.

4. Si tous les noeuds n'ont pas été choisis, recommencer à l'étape 2 [PG96].

### 2.2.8 Les réseaux de neurones

Quelquefois, les réseaux de neurones sont utilisés pour résoudre le problème du TSP standard. Chaque neurone du réseau représente une étape pouvant être effectuée par le voyageur de commerce. l'activation du neurone $a_{i,j}$ indique que la ville $i$ sera visitée à l'étape $j$.

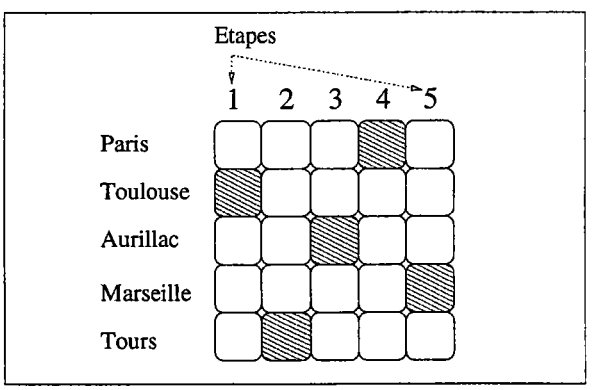



FIG. 2.8: *Réseau indiquant le trajet du voyageur de commerce.*

La connectivité du réseau satisfait quelques contraintes telles que :

1. Un voyageur de commerce visite une ville différente à chaque étape et ne peut la visiter plusieurs fois. Les neurones d'une même rangée ainsi que ceux d'une même colonne doivent alors s'inhiber.

2. Toutes les villes doivent être visitées par le voyageur du commerce. Ceci implique que seul un neurone, par rangée et par colonne, est actif.

3. Quand deux villes se trouvent à petite distance l'une de l'autre, le voyageur de commerce à intérêt à les visiter l'une après l'autre. Pour cela, plus les distances sont courtes plus les poids des arcs les représentant sont importants [Jod94].

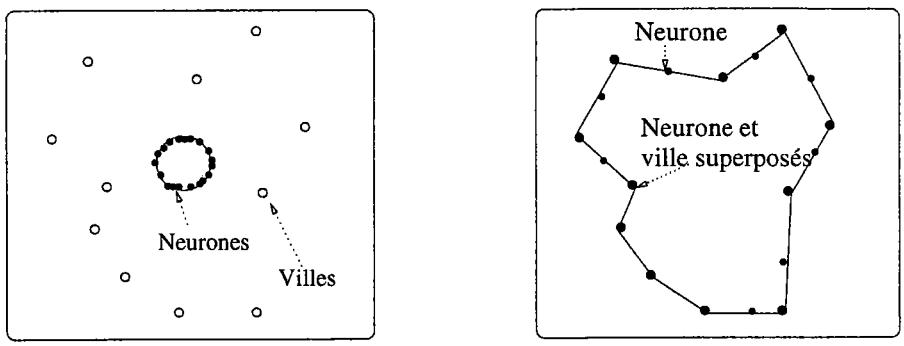
Au début, une activation aléatoire est affectée aux neurones. Ensuite cette activation est propagée jusqu'à ce qu'une stabilité soit atteinte. Celle-ci indique que les neurones satisfont localement, au mieux, les contraintes citées précédemment. Nous obtenons alors une solution admissible, (voir figure 2.8).

La méthode des réseaux de neurones géométriques est une autre méthode destinée à résoudre les problèmes de TSP standard. Le problème général s'énonce comme suit : nous avons une surface où sont répartis nos $N$ sites. Au milieu de celle-ci, les M neurones sont placés en forme de cercle (voir figure 2.9 (a)), en remarquant que $N \leq M$. Le but est de déplacer petit à petit ces neurones vers les différents sites, et ceci, jusqu'à ce que chaque site coïncide avec un neurone. Les neurones en surplus doivent se trouver sur la ligne droite qui rejoint les deux neurones qui l'entourent qui coincident avec des sites (voir figure 2.9 (b)) [PS97].

Il existe deux variantes de cette méthode. La première s'intitule 'les réseaux élastiques' ou ELASTIC NETWORKS et la seconde s'appelle 'Plan auto-organisé' ou SELF-ORGANISED MAP [PS97]. Les paragraphes suivants expliquent ces deux méthodes.

### 2.2.8.1 Réseaux élastiques

Durbin et Willshaw ont développé l'approche réseaux élastiques. A chaque itération, les neurones changent de place. Chaque neurone détermine individuellement sa nouvelle position en fonction de sa position courante, des emplacements des différentes villes, des positions des deux neurones qui lui sont adjacents, et d'un paramètre K qui diminue à chaque itération. Le nom élastique vient du fait que chaque neurone est attiré par les villes qui lui sont proches et, en même temps, par les deux autres neurones qui lui sont adjacents. Les neurones s'attirent mutuellement pour que leur enveloppe soit la plus petite possible [PS97].




(a) Etape intiale (b) Etape finale

FIG. 2.9: *Illustration de la méthode géométrique des réseaux de neurones*

#### 2.2.8.2 Plan auto-organisé

Dans cet algorithme il y a une compétition entre les différents neurones. A chaque itération, une ville est choisie aléatoirement, et le neurone $n$ qui est le plus proche est sélectionné. L'emplacement de ce neurone $n$ change afin de s'approcher de la ville. Les neurones qui sont proches du neurone $n$ se rapprochent aussi de la ville mais plus lentement, et ce rapprochement faiblit au fur et à mesure [PS97].

# Chapitre 3

# *Simulation.*

## Résumé

*Ce chapitre a pour objet d'effectuer un résumé sur la simulation, et les différentes étapes nécessaires pour concevoir un logiciel de simulation dédié à un problème réel.*

*Mots-clés : Simulation*

## Sommaire

## 3.1 Introduction

En général, deux méthodes sont proposées pour évaluer un système (de production, de transport, etc). La première est d'utiliser les outils de l'analyse mathématique, et la seconde est la simulation. L'analyse mathématique est souvent utilisée pour l'analyse de problèmes simples. Pour les problèmes complexes, la simulation est un moyen pratique pour se faire une idée du comportement futur du système. On sait que les problèmes réels sont souvent complexes. Dans ce chapitre, nous nous limitons à la description de la simulation.

Avant de poursuivre cet exposé, nous devons expliquer ce que l'on entend par le mot *évaluer un système*. D'après [PH90], évaluer un système a pour but de répondre à la question : *quelle décision devons nous prendre afin d'optimiser les performances d'un système, et quelles sont ces performances?*.

## 3.2 Evaluation d'un système

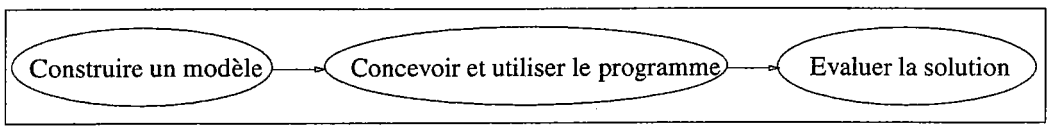



FIG. 3.1: *Les étapes de l'évaluation d'un système.*

Les étapes (voir figure 3.1) pour évaluer un système sont au nombre de trois :

1. **Construire un modèle.**

   La première étape consiste à construire un modèle probabiliste. Autrement dit, cette étape consiste à formaliser les caractéristiques du système, c'est à dire les entités qui le composent, leurs attributs, les contraintes à satisfaire et les critères à évaluer [Ros91].

   Un modèle de simulation comprend :

   - des entités. Pour un problème de production par exemple, les entités sont les machines, les ouvriers, les moyens de transport etc.
   - des attributs. Ces attributs décrivent les différentes caractéristiques de ces entités. Prenons l'exemple des attributs d'une machine de production. Nous pouvons citer sa capacité, sa fréquence de réparation et de pannes, ses périodes de marche etc. [PH90].
   - un état. L'état du système est l'ensemble des valeurs des attributs à l'instant considéré.
   - des critères. Ces critères peuvent être par exemple la productivité, ou les en-cours.
   - des contrôles. Ce sont les diverses décisions prises afin de satisfaire les contraintes externes et internes. Pour le problème de production, ces contrôles sont par exemple les lancements en fabrication ou le choix de la séquence des produits à transformer sur chaque machine [PH90].

2. **Concevoir et utiliser le programme.**

   La seconde étape consiste à construire un programme de simulation, image dynamique du système. Ce programme de simulation peut être développé en utilisant les différents langages évolués standards connus tels que Pascal, C, et bien d'autres. Les développements nécessitent un effort considérable, c'est pourquoi on a souvent recours à des logiciels de simulation. Dans ces logiciels, la phase programmation est restreinte et des facilités graphiques sont fournies. Les premiers logiciels de simulation sont apparus en 1960. Nous pouvons citer SIMULA, SIMSCRIPT, SLAM, SEDRIC etc [PH90]. L'inconvénient de ces logiciels est qu'ils imposent des contraintes qui dépendent de la philosophie qui sous-tend leur conception. En règle générale, plus un logiciel de simulation est "intelligent", c'est à dire moins il requiert d'efforts de la part de l'utilisateur, plus il est contraint.

   Le programme de simulation est lancé avec en entrée des valeurs adéquates des attributs qui représentent l'état initial du système, et le contrôle à appliquer. Les états initiaux testés doivent être variés afin d'intégrer les possibilités les plus importantes auxquelles le système pourrait faire face, sans toutefois être trop nombreux afin de ne pas consommer un temps de calcul considérable [PH90]. Les résultats d'une simulation sont les valeurs prises par les différents critères. Dans un programme de simulation, les nombres aléatoires sont utilisés car ils permettent de simuler les évènements endogènes et exogènes qui influencent le système [Ros91].

3. **Evaluer la solution.**

   La troisième étape consiste à se servir des résultats obtenus par la simulation. Généralement, l'ensemble des résultats est important et ne donne qu'une information partielle sur le comportement du système [PH90]. Les statistiques sont souvent utilisées dans l'évaluation d'une solution. Quelquefois, lorsque les données réelles sont disponibles, le modèle probabiliste déterminé dans la première étape, peut être vérifié par le biais de la simulation. [Ros91]

# Troisième partie

# Redistribution des voitures

# Chapitre 1

# *Etats du système*

## Résumé

*Ce chapitre introduit le problème de la redistribution des voitures. Son but est de déterminer le moment où il est impératif de réagir et de lancer le dispositif de rééquilibrage afin de faire face aux demandes dans les sites.*

*Mots-clés : Simulation.*

## Sommaire

## 1.1 Introduction

Dans cette partie de notre travail, nous étudions la gestion du parc de voitures électriques mises en libre service en complément des transports en commun (une méthode qui intègre la conception et la gestion d'un tel système a été développée, voir [All98], mais du fait de sa complexité, il est difficile de l'appliquer dans le cas réel). Les voitures sont réparties sur les différents sites du système. Un client qui veut utiliser ce nouveau mode de transport, se présente dans un site, prend une voiture disponible, effectue sa course, et finalement dépose la voiture dans un site de son choix. Comme les clients sont facturés en fonction du temps d'utilisation d'une voiture, ils ne la monopolisent que pour de courtes durées. Une voiture est donc utilisée plusieurs fois au cours d'une journée.

Au bout d'un certain nombre de courses, en fonction des demandes formulées dans les différents sites, plusieurs situations défavorables peuvent se présenter. Si nous voulons obtenir une bonne gestion du système nous devons prévoir au mieux ces situations (en particulier à l'aide de l'historique du système) et réagir en temps voulu afin de les éviter. Comme exemple de situations défavorables, nous pouvons citer le cas où un ou plusieurs sites sont incapables d'accueillir de nouvelles voitures à cause du nombre limité de places de parking. Dans ce cas, nous dirons qu'il y'a un *encombrement* dans ce ou ces sites. De même, nous pouvons nous trouver dans la situation inverse où un site est dans l'incapacité de répondre aux demandes des clients en leur fournissant des voitures. Dans ce cas, nous dirons que ce site est en *rupture* de voitures. Ces deux phénomènes sont habituellement simultanés, en ce sens qu'à un site en rupture de voitures correspond généralement un site dans lequel les voitures sont en excès. Afin de résoudre ces deux problèmes, et par conséquent de satisfaire un maximum d'utilisateurs, il nous faut installer un dispositif de rééquilibrage. Ce dispositif consiste à déplacer quelques voitures à vide (sans clients) de certains sites vers d'autres sites, en prévision des demandes qui seront formulées dans les différents sites.

Les questions auxquelles doit répondre un tel dispositif sont les suivantes :

- Quel est le moyen utilisé pour le rééquilibrage?
- Quel est le trajet du moyen utilisé pour le rééquilibrage?
- Ce moyen doit-il avoir une capacité unique?
- Quelles sont les contraintes auxquelles nous devons faire face?
- Quand un dispositif de rééquilibrage doit-il être lancé?
- Combien de voitures doivent être déplacées?
- De quels sites vers quels autres sites les voitures sont-elles déplacées?
- Quels sont les critères sur lesquels nous pouvons nous baser afin de juger le dispositif de rééquilibrage?

Ce chapitre tente de répondre à la question *"Quand un dispositif de rééquilibrage doit-il être lancé?"*

## 1.2 Calcul du nombre de répartitions possibles

Notre système de voitures en libre service est essentiellement constitué d'un nombre fini de sites et d'un ensemble fini de voitures qui représente la flotte du système. Une voiture peut se trouver dans diverses situations :

Elle peut être :

1. *En cours d'utilisation.* Cette voiture est utilisée par un client, et elle se dirige vers un site.
2. *Disponible.* Cette voiture se situe dans un site en attente d'un client éventuel.
3. *En panne.* Cette voiture est inutilisable pour cause de panne.
4. *En cours de chargement.* Cette voiture se situe sur une borne de recharge, et en profite pour recharger ses batteries.

Dans la suite, nous supposons :

- que les temps nécessaires aux réparations sont négligeables. En fait la probabilité de panne d'une voiture électrique est négligeable si elle est bien maintenue, et la maintenance peut se faire de nuit, période durant laquelle le taux d'utilisation est très faible.
- que la charge des voitures est cyclique, et que le nombre de voitures disponibles peut être considéré comme constant. Ceci est nécessaire si nous voulons déconnecter le problème de rééquilibrage du problème de charge.

Dans ce chapitre, nous allons uniquement considérer les voitures disponibles présentes dans les différents sites. Nous appelons *répartition* l'ensemble des nombres de voitures dans les sites. Les deux figures suivantes sont deux répartitions possibles pour le même nombre total de voitures et de sites.

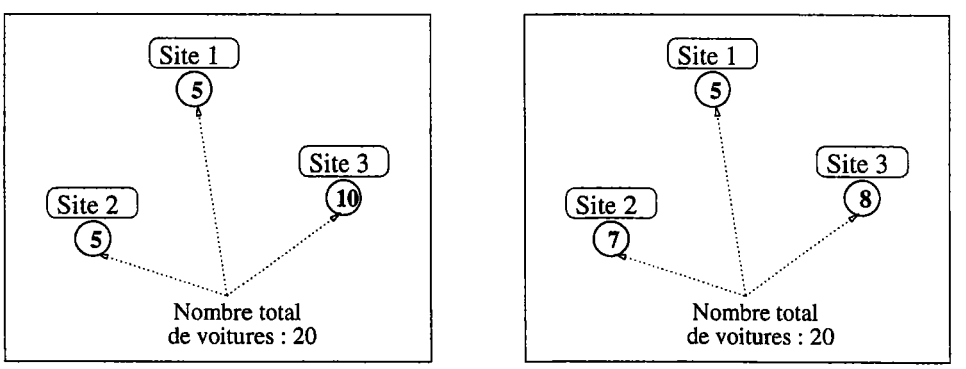



FIG. 1.1: *Deux répartitions pour 3 sites et 20 voitures.*

Le nombre $R(N,K)$ correspond au nombre de répartitions possibles lorsque l'on a $N$ voitures et $K$ sites. Lorsque le système comprend un site unique, une seule répartition existe. En effet, toutes les voitures disponibles se touvent dans le site, et nous obtenons donc la relation 1.1, ci-dessous :

$$R(N,1) = 1 \tag{1.1}$$

Considérons maintenant le cas de $N$ voitures et de $K$ sites. On sélectionne un site, et ensuite on considére $N+1$ types de répartitions caractérisés par le nombre de voitures dans le site sélectionné. Nous appelons 'classe' une telle répartition.

La $(i+1)$ème classe est celle qui correspond à la répartition suivante : mettre $i$ voitures dans le site sélectionné, et $N-i$ voitures dans les autres sites. Ceci est à faire pour $i = 0, 1, 2, ..., N$. D'où la formule :

$$R(N,K) = R(N,K-1) + R(N-1,K-1) + R(N-2,K-1)$$
$$+ ... + R(1,K-1) + R(0,K-1) \tag{1.2}$$

Autrement dit : $R(N,K) = \sum_{i=0}^{N} R(i,K-1)$

Des formules 1.1 et 1.2, nous déduisons le tableau 1.1 qui donne le nombre de répartitions en fonction de $N$ et $K$.

TAB. 1.1: *Nombre de répartitions pour différentes valeurs de $N$ et $K$*

| N↓ K→ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 2 | 1 | 3 | 6 | 10 | 15 | 21 | 28 | 36 | 45 | 55 |
| 3 | 1 | 4 | 10 | 20 | 35 | 56 | 84 | 120 | 165 | 220 |
| 4 | 1 | 5 | 15 | 35 | 70 | 126 | 210 | 330 | 495 | 715 |
| 5 | 1 | 6 | 21 | 56 | 126 | 252 | 462 | 792 | 1287 | 2002 |
| 6 | 1 | 7 | 28 | 84 | 210 | 462 | 924 | 1716 | 3003 | 5005 |
| 7 | 1 | 8 | 36 | 120 | 330 | 792 | 1716 | 3432 | 6435 | 11440 |
| 8 | 1 | 9 | 45 | 165 | 495 | 1287 | 3003 | 6435 | 12870 | 24310 |
| 9 | 1 | 10 | 55 | 220 | 715 | 2002 | 5005 | 11440 | 24310 | 48620 |
| 10 | 1 | 11 | 65 | 285 | 1000 | 3002 | 8007 | 19447 | 43757 | 92377 |

Considérons le tableau 1.1, nous remarquons que chaque nombre est le résultat de l'addition du nombre qui se trouve au dessus avec celui qui se trouve à sa gauche. Ce qui nous conduit à considérer la formule suivante :

$$R(N,K) = R(N-1,K) + R(N,K-1). \tag{1.3}$$

L'explication est simple. En effet, nous observons que la formule 1.2 :

$$R(N,K) = R(N,K-1) + R(N-1,K-1) + R(N-2,K-1) \\ + ... + R(1,K-1) + R(0,K-1)$$

peut se réécrire de la façon suivante :

$$R(N,K) = R(N,K-1) + \sum_{i=0}^{N-1} R(i,K-1)$$

et donc : $R(N,K) = R(N,K-1) + R(N-1,K)$

Les formules 1.2 et 1.3 sont donc équivalentes.

## 1.3 Enumération des différentes répartitions

L'algorithme que nous décrivons dans cette section a pour but de fournir les différentes répartitions possibles pour $K$ sites et $N$ voitures.

Afin d'énumérer ces différentes répartitions, nous avons besoin de connaître le nombre de sites $K$ et le nombre de voitures disponibles $N$ de notre système.

### 1.3.1 Structure des données

La principale stucture de données utilisée dans cet algorithme est un tableau **Sites** de dimension $K$, comme le montre la figure 1.2.

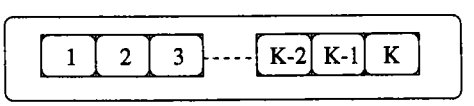



FIG. 1.2: *Tableau* **Sites**.

### 1.3.2 Algorithme *Enumération des répartitions*

```
1. Initialiser les valeurs du tableau Sites à zéro.
2. Faire Sites[1] = N.
3. Initialiser la variable res à zéro ainsi que la variable index.
4. Stop = Faux
5. Répéter tant que Stop = Faux
6.     Imprimer la répartition se trouvant dans le tableau Sites.
7.     Si S[K] = N
8.         alors Stop.
9.     Sinon:
10.        Si index = K
11.            alors:
12.                Décrémenter index.
13.                Répéter tant que Sites[index] = 0
14.                    Décrémenter l'index.
15.                Faire res = Sites[K]
16.                Faire Sites[K] = 0.
17.                Décrémenter Sites[index].
18.                Faire Sites[index+1] = res + 1.
19.                Incrémenter index.
20.            Sinon:
21.                Décrémenter Sites[index].
22.                Incrémenter Sites[index+1].
23.                Incrémenter index.
```

**Description de l'algorithme 1.1:** *Enumération des répartitions*

### 1.3.3 Exemples de répartitions

Dans le tableau 1.2, nous listons les répartitions au fur et à mesure que celles-ci sont calculées afin d'illustrer le fonctionnement de l'algorithme. Les colonnes numérotées de 1 à 5 correspondent au tableau **Sites**, et la colonne *res* indique la valeur de cette variable tout le long du programme. Le principe de l'algorithme consiste à diminuer la valeur d'une case de 1 et à incrémenter la case qui lui est adjacente du côté droit, ceci est répété jusqu'à buter l'extrême droite du tableau **Sites**. Lorsque celle-ci est atteinte, la valeur de la dernière case est gardée dans la variable *res*. Cette valeur est ensuite ajoutée à la case se situant à droite du point de départ de l'étape suivante. Ce nouveau point de départ correspond à la première case qui ne contient pas de zéro en partant de la droite.

TAB. 1.2: *Répartitions lorsque N = 3 et K = 5 calculé avec l'algorithme 1.1*

| site ⟶ | 1 | 2 | 3 | 4 | 5 | *res* | site ⟶ | 1 | 2 | 3 | 4 | 5 | *res* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 3 | 0 | 0 | 0 | 0 | 0 | **19** | 0 | 2 | 0 | 0 | 1 | 2 |

*suite page suivante*

| *suite de la page précédente* | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| site ⟶ | 1 | 2 | 3 | 4 | 5 | *res* | site ⟶ | 1 | 2 | 3 | 4 | 5 | *res* |
| **2** | 2 | 1 | 0 | 0 | 0 | 0 | **20** | 0 | 1 | 2 | 0 | 0 | 1 |
| **3** | 2 | 0 | 1 | 0 | 0 | 0 | **21** | 0 | 1 | 1 | 1 | 0 | 1 |
| **4** | 2 | 0 | 0 | 1 | 0 | 0 | **22** | 0 | 1 | 1 | 0 | 1 | 1 |
| **5** | 2 | 0 | 0 | 0 | 1 | 0 | **23** | 0 | 1 | 0 | 2 | 0 | 1 |
| **6** | 1 | 2 | 0 | 0 | 0 | 1 | **24** | 0 | 1 | 0 | 1 | 1 | 1 |
| **7** | 1 | 1 | 1 | 0 | 0 | 1 | **25** | 0 | 1 | 0 | 0 | 2 | 1 |
| **8** | 1 | 1 | 0 | 1 | 0 | 1 | **26** | 0 | 0 | 3 | 0 | 0 | 2 |
| **9** | 1 | 1 | 0 | 0 | 1 | 1 | **27** | 0 | 0 | 2 | 1 | 0 | 2 |
| **10** | 1 | 0 | 2 | 0 | 0 | 1 | **28** | 0 | 0 | 2 | 0 | 1 | 2 |
| **11** | 1 | 0 | 1 | 1 | 0 | 1 | **29** | 0 | 0 | 1 | 2 | 0 | 1 |
| **12** | 1 | 0 | 1 | 0 | 1 | 1 | **30** | 0 | 0 | 1 | 1 | 1 | 1 |
| **13** | 1 | 0 | 0 | 2 | 0 | 1 | **31** | 0 | 0 | 1 | 0 | 2 | 1 |
| **14** | 1 | 0 | 0 | 1 | 1 | 1 | **32** | 0 | 0 | 0 | 3 | 0 | 2 |
| **15** | 1 | 0 | 0 | 0 | 2 | 1 | **33** | 0 | 0 | 0 | 2 | 1 | 2 |
| **16** | 0 | 3 | 0 | 0 | 0 | 2 | **34** | 0 | 0 | 0 | 1 | 2 | 1 |
| **17** | 0 | 2 | 1 | 0 | 0 | 2 | **35** | 0 | 0 | 0 | 0 | 3 | 2 |
| **18** | 0 | 2 | 0 | 1 | 0 | 2 | | | | | | | |

*Remarque :*
La première répartition du tableau 1.2 est déterminée à l'étape 2 de l'algorithme 1.1. Les répartitions 2 à 5 sont calculées grâce aux étapes 21 à 23, quant à la sixième répartition, elle est calculée à l'aide des étapes 12 à 19. La suite des calculs reprend la même séquence à partir de l'étape 2.

## 1.4 Détermination des différents états du système

Dans les sections précédentes, nous avons constaté que le nombre de répartitions qui peuvent exister pour un nombre $K$ de sites et $N$ de voitures est important. Parmi toutes les répartitions possibles, nous considérons deux catégories particulières qui sont les états favorables et les états défavorables. L'intérêt de cette distinction est de déterminer l'instant où le dispositif de rééquilibrage doit être mis en oeuvre. Il nous faut mentionner que plusieurs répartitions peuvent être classées *état favorable*, de même que nous pouvons avoir plusieurs états défavorables et ceci, bien sûr, pour les mêmes valeurs de $N$ et $K$. Notre but est de réagir lorsque la répartition de voitures est reconnue comme étant dans un état défavorable, et ainsi lancer le processus de rééquilibrage qui indiquera les mouvements à effectuer afin d'atteindre une répartition répertoriée *favorable*.

### 1.4.1 Le simulateur

Afin d'évaluer chaque répartition, nous avons recours à la simulation.

Nous listons ci-dessous les différents paramètres qui caractérisent le système.

- Le nombre de sites $K$ ainsi que le nombre $C_i$ de places de parking dans chacun des sites $i$, $i = 1, 2, \cdots, K$.

- Le nombre total $N$ de voitures en service ainsi que leur état (en cours d'utilisation, disponible, en panne ou en cours de chargement).
- La durée de chaque période élémentaire $\Delta$, ainsi que le nombre de périodes élémentaires qui constituent l'horizon de l'étude.
- Les probabilités de déplacement entre tout couple de sites pour chaque période élémentaire. On note $p_t^c(i,j)$ la probabilité que l'on ait, pendant la période $t$ à $t+\Delta t$, $c$ clients qui demandent une voiture afin d'effectuer un trajet qui débute au site $i$ et qui se termine au site $j$. Ces deux sites peuvent être identiques.
- Les durées des déplacements entre tout couple de sites en connaissant la période élémentaire du départ. On note $D_t(i,j)$ la durée nécessaire pour effectuer un trajet entre le site $i$ et le site $j$, et ceci en commençant la course au début de la période élémentaire $t$. Cette quantité est exprimée en nombre de périodes élémentaires.

A chaque période élémentaire, les différents évènements à considérer sont :

- L'arrivée d'une voiture dans un site.
- L'arrivée d'une demande dans un site.
- Le départ d'une voiture d'un site.

Lorsqu'un tel évènement se produit, plusieurs mises à jour sont établies à chaque période élémentaire, à savoir :

- L'emplacement des différentes voitures.
- Le nombre de voitures présentes dans chaque site.
- Les files d'attente des demandes dans chaque site.

Les sections suivantes ont pour objets de définir chacun des états favorables et défavorables, de présenter les méthodes utilisées pour les déterminer, et enfin de terminer avec un exemple de simulation.

## 1.4.2 Etat favorable

Un état favorable est une répartition de voitures qui, à un moment donné, garanti le bon fonctionnement du système, autrement dit, sans encombrement et sans rupture dans les différents sites, et ceci sur un horizon aussi important que possible.

### 1.4.2.1 Définition

Nous nous donnons un seuil de réussite $\mu$ (ex : 95%) et un horizon $h$.

> "*Un état est dit favorable pour le seuil de réussite $\mu$ et l'horizon $h$ si la probabilité de bon fonctionnement (sans rupture et sans encombrement) avant cet horizon $h$ est supérieure à $\mu$, et ceci pour tous les sites.*"

### 1.4.2.2 Description de l'algorithme

Afin de déterminer un état favorable, nous effectuons $Z$ simulations. Nous commençons chaque simulation $z$ avec une répartition initiale où aucune voiture ne figure dans les sites. Les voitures se situent dans une file d'attente que l'on nomme *Stock* et sont mises à la disposition des clients quand ils émettent des demandes. A chaque période élémentaire $\Delta$, les demandes dans chaque site sont générées aléatoirement en fonction des probabilités données. Si le nombre de voitures présentes dans le site $i$ (ces voitures sont arrivées dans le site $i$ lors des précédentes périodes) est insuffisant pour répondre à toutes les demandes formulées, des voitures en provenance du *Stock* sont ajoutées dans ce site au fur et à mesure des besoins. A la fin de chaque simulation, nous comptabilisons le nombre total $m_{(i,z)}$ de voitures ajoutées dans le site $i$ tout au long de la période $[t, t+h]$ afin d'éviter les ruptures. Autrement dit, $m_{(i,z)}$ est le nombre minimal de voitures qui, présentes dans le site $i$ à l'instant initial $t$ de la simulation $z$, garantie la satisfaction des demandes formulées dans ce site (pas de ruptures) pendant la période $[t, t+h]$.

Considérons la figure 1.3 qui présente les résultats obtenus pour un site $i$ lors d'une simulation $z$.

- ⋆ La courbe $\mathcal{A}$ représente l'évolution des demandes dans le site $i$. Les parties négatives de la courbe sont les départs du site $i$ à l'instant $t_x$, et les parties positives sont les arrivées au site considéré à l'instant $t_x$.

- ⋆ La courbe $\mathcal{B}$ indique la situation dans le site lorsque ce sont uniquement les voitures en provenance des autres sites qui servent à satisfaire les demandes jusqu'à l'instant $t_x$. Les parties négatives de la courbe correspondent aux ruptures dans le site considéré.

- ⋆ La courbe $\mathcal{C}$ est la courbe des ajouts de voitures en provenance du *Stock* dans le site au fur et à mesure que ceci est nécessaire. La courbe représente le cumul des voitures ajoutées.

- ⋆ La courbe $\mathcal{D}$ est l'évolution cumulée du nombre de voitures dans le site $i$ lors de la simulation $z$ si, à l'instant $t = t_0$, le nombre initial de voitures présentes dans le site $i$ était $m_{(i,z)}$. De ce fait, nous avons la garantie de ne pas avoir de rupture dans le site pendant la période $[t, t+h]$. L'étape suivante est de vérifier si, par contre, un encombrement a eu lieu. $M_{(i,z)}$ est le nombre maximal de voitures atteint dans le site $i$ pendant la période $[t, t+h]$ lorsque le nombre initial de voitures dans ce même site était $m_{(i,z)}$ à l'instant $t$. Pour qu'il n'y ait pas d'encombrement dans ce site, $M_{(i,z)}$ doit être inférieur ou égal à la capacité du site $i$ qui est $C_i$. En d'autres termes, la courbe $\mathcal{D}$ s'obtient de la courbe $\mathcal{B}$ et ceci en la faisant translater d'une distance égale à $m_{(i,z)}$.

- ⋆ La courbe $\mathcal{E}$ est le résultat du déplacement de la courbe $\mathcal{D}$ vers le haut en tenant compte de la capacité du site $i$. Nous obtenons ainsi $s_{(i,z)}$ qui est égal à $m_{(i,z)} + (C_i - M_{(i,z)})$ et qui indique le nombre maximal de voitures qui, présentes à l'instant initial $t$ dans le site $i$, ne causeraient pas d'encombrement tout le long de la période $[t, t+h]$. Finalement nous obtenons pour chaque site deux bornes qui sont $m_{(i,z)}$ et $s_{(i,z)}$. Lors de la simulation $z$, si le nombre initial de voitures dans le site $i$ à l'instant $t$ se situe entre les

deux bornes $m_{(i,z)}$ et $s_{(i,z)}$, nous avons la garantie du bon fonctionnement du système sur toute la période $[t, t+h]$.

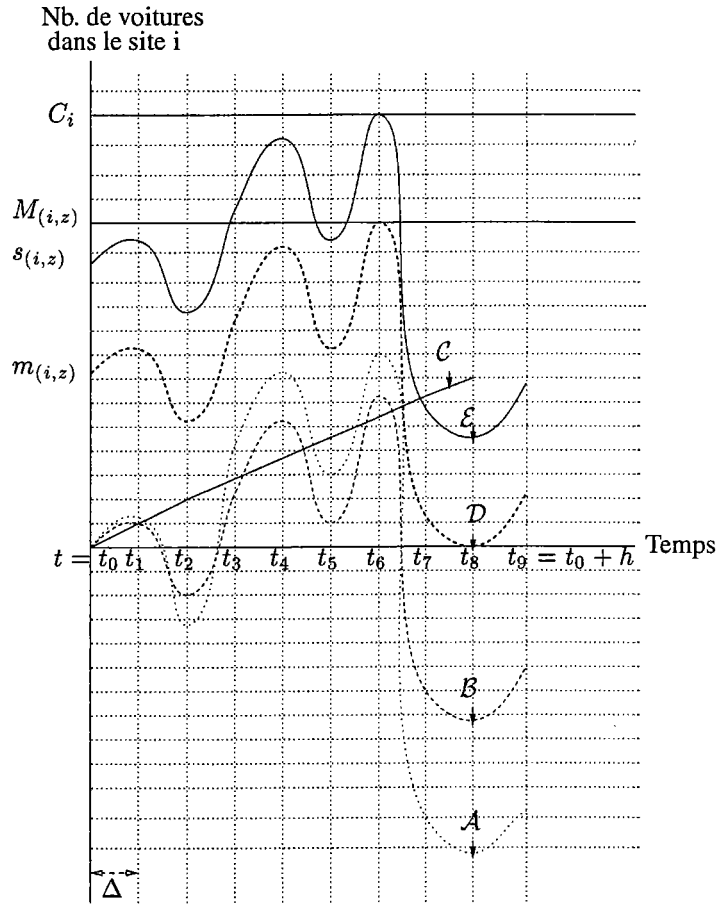



FIG. 1.3: *Représentation du nombre de voitures dans un site pour une simulation z.*

Nous continuons nos explications en considérant uniquement le site $i$, mais les mêmes étapes sont appliquées aux autres sites du système. A la fin de chacune des simulations $z$, nous calculons les bornes $m_{(i,z)}$ et $s_{(i,z)}$ (si possible) comme nous l'avons décrit précédemment. Nous avons précisé "si possible" car $m_{(i,z)}$ qui garantie uniquement la non rupture dans le site $i$, peut avoir comme conséquence un encombrement dans le site, ou bien être elle-même supérieure à la capacité du site. Dans ces cas là, $m_{(i,z)}$ n'est pas retenue. Nous reportons les résultats dans le tableau *Resultats* de dimension $C_i + 1$, comme le montre la figure 1.4. En fait, les valeurs du tableau *Resultats* qui se situent entre $m_{(i,z)}$ et $s_{(i,z)}$ sont incrémentées de 1. Au bout des $Z$ simulations, pour $n$ voitures initialement présentes dans le site $i$ ($n$ correspond au nombre initial de voitures présentes dans le site $i$ et varie entre 0 et $Min[C_i, N]$, sachant que $C_i$ est la capacité du site $i$ et que $N$ est le nombre total de voitures) nous connaissons le nombre $g^t_{(i,n)}$ de simulations qui se seraient bien déroulées (sans rupture et sans encombrement), puisque $n \in [m_{(i,z)}, s_{(i,z)}]$ et sachant que $z = 1, \cdots, Z$.

L'étape suivante consiste à calculer les différentes probabilités de bon fonctionnement (sans rupture, ni encombrement) pour le site $i$. Pour un horizon $h$, La probabilité de bon fonction-

nement dans le site $i$ lorsque $n$ voitures s'y trouvent à l'instant initial $t$ est égale à :

$$p^t_{(i,n)} = \frac{g^t_{(i,n)}}{Z} \qquad \forall\ n = 0, .., Min[C_i, N]$$

Ce calcul fait l'hypothèse que tous les sites sont toujours en mesure de fournir les voitures demandées par le site $i$. $p^t_{(i,n)}$ est donc une évaluation optimiste de la probabilité de bon fonctionnement lorsque le site $i$ contient $n$ voitures à l'instant $t$.

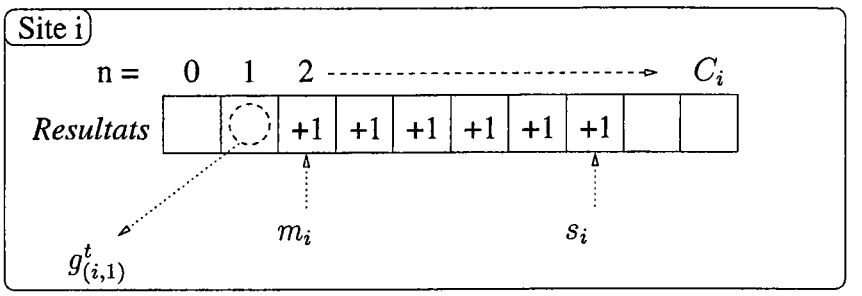



FIG. 1.4: *Tableau Résultats* pour le site $i$.

Notre objectif est de déterminer les états favorables à un moment donné afin de pouvoir ramener une situation classée *état défavorable* vers une situation favorable; ceci bien sûr avec les moyens dont nous disposons, autrement dit le nombre $\mathcal{N}$ de voitures garées dans les différents sites. L'étape suivante consiste à calculer les différents états favorables pour chacune des $\mathcal{N}$ voitures possibles sachant que $\mathcal{N} \in [0, N]$ et où $N$ est le nombre total de voitures.

Pour un nombre $\mathcal{N}$ de voitures, nous sélectionnons les plus grands $p^t_{(i,n)}$, de façon à ce que :

$$\sum_{i=1}^{K} (n^t_i) \leq \mathcal{N}$$

sachant que $n^t_i$ représente le nombre $n$ de voitures qui se trouvent dans le site $i$ à l'instant $t$.

Si ces différentes probabilités $p^t_{(i,n)}$ sont supérieures à $\mu$, cet état est retenu comme étant un *état favorable*.

## 1.4.3 Etat défavorable

Un état défavorable est une répartition qui ménera à une rupture ou à un encombrement dans un bref delai. Ceci dit, si l'on réagit dans les temps en rééquilibrant le système, ce problème peut être esquivé.

### 1.4.3.1 Définition

> *"Pour H petit donné (fixé supérieur ou égal au temps nécessaire pour le rééquilibrage du système) et $\nu$ donné (supérieur à 30%) :*

*Le nombre de voitures dans le site $i$ à l'instant $t$ participe à un état défavorable si la probabilité de* **rupture durable** *ou la probabilité d'*encombrement durable *sur l'intervalle $[t, t+H]$ est supérieur à $\nu$.*

*Une rupture ou un encombrement est dit durable s'il n'y a pas de rétablissement naturel de la situation dans le site considéré en moins de $L$ périodes élémentaires (ex. L=3).*"

### 1.4.3.2 Description de l'algorithme

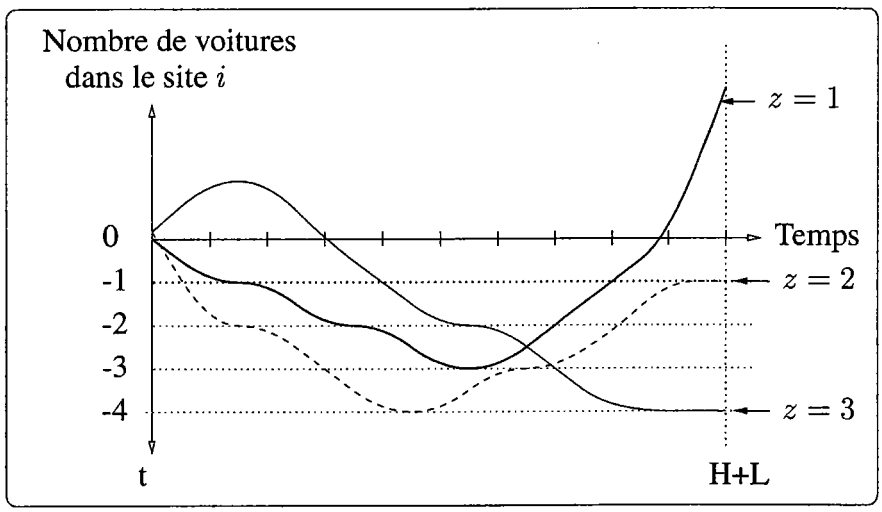



FIG. 1.5: *Différentes simulations.*

Avant de lancer les $Z$ simulations, nous déterminons l'instant initial $t$, l'horizon $H$, le nombre de périodes $L$, et la répartition initiale des voitures dans les différents sites. En fait, la répartition initiale est la même que celle que nous avons utilisée pour la détermination des états favorables. Toutes les voitures sont placées dans la file d'attente *Stock*.

Afin de déterminer les états défavorables à l'instant $t$, nous lançons $Z$ simulations sur un horizon de $[t, t+H+L]$ périodes élémentaires.

Au début de chaque période élémentaire, plusieurs évènements sont possibles, à savoir (nous nous plaçons dans le site $i$) :

- Les arrivées en provenance des autres sites. En fait nous considérons que les demandes formulées dans les autres sites sont toutes satisfaites. Ceci est donc une vue optimiste du système.

- Les départs de voitures du site $i$. Les voitures arrivées dans le site $i$ lors des précédentes périodes servent à satisfaire les demandes formulées au site $i$.

- Le nombre $R_i^\tau$ de demandes insatisfaites ou, autrement dit, de voitures indisponibles à la période $\tau$.

- Le nombre $I_i^\tau$ de voitures qui ne peuvent pas se garer dans le site $i$ car toutes les places de parking sont déjà prises.

*Remarque :*
*Les calculs nécessaires pour déterminer les états défavorables dûs à un encombrement de voitures dans un ou plusieurs sites sont les mêmes que ceux que nous détaillons ci-dessous pour déterminer les états défavorables dûs à une rupture dans un ou plusieurs sites.*

A la fin de chacune des simulations et pour chacun des sites, nous calculons la plus grande rupture durable qui a eu lieu, c'est à dire le nombre maximal de voitures qui ont manqué à chaque site pendant une durée supérieure ou égale à $L$.

Pour le site $i$, celui-ci est en rupture durable à l'instant $\tau$ si :

$$R_i^x > 0 \qquad \forall\ x \in [\tau, \tau + L]$$

Afin d'estimer le nombre $j$ de voitures qui manquent dans le site $i$ de $\tau$ à $\tau + L$, nous utilisons l'équation suivante :

$$j_\tau = \lceil \frac{1}{L} \sum_{x=\tau}^{\tau+L} R_i^x \rceil$$

où $j_\tau$ représente le nombre moyen (arrondi à l'entier supérieur) de voitures manquantes dans le site sur les périodes $\tau$, $\tau$ + 1, ..., $\tau$ + L.

A la fin de chaque simulation $z$, la plus grande rupture durable $j_{maxi}$ calculée est reportée dans le tableau 1.3.

$$j_{maxi} = Max_{\tau=1,2,...,H} \quad j_\tau$$

Ainsi, au bout des $Z$ simulations nous obtenons $S_i^t(j)$ qui est la fréquence des différentes ruptures. Autrement dit, pour un instant $t$, un horizon $H$ et un site $i$, $S_i^t(j)$ est le nombre de simulations où la plus grande rupture durable détectée était $j$.

TAB. 1.3: *La fréquence $S_i^t(j)$ pour différentes valeurs de ruptures durables.*

| Nombre de voitures | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| $S_i^t(j)$ | 325 | 165 | 150 | 99 |

Nous utilisons l'équation suivante afin de calculer $P_i^t(j)$ qui est la probabilité qu'un site $i$ soit en rupture de $j$ voitures à l'instant $t$ sur l'horizon $[t, t+H]$ :

$$P_i^t(j) = \frac{S_i^t(j)}{Z}$$

Nous obtenons ainsi le tableau 1.4.

TAB. 1.4: *La probabilité $P_i^t(j)$ pour $j$ = 0 à 3.*

| Nombre de voitures | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| $P_i^t(j)$ | 0.325 | 0.165 | 0.150 | 0.099 |

Comme nous nous intéressons aux cumul des fréquences, voir tableau 1.5, nous calculons :

$$c_i^t(j) = \sum_{m=0}^{max(j)} P_i^t(m)$$

TAB. 1.5: *Le cumul des fréquences $c_i^t(j)$.*

| Nombre de voitures | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| $c_i^t(j)$ | 0.739 | 0.414 | 0.249 | 0.099 |

Finalement, un état est catalogué *défavorable* pour cause de ruptures, si le cumul des fréquences pour le nombre de voitures qui s'y trouvent est supérieur au seuil $\nu$.

Prenons l'exemple où le seuil $\nu$ est 30% et le tableau des cumul de fréquences est celui listé ci-dessus, voir tableau 1.5. Nous constatons que nous avons un état défavorable si moins de 2 voitures se trouvent dans le site $i$ à l'instant $t$.

## 1.5 Conclusion

A la fin de ce chapitre, nous sommes en mesure de déterminer le moment où une procédure de rééquilibrage doit être enclenchée afin d'éviter des problèmes comme celui de rupture ou d'encombrement dans un ou des sites du système. Lorsqu'un état défavorable est detecté, des mesures sont prises afin de ramener le système vers un état favorable. Mais, quelles sont ces mesures? Les chapitres suivants vont répondre à cette question.

# Chapitre 2

# *Optimisation du coût de rééquilibrage*

## Résumé

*Ce chapitre a pour but de transformer une situation classée défavorable en une situation favorable en déterminant le nombre de voitures qu'il va falloir transporter de certains sites vers d'autres.*

*Mots-clés : Programmation linéaire, Simplexe, Transport.*

## Sommaire

## 2.1 Introduction

Dans ce chapitre, nous sommes dans la situation où un état défavorable a été détecté. Autrement dit, si nous ne réagissons pas dans les temps, un problème (soit un encombrement, soit une rupture) va avoir lieu dans un ou plusieurs sites. Notre but est de ramener la situation défavorable vers la meilleure situation favorable possible (calculée dans le chapitre précédent) avec le nombre de voitures dont nous disposons dans les différents sites. Autrement dit, nous allons déplacer des voitures à vide (sans clients) d'un site vers un autre. Cependant, des questions auxquelles nous allons répondre dans ce chapitre surgissent, à savoir :

1. Combien de voitures doivent être déplacées?
2. De quels sites vers quels autres sites les voitures doivent-elles être déplacées?

Le processus de rééquilibrage consiste à déplacer des voitures à vide d'un site vers un autre afin de se retrouver dans une situation du système classée *favorable*. Ce processsus de rééquilibrage peut avoir lieu de différentes façons, à savoir l'utilisation :

- d'un déplacement individuel. Un "jockey" déplace une voiture d'un site vers un autre.
- d'un déplacement collectif. Nous avons deux possibilités : la première est l'utilisation de camions sur lesquels les voitures sont chargées. Ces camions peuvent être de différentes capacités. Remarquez que lorsqu'un camion est de capacité un, ceci revient à utiliser un jockey. La seconde est l'utilisation du train de véhicules. C'est une technique développée au sein de l'équipe Praxitèle à l'I.N.R.I.A. Elle consiste à déplacer les voitures sans lien matériel entre elles. Un conducteur se trouve dans la première voiture du peloton et les autres, à l'aide de capteurs, la suivent. Pour plus de détails techniques, voir les articles [Fou96] et [Ele96].

## 2.2 Définition du problème

La situation actuelle de notre système est reconnue comme étant un état défavorable. Autrement dit, si aucune action de rééquilibrage n'est prise, nous aurons un problème sur un ou plusieurs sites du système avant d'atteindre l'horizon $h$, et ceci avec une forte probabilité $\mu$. Notre but est donc de nous ramener à un état favorable tout en minimisant les coûts. Nous connaissons :

- ⋆ La situation actuelle du système qui est classée *défavorable*. Autrement dit, nous connaissons la répartition de l'ensemble des voitures dans les sites. Ces voitures ne sont ni en cours d'utilisation ni en panne.
- ⋆ La situation favorable que l'on voudrait atteindre.
- ⋆ Le nombre de camions que l'on va utiliser pour effectuer les différents trajets du rééquilibrage, de même que les différentes capacités de ces camions.
- ⋆ Le coût associé au déplacement d'un camion entre deux sites. Ce coût comprend le temps mis par le camion pour effectuer ce déplacement, la paye du conducteur, le carburant nécessaire etc.

Notre but est de déterminer le nombre de voitures que l'on va déplacer, de même que les sites exédentaires et déficitaires. Un site exédentaire est un site qui fournit des voitures aux autres sites, tandis qu'un site déficitaire est celui qui a besoin de voitures pour satisfaire les demandes qui seront formulées dans le site avant l'horizon déterminé. Plus exactement, nous voulons déterminer les différents déplacements que nous devons effectuer afin d'atteindre l'état favorable choisi tout en minimisant les différents coûts.

## 2.3 Méthode de résolution : Simplexe

Nous formulons notre problème $\mathcal{P}_1$ comme suit :

$$\min \sum_{i=1}^{K} \sum_{j=1, j\neq i}^{K} \lceil \frac{r_{ij}}{z_{ij}} \rceil c_{ij}$$

$$\sum_{j=1, j\neq i}^{K} r_{ji} - \sum_{j=1, i\neq j}^{K} r_{ij} = D_i \qquad \forall \quad i = 1 \cdots K$$

$$r_{ij} \geq 0 \qquad \forall i, j \neq i$$

$$r_{ij} \quad entier$$

où :

- $r_{ij}$ est le nombre de voitures (que nous voulons déterminer) à déplacer du site $i$ vers le site $j$.
- $z_{ij}$ est la capacité du camion qui effectue le trajet entre le site $i$ et le site $j$.
- $c_{ij}$ est le coût du déplacement d'un camion du site $i$ vers le site $j$.
- $K$ est le nombre total de sites.
- $D_i$ représente la différence entre le nombre de voitures dans le site $i$ correspondant à l'état favorable et du nombre de voitures réellement présentes dans le site $i$ et qui fait partie d'un état classé défavorable. Si ce nombre $D_i$ est positif, ceci veut dire que des voitures doivent être ajoutées dans ce site parce que ce site sera bientôt en rupture de voitures.

La première contrainte indique le nombre de voitures ajoutées dans chacun des sites ou enlevées de chacun des sites pour atteindre le nombre de voitures correspondant à l'état favorable choisi. Pour chaque site considéré, les voitures ajoutées dans (ou enlevées de) ce site sont de provenance de n'importe quel autre site du système sauf bien sûr de site considéré.

Remarquez que notre objectif comprend une fonction non-linéaire conséquence du fait qu'un camion peut se déplacer sans être chargé au maximum. L'étape suivante consiste à négliger

cette contrainte ainsi que la contrainte d'intégrité afin d'obtenir un problème linéaire. Nous pourrons ainsi résoudre ce problème par la méthode du Simplexe détaillée dans le chapitre 1.2.1 de la seconde partie de ce mémoire.

Ainsi, nous obtenons le problème linéaire suivant, noté $\mathcal{P}_2$ :

$$\min \sum_{i=1}^{K} \sum_{j=1, j\neq i}^{K} r_{ij}\alpha_{ij}$$

$$\sum_{j=1, j\neq i}^{K} r_{ji} - \sum_{j=1, i\neq j}^{K} r_{ij} = D_i \qquad \forall \quad i = 1 \cdots K$$

$$r_{ij} \geq 0 \qquad \forall i, j \neq i$$

où :

- $\alpha_{ij}$ est le coût associé au déplacement d'une voiture du site $i$ au site $j$.

Dans ce modèle, nous mettons de côté les camions et leurs capacités. En fait nous nous focalisons sur les différents déplacements à effectuer entre les sites pour atteindre un état favorable sans nous soucier des moyens utilisés pour les effectuer. Notre objectif se transforme et consiste désormais à minimiser les coûts associés aux divers déplacements nécessaires pour faire basculer la situation vers un état favorable.

Les solutions obtenues en appliquant l'algorithme du Simplexe au problème linéaire $\mathcal{P}_2$, tout en abandonnant la contrainte d'intégrité, sont entières. En fait, si nous reformulons la première contrainte du problème linéaire $\mathcal{P}_2$, nous obtenons :

$$\sum_{j=1, j\neq i}^{K} r_{ji} - \sum_{j=1, i\neq j}^{K} r_{ij} = D_i \qquad \forall \quad i = 1 \cdots K$$

$$\sum_{j=1, j\neq i}^{K} r_{ji} - \sum_{j=1, i\neq j}^{K} r_{ij} = b_i - a_i \qquad \forall \quad i = 1 \cdots K$$

$$\text{Ainsi : } \sum_{j=1, j\neq i}^{K} r_{ji} = b_i \qquad \forall \quad i = 1 \cdots K$$

$$\sum_{j=1, i\neq j}^{K} r_{ij} = a_i \qquad \forall \quad i = 1 \cdots K$$

où $a_i$ représente le nombre de voitures que le site $i$ peut fournir aux autres sites et $b_i$ par contre, représente le nombre de voitures dont le site $i$ a besoin.

Nous obtenons ainsi le problème linéaire noté $\mathcal{P}_3$ :

$$\min \sum_{i=1}^{K} \sum_{j=1, j\neq i}^{K} r_{ij}\alpha_{ij}$$

$$\sum_{j=1, j\neq i}^{K} r_{ji} = b_i \qquad \forall \quad i = 1 \cdots K$$

$$\sum_{j=1, i\neq j}^{K} r_{ij} = a_i \qquad \forall \quad i = 1 \cdots K$$

$$r_{ij} \geq 0 \qquad \forall i, j \neq i$$

Rappelons le corollaire 1 cité dans [Sak84a] : *Toute solution de base du programme linéaire du problème de transport est entière si a et b sont entièrs.* Ce corollaire s'applique à la formulation précédente car les $b_i$ (resp. $a_i$) qui représentent le nombre de voitures à ajouter (resp. enlever) dans le (resp. du) site $i$ sont des entiers. Donc la solution optimale obtenue sera entière.

## 2.4 Conclusion

A la fin de ce chapitre, grâce à la méthode du Simplexe, nous connaissons le nombre de voitures que l'on doit déplacer entre chaque paire de sites afin de faire basculer la situation du système qui est défavorable vers une situation favorable. La question que l'on se pose ensuite est la suivante : comment allons-nous effectuer ces déplacements? Le chapitre suivant répond à cette question.

# Chapitre 3

# *Optimisation du temps de rééquilibrage*

## Résumé

*Ce chapitre a pour but de déterminer les tournées des différents moyens utilisés pour le rééquilibrage du système de façon à minimiser le temps total nécessaire pour l'effectuer.*

*Mots-clés : Simulation, Transport, Programmation linéaire*

## Sommaire

## 3.1 Introduction

Dans le chapitre précédent, nous avons déterminé le nombre de voitures que l'on doit déplacer entre chaque paire de sites afin de faire basculer notre système, qui se trouve dans un état classé *défavorable*, vers un état classé *favorable*. Ce changement d'état doit être accompli dans les plus brefs délais. En effet, une voiture transportée par un système quelconque ne peut pas être mise en service, et par conséquent être utilisée, par un client. Notre intérêt est de diminuer au maximum le temps d'immobilisation d'une voiture, et surtout le temps total nécessaire pour rééquilibrer le système en entier. Souvenez-vous que l'horizon utilisé pour les états favorables et défavorables est déterminé en fonction du temps nécessaire au rééquilibrage. De plus, le personnel monopolisé pour le rééquilibrage ne peut effectuer d'autres tâches comme, par exemple, le rapatriement des voitures en panne.

Dans ce chapitre, les réponses aux deux questions posées dans le premier chapitre de cette partie et qui sont *Quel est le moyen utilisé pour le rééquilibrage?* et *Ce moyen doit-il avoir une capacité fixe?*, sont connues. En fait, nous considérons que la société d'exploitation de notre système fixe le type des différents moyens de rééquilibrage, - dans ce chapitre nous parlerons de camions - le nombre de ces moyens ainsi que leur différentes capacités. Par contre, ce chapitre a pour objectif de répondre à la question : *Quel est le trajet du moyen utilisé pour le rééquilibrage?* A chaque moyen doivent être attribués plusieurs déplacements à effectuer, en précisant l'ordre d'exécution des tâches, de façon à ce que le temps total du rééquilibrage soit le plus petit possible.

## 3.2 Définition du problème

Nous sommes dans la situation où un état défavorable a été détecté. Un rééquilibrage du système est nécessaire si l'on veut éviter les ruptures et les encombrements dans les sites. Nous connaissons les différents déplacements à effectuer, calculés dans le chapitre précédent, et nous voulons déterminer les tournées des camions utilisés pour le rééquilibrage du système. La situation est illustrée dans la figure 3.1.

Les différents paramètres dont nous disposons sont :

- ⋆ Les différents déplacements calculés dans le chapitre précédent qui sont nécessaires au rééquilibrage du système.
- ⋆ Le nombre de camions utilisés pour la redistribution des voitures dans les sites.
- ⋆ Les capacités des différents camions.
- ⋆ Les durées nécessaires aux camions pour couvrir les distances entre chaque paire de sites.
- ⋆ L'emplacement de chacun de ces camions au moment du lancement du processus de rééquilibrage.

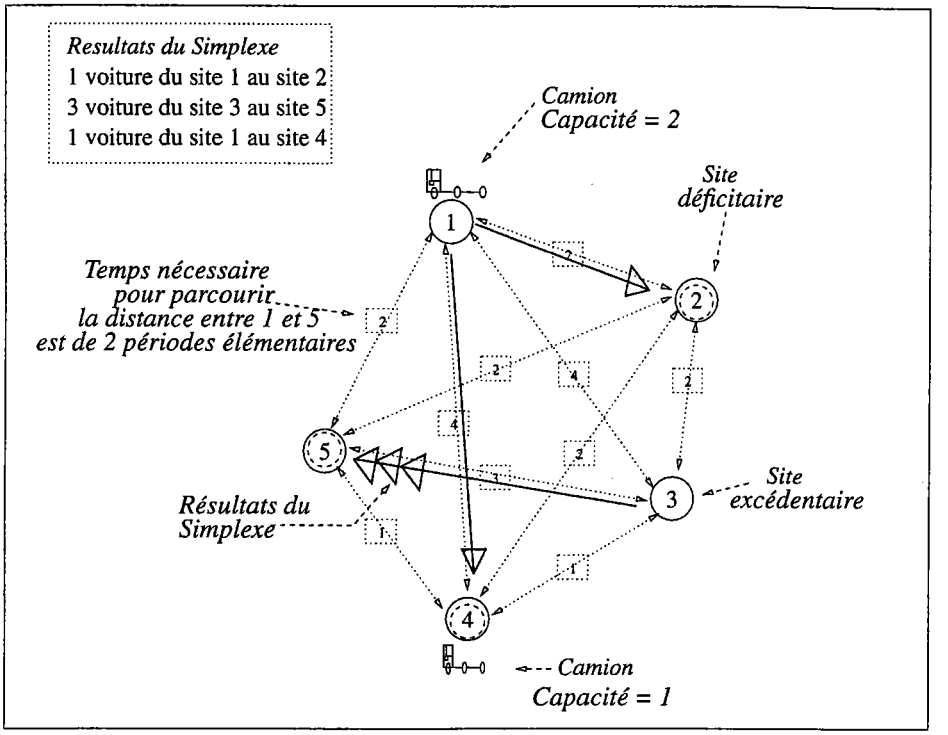



FIG. 3.1: *Présentation de la situation avant la détermination des trajets des camions*

Notre objectif est de minimiser le temps global nécessaire pour accomplir tous les déplacements du rééquilibrage déterminés dans le chapitre précédent. Un déplacement de rééquilibrage consiste à déplacer une voiture d'un site excédentaire vers un site déficitaire. Le processus de rééquilibrage est terminé lorsque le dernier déplacement est accompli. Nous voulons attribuer à chacun des camions une tournée, autrement dit, une séquence de déplacements à effectuer. En fait, minimiser le temps global du processus de rééquilibrage revient à minimiser la durée de la plus longue tournée parmi les tournées des différents camions.

Un camion est habituellement obligé d'effectuer un trajet à vide (sans voitures) avant d'atteindre un site excédentaire, sauf si le camion se trouve déjà dans un site excédentaire. Lorsqu'un camion passe dans un site excédentaire, des voitures y sont chargées. Si la capacité du camion le permet, tous les déplacements calculés entre deux sites spécifiques sont satisfaits lors d'un unique passage du camion, sinon, plusieurs camions ou plusieurs passages sont nécessaires. Par hypothèse, un camion est obligé de passer par un site excédentaire avant de passer par un site déficitaire.

Les paramètres utilisés dans le modèle sont :

- $w_{ij}^c$ est le temps nécessaire au camion $c$ pour couvrir la distance qui sépare le site $i$ du site $j$.

- $d_{ij}^c$ est le nombre de fois que le camion $c$ parcourt la distance qui sépare le site $i$ du site $j$.

- $\mathcal{N}^c_{ij(s)}$ est le nombre de fois que le camion $c$ parcourt la distance entre le site $i$ et le site $j$ en ayant débuté sa course au site $s$.
- $n$ est le nombre de camions chargés du rééquilibrage.
- $r_{ij}$ est le nombre de déplacements entre le site $i$ et le site $j$, calculés dans le chapitre précédent, qui sont nécessaires pour faire basculer la situation vers l'état favorable choisi.
- $K$ est le nombre de sites qui constituent le système.
- $V^c$ est la capacité du camion $c$.

Nous formulons le problème de la manière suivante :

$$\min T$$

$$T \geq \max_{c\in[1,n]} \sum_{s=1}^{K}\sum_{i=1}^{K}\sum_{j=1}^{K}(w^c_{si} + w^c_{ij})\mathcal{N}^c_{ij(s)}$$

$$\sum_{s=1}^{K} \mathcal{N}^c_{ij(s)} = d^c_{ij} \tag{3.1}$$

$$\sum_{c=1}^{n} d^c_{ij} V^c \geq r_{ij} \qquad \forall\, i,j = 1,\cdots,K \tag{3.2}$$

$$\sum_{l=1,l\neq j}^{K} d^c_{lj} - \sum_{m=1,m\neq j}^{K} d^c_{jm} = \begin{cases} 1 & \text{si } j \in \text{sites destinataires} \\ \text{-1} & \text{si } j \in \text{sites sources} \\ 0 & \text{Sinon} \end{cases} \qquad \begin{array}{l} \forall\ j = 1,\cdots,K \\ et\ c = 1,\cdots,n \end{array} \tag{3.3}$$

$d^c_{ij}$ et $\mathcal{N}^c_{ij(s)}$ entiers positifs. (3.4)

Notre objectif, comme précisé antérieurement, est de minimiser le temps global d'exécution de chaque processus de rééquilibrage lancé. Cet objectif revient à minimiser la plus longue tournée (dans le temps). La tournée d'un camion est constituée d'une chaîne de trajets. Un trajet est, lui même, divisé en deux déplacements successifs; le premier est un déplacement à vide (le camion ne transporte pas de voitures) afin d'atteindre le site excédentaire et le second est un déplacement du rééquilibrage (le camion transporte des voitures) : le camion atteint un site déficitaire et y dépose les voitures demandées. Le nombre de fois où le camion $c$ parcourt la distance entre le site $i$ et le site $j$ est égal à la somme des trajets qui passent par ces sites en ayant débuté la course dans différents sites, voir 3.1. La contrainte 3.2 indique qu'il est impératif que tous les déplacements, calculés dans le chapitre précédent, soient effectués. Un camion peut, en parcourant une seule fois la distance entre le site $i$ et le site $j$, accomplir le nombre de déplacements nécessaires. Une autre alternative se présente lorsque le camion est obligé de parcourir plusieurs fois ce trajet car sa capacité est insuffisante. La

dernière alternative est que plusieurs camions se partagent la tâche. De toutes les façons, tous les déplacements entre le site $i$ et $j$ calculés dans le chapitre précédent doivent être effectués. La contrainte 3.3 indique que le trafic des camions dans les sites suit la loi de Kirchoff, à l'exception de deux cas. Dans le premier cas, le camion débute sa tournée. Dans le second cas, le camion termine sa tournée en atteignant le dernier site déficitaire. Finalement la dernière contrainte signale qu'un camion ne peut effectuer un trajet partiellement.

## 3.3 Différentes méthodes de résolution

Comme le problème de l'attribution des différents déplacements aux différents camions est un problème $NP$-difficile, nous appliquons une heuristique. L'article [DFR98] propose une méthode basée sur le Branch and Bound pour résoudre ce problème, mais celle-ci n'est pas adaptable à notre cas, puisque notre but est de déterminer les tournées des camions en temps réél. Les sections suivantes de ce chapitre présentent les différentes heuristiques développées. Celles-ci sont décrites, testées et commentées. Le but de ces heuristiques, comme nous l'avons déjà précisé, est de définir les différents trajets de chacun des camions afin de terminer le processus de rééquilibrage dans les plus brefs délais. Le lieu où se trouve un camion est important lors de l'attribution des différents déplacements aux différents camions.

*Remarque :* nous établissons une distinction entre trajet et déplacement. Un trajet est le chemin entre deux sites qu'un camion doit parcourir parce qu'il a un ou plusieurs déplacements à effectuer. Un déplacement est fixé par le Simplexe et consiste à déplacer une seule voiture d'un site excédentaire vers un site déficitaire.

### 3.3.1 Heuristique 1 : *Priorité au site le plus proche*

La philosophie de cette heuristique est de commencer à satisfaire les déplacements qui sont le plus rapidement terminés. En d'autres termes, le but de cette heuristique est de s'acquitter de la majeure partie du rééquilibrage dans les plus brefs délais, et de laisser les déplacements qui nécessitent le plus de temps pour la fin. Ainsi, un grand nombre de voitures sont remises en service plus rapidement. La figure 3.2 montre le fonctionnement de l'algorithme développé.

Les différentes étapes de l'algorithme *Priorité au site le plus proche* sont :

**1 Saisie des données**

1.1 DateCourante = date du début du rééquilibrage

1.2 Pour tous les camions $ca$,

1.2.1 Emplacement[$ca$] = le site où se trouve le camion $ca$

1.2.2 Capacité[$ca$] = la capacité du camion $ca$

1.2.3 DateLibre[$ca$] = moment où le camion $ca$ est libre. Ce moment est initialisé à DateCourante

1.3 Pour tous les trajets $tr$,

1.3.1 Départ[$tr$] = site de départ du trajet $tr$

1.3.2 Arrivée[$tr$] = site d'arrivée du trajet $tr$

1.3.3 NbTrajet[$tr$] = nombre de déplacements à effectuer sur le trajet $tr$

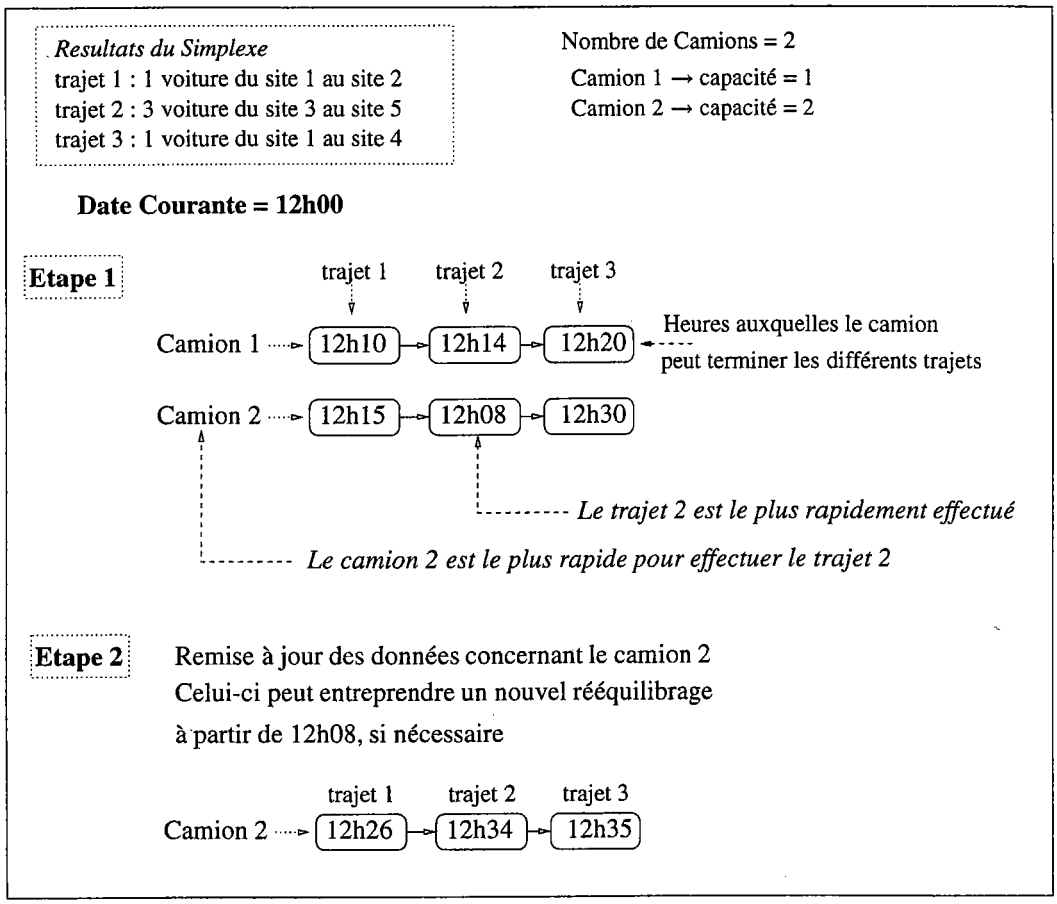



FIG. 3.2: *Présentation de l'heuristique* PRIORITÉ AU SITE LE PLUS PROCHE

## 2 Estimation des temps nécessaires à chaque camion pour accomplir les différents trajets

2.1 Pour tous les trajets $tr$

2.1.1 Pour tous les camions $ca$,

2.1.1.1 Calculer TrajetVide[$tr$][$ca$]. C'est le temps nécessaire au camion pour parcourir la distance entre Emplacement[$ca$] et Départ[$tr$] en commençant à la date DateLibre[$ca$]

2.1.1.2 Calculer TrajetPlein[$tr$][$ca$]. C'est le temps nécessaire au camion $ca$ pour parcourir la distance entre le site du Départ[$tr$] et le site d'Arrivée[$tr$] à la date de départ DateLibre[$ca$]+TrajetVide[$tr$][$ca$]

2.1.1.3 Calculer FinTrajet[$tr$][$ca$] = DateLibre[$ca$] + TrajetVide[$tr$][$ca$] + TrajetPlein[$tr$][$ca$]

**3 Affectation des différents déplacements aux différents camions.**

3.1 Tant que les déplacements ne sont pas tous affectés,

3.1.1 Pour tous les camions $ca$,

3.1.1.1 Pour tous les trajets $tr$ tel que NbTrajet[$tr$] $\neq 0$,

3.1.1.1.1 Choisir le camion $ca^c$ qui est le plus rapide pour effectuer un trajet. Autrement dit, FinTrajet[$tr$][$ca^c$] est minimal.

3.1.1.1.2 Si plusieurs camions sont de rapidité égale pour accomplir un même trajet $tr$ , alors nous prenons en considération d'autres critères tels que :

a. Opter pour le camion qui a la Capacité[$ca$] la plus petite supérieure à NbTrajet[$tr$]

b. Si les Capacité[$ca$] < NbTrajet[$tr$]

Alors opter pour le camion qui a la Capacité[$ca$] la plus grande

3.1.2 $tr\_c$ est le trajet qui peut être effectué le plus rapidement.

3.1.3 Si NbTrajet[$tr\_c$] > Capacité[$ca^c$]

Alors $\longrightarrow$ NbTrajet[$tr\_c$] = NbTrajet[$tr\_c$] - Capacité[$ca^c$]

Sinon $\longrightarrow$ NbTrajet[$tr\_c$] = 0

3.1.4 DateLibre[$ca^c$] = DateLibre[$ca^c$] + TrajetVide[$tr\_c$][$ca^c$] + TrajetPlein[$tr\_c$][$ca^c$]

3.1.5 Emplacement[$ca^c$] = Arrivée[$tr\_c$]

3.1.6 Pour tous les trajets $tr$ tel que NbTrajet[$tr$] $\neq 0$,

3.1.6.1 Calculer TrajetVide[$tr$][$ca^c$]. C'est le temps nécessaire au camion $ca^c$ pour parcourir la distance entre Emplacement[$ca^c$] et Départ[$tr$] en partant à la date DateLibre[$ca^c$]

3.1.6.2 Calculer TrajetPlein[$tr$][$ca^c$]. C'est le temps nécessaire au camion $ca^c$ pour parcourir la distance entre le site du Départ[$tr$] et le site d'Arrivée[$tr$] à la date de départ DateLibre[$ca^c$]+TrajetVide[$tr$][$ca^c$]

3.1.6.3 Calculer FinTrajet[$tr$][$ca^c$] = DateLibre[$ca^c$] + TrajetVide[$tr$][$ca^c$] + TrajetPlein[$tr$][$ca^c$]

### 3.3.2 Heuristique 2 : *Services groupés*

L'idée principale de cette heuristique est de grouper plusieurs trajets. Autrement dit, si un site doit fournir deux voitures à deux sites différents, si la capacité du camion le permet, et si bien sûr c'est plus rapide, le camion va charger les deux voitures et desservir les deux sites déficitaires en séquence. Pour faciliter la compréhension prenons l'exemple suivant (voir illustration figure 3.3) où un seul camion se charge de la procédure de rééquilibrage. Les résultats du Simplexe sont de déplacer une voiture du site 1 vers le site 2 et une autre voiture du site 1 vers le site 4. L'heuristique examine la situation. Si c'est plus avantageux de desservir les deux sites déficitaires sans repasser par le site excédentaire, les deux voitures seront alors chargées dans le camion lorsque celui-ci se trouve dans le site excédentaire. La

trajectoire du camion sera la suivante : lorsque le camion est dans le site 1, il charge les deux voiture, ensuite il passe par le site 2, dépose la première voiture, continue sa route jusqu'à atteindre le site 4, et finalement décharge la dernière voiture.

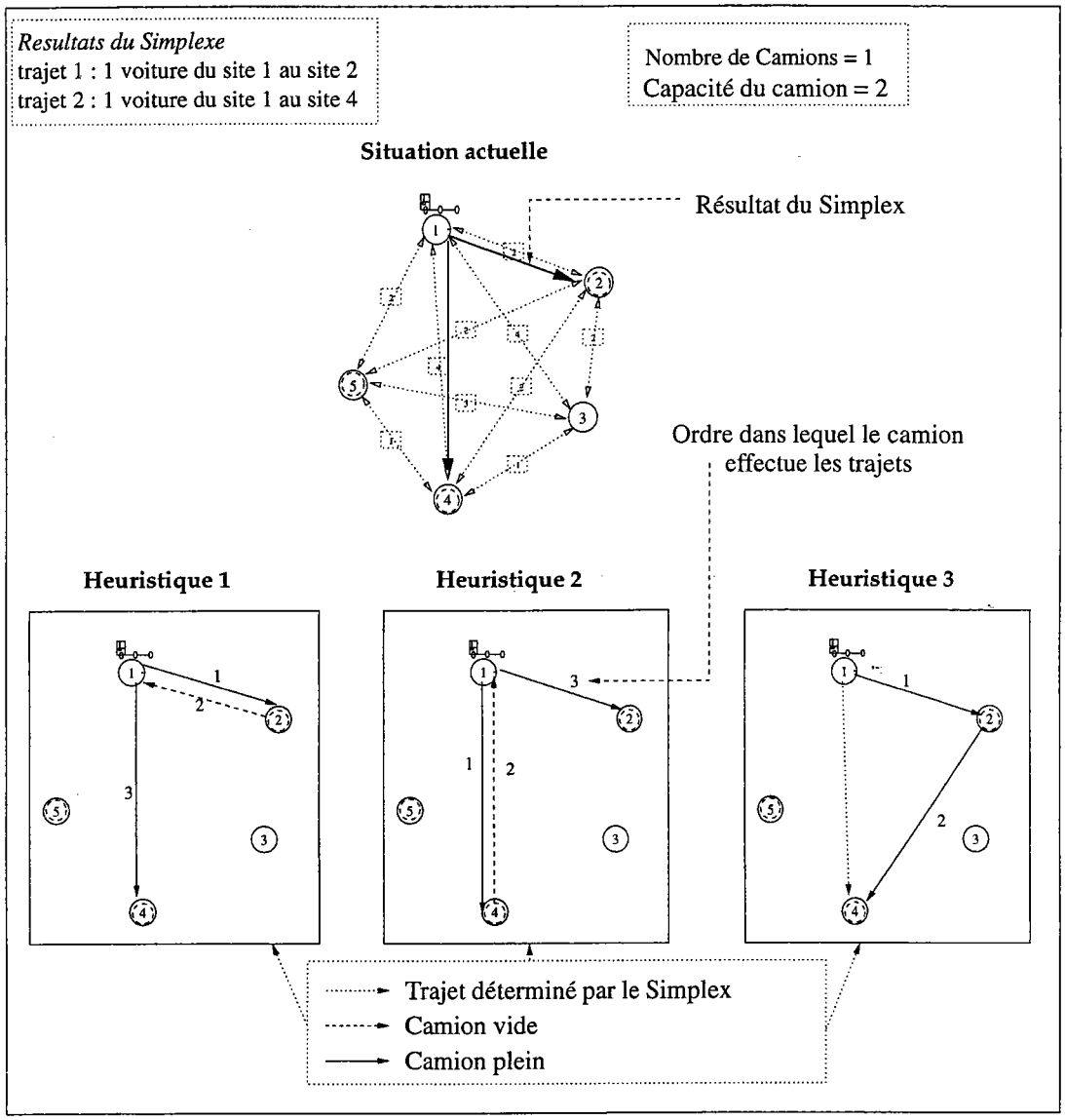



FIG. 3.3: *Présentation du fonctionnement de l'heuristique* SERVICE GROUPÉS

Pour cette heuristique, nous allons introduire un nouveau terme qui est *trajet de liaison*. Les trajets de liaison sont les trajets qui relient les différents sites déficitaires. Ces trajets ne font pas parti des trajets déterminés par le Simplexe pour le rééquilibrage.

Les paragraphes suivants détaillent l'algorithme *Services groupés*.

**1 Saisie et initialisation des données**

1.1 DateCourante = date à laquelle on se trouve

1.2 Pour tous les camions $ca$,

1.2.1 Emplacement[$ca$] = le site où se trouve le camion $ca$

1.2.2 Capacité[$ca$] = la capacité du camion $ca$

1.2.3 DateLibre[$ca$] = moment où le camion $ca$ est libre = DateCourante

1.3 Pour tous les trajets $tr$,

1.3.1 Départ[$tr$] = site de départ du trajet $tr$

1.3.2 Arrivée[$tr$] = site d'arrivée du trajet $tr$

1.3.3 NbTrajet[$tr$] = nombre de voitures qu'il faut transférer sur le trajet $tr$

1.3.4 Source = la liste des stations excédentaires

1.3.5 Destination = la liste des stations déficitaires

1.3.6 NbTotalDépart[Source] = le nombre total de voitures qui doivent partir de ce site

1.4 Pour tous les sites déficitaires $i$ appartenant à la liste Destination

1.4.1 Pour tous les sites déficitaires $j$ de la liste Destination, et lorsque $i \neq j$

1.4.1.1 DépartImag[$tr^I$] = $i$

1.4.1.2 ArrivéeImag[$tr^I$] = $j$

**2 Calcul des temps nécessaires mis par camion pour effectuer les différents trajets**

2.1 Pour tous les trajets $tr$

2.1.1 Pour tous les camions $ca$,

2.1.1.1 Calculer TrajetVide[$tr$][$ca$]. C'est le temps nécessaire au camion pour parcourir la distance entre Emplacement[$ca$] et Départ[$tr$] en partant à la date DateLibre[$ca$]

2.1.1.2 Calculer TrajetPlein[$tr$][$ca$]. C'est le temps nécessaire au camion $ca$ pour parcourir la distance entre le site du Départ[$tr$] et le site d'Arrivée[$tr$] à la date de départ DateLibre[$ca$]+TrajetVide[$tr$][$ca$]

2.1.1.3 Calculer FinTrajet[$tr$][$ca$] = DateLibre[$ca$] + TrajetVide[$tr$][$ca$] + TrajetPlein[$tr$][$ca$]

**3 Affectation des différents déplacements aux différents camions**

3.1 Tant que tous les trajets ne sont pas satisfaits

3.1.1 Choisir le plus grand camion qui peut le plus rapidement s'acquitter d'un trajet. Autrement dit, FinTrajet[$tr$][$ca$] est minimal.

$tr^a = tr$ et $ca^a = ca$ représente le camion choisi pour effectuer le trajet $tr^a$

3.1.2 Emplacement[$ca^a$] = Arrivée[$tr^a$]

3.1.3 DateLibre[$ca^a$] = DateLibre[$ca^a$] + FinTrajet[$tr^a$][$ca^a$]

3.1.4 Si Capacité[$ca^a$] $\leq$ NbTrajet[$tr^a$]

Alors $\longrightarrow$ NbTotalDépart[Départ[$tr^a$]] -= Capacité[$ca^a$]

NbTrajet[$tr^a$] = NbTrajet[$tr^a$] - Capacité[$ca^a$]

Sinon $\longrightarrow$ NbTotalDépart[Départ[$tr^a$]] -= NbTrajet[$tr^a$]
$nbv$ = NbTrajet[$tr^a$]
*Remarque : nbv est le nombre de voitures présentes dans le camion.*
NbTrajet[$tr^a$] = 0

3.1.5 Pour tous les trajets $tr$ tel que NbTrajet[$tr$] $\neq 0$,
3.1.5.1 Calculer TrajetVide[$tr$][$Ca^a$]. C'est le temps nécessaire au camion pour parcourir la distance entre Emplacement[$Ca^a$] et Départ[$tr$] en partant à la date DateLibre[$Ca^a$]
3.1.5.2 Calculer TrajetPlein[$tr$][$Ca^a$]. C'est le temps nécessaire au camion $Ca^a$ pour parcourir la distance entre le site du Départ[$tr$] et le site Arrivée[$tr$] à la date de départ = DateLibre[$Ca^a$] + TrajetVide[$tr$][$Ca^a$]
3.1.5.3 Calculer FinTrajet[$tr$][$Ca^a$]=DateLibre[$Ca^a$]+TrajetVide[$tr$][$Ca^a$] + TrajetPlein[$tr$][$Ca^a$]

3.1.6 Si Capacité[$ca^a$] > NbTrajet[$tr^a$] et NbTrajet[$tr^a$] < NbTotalDépart[$tr^a$]
3.1.6.1 Tant que $nbv$ < Capacité[$ca^a$]
3.1.6.2.1 Pour tous les trajets de liaison où DépartImag[$tr^I$] = Emplacement[$ca^a$]
3.1.6.2.1.1 Calculer FinTrajetsImags[$tr^I$][$ca^a$]
3.1.6.2.2 Sélectionner $tr^{Ia}$ qui est le trajet le plus rapide parmi ces trajets de liaison. Autrement dit celui dont FinTrajetsImags[$tr^{Ia}$][$ca^a$] est minimal.
3.1.6.2.3 *Combien* est initialisé à zéro
3.1.6.2.4 Pour tous les camions $ca$,
3.1.6.2.4.1 Si FinTrajet[$tr^n$][$ca$] $\leq$ FinTrajetsImags[$tr^{Ia}$][$ca^a$]
*Remarque : $tr^n$ est le trajet dont Départ[$tr^n$] = Départ[$tr^a$] et Arrivée[$tr^n$] = ArrivéeImag[$tr^{Ia}$]*
Alors $\longrightarrow$ *Combien* = Capacité[$ca$]
3.1.6.2.5 Si NbTrajet[$tr^n$] > *Combien*
3.1.6.2.5.1 Emplacement[$ca^a$] = ArrivéeImag[$tr^{Ia}$]
3.1.6.2.5.2 DateLibre[$ca^a$] = DateLibre + FinTrajetsImags[$tr^{Ia}$][$ca^a$]
3.1.6.2.5.3 Si NbTrajet[$tr^n$] < (Capacité[$ca^a$]-$nbv$)
Alors $\longrightarrow$ $nbv$ = $nbv$ + NbTrajet[$tr^n$]
NbTrajet[$tr^n$] = 0
Sinon $\longrightarrow$ $nbv$ = Capacité[$ca^a$]
NbTrajet[$tr^n$] -= (Capacité[$ca^a$] - $nbv$)

### 3.3.3 Heuristique 3 : *Priorité au site le plus éloigné*

La troisième heuristique présentée dans ce mémoire se résume à dire : puisque de toutes façons tous les déplacements doivent être effectués, autant s'acquitter en premier des trajets

qui nécessitent le plus de temps, et de terminer avec les trajets les plus rapides. Le camion qui peut effectuer le trajet le plus long dans les plus brefs délais est choisi. La figure 3.4 détaille le fonctionnement de l'heuristique 3.

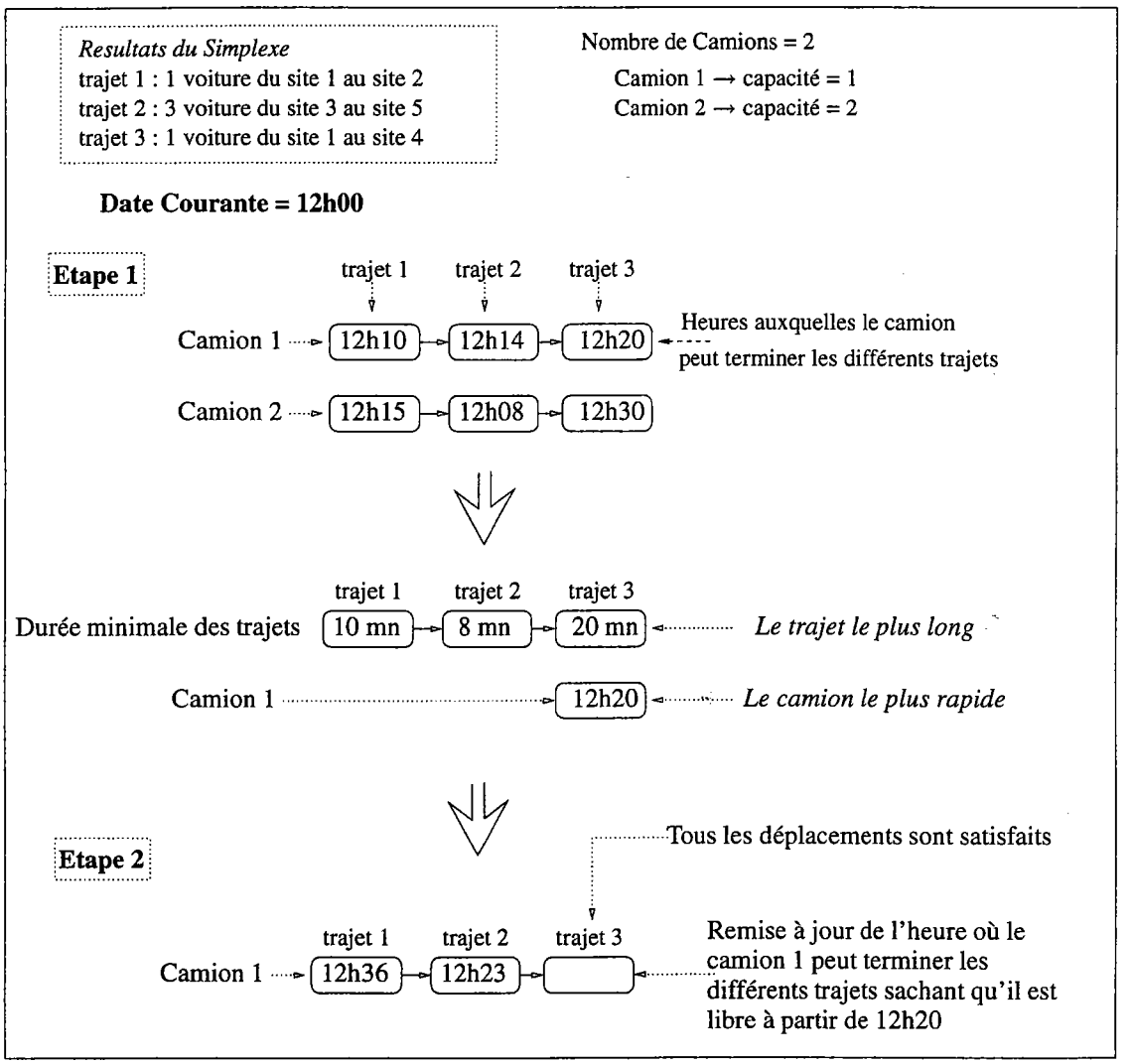



FIG. 3.4: *Présentation de* PRIORITÉ AU SITE LE PLUS ÉLOIGNÉ

Les différentes étapes de l'algorithme *Priorité au site le plus éloigné* sont :

**1 Saisie des données**

1.1 DateCourante = date du début du rééquilibrage

1.2 Pour tous les camions *ca*,

1.2.1 Emplacement[*ca*] = le site où se trouve le camion *ca*

1.2.2 Capacité[*ca*] = la capacité du camion *ca*

1.2.3 DateLibre[*ca*] = moment où le camion *ca* est libre et il est initialisé à DateCourante

1.3 Pour tous les trajets $tr$,

1.3.1 Départ[$tr$] = site de départ du trajet $tr$

1.3.2 Arrivée[$tr$] = site d'arrivée du trajet $tr$

1.3.3 NbTrajet[$tr$] = nombre de déplacements à effectuer sur le trajet $tr$

**2 Calcul des temps nécessaires pour chaque camion pour effectuer les différents trajets**

2.1 Pour tous les trajets $tr$

2.1.1 Pour tous les camions $ca$,

2.1.1.1 Calculer TrajetVide[$tr$][$ca$]. C'est le temps nécessaire au camion pour parcourir la distance entre Emplacement[$ca$] et Départ[$tr$] en partant à la date DateLibre[$ca$]

2.1.1.2 Calculer TrajetPlein[$tr$][$ca$]. C'est le temps nécessaire au camion $ca$ pour parcourir la distance entre le site du Départ[$tr$] et le site d'Arrivée[$tr$] à la date de départ DateLibre[$ca$]+TrajetVide[$tr$][$ca$]

2.1.1.3 Calculer FinTrajet[$tr$][$ca$] = DateLibre[$ca$] + TrajetVide[$tr$][$ca$] + TrajetPlein[$tr$][$ca$]

**3 Affectation des différents déplacements aux différents camions.**

3.1 Tant que les déplacements ne sont pas tous affectés,

3.1.1 Pour tous les trajets $tr$ tel que NbTrajet[$tr$] $\neq 0$,

3.1.1.1 Choisir *Lcaplusrapide* qui est le camion le plus rapide pour effectuer le trajet $tr$. Autrement dit, FinTrajet[$tr$][$ca$] est minimal.

3.1.1.1.1 Si plusieurs camions sont de rapidité égale pour effectuer le même trajet $tr$ , alors nous optons pour celui qui satisfait un des critères suivants, listés dans l'ordre de priorité, à savoir :

a. Choisir le camion dont la Capacité[$ca$] = la plus petite supérieure à NbTrajet[$tr$]

b. Si Capacité[$ca$] des différents camions < NbTrajet[$tr$]
Alors opter pour le camion qui a la Capacité[$ca$] la plus grande

3.1.1.2 FinTrajetAuPlusTot[$tr$] = FinTrajet[$tr$][*Lcaplusrapide*]

3.1.1.3 CamionLePlusRapide[$tr$]=*Lcaplusrapide*

3.1.2 Rechercher *Ltrpluslong* qui est le trajet le plus long à effectuer parmi la liste des trajets $tr$ où NbTrajet[$tr$] $\neq 0$. Autrement dit, rechercher où FinTrajetAuPlusTot[$tr$] est maximal.

3.1.3 Si card(*Ltrpluslong* ) = 1

Alors $\longrightarrow$ $Tr^{'}$ = *Ltrpluslong* et $Ca^{'}$ = CamionLePlusRapide[$Tr^{'}$]

Sinon $\longrightarrow$ choisir $Tr^{'}$ parmi la liste de *Ltrpluslong* en considérant différents critères comme :

a. Choisir le trajet qui, une fois desservi, aura comme conséquence NbTrajet[$Tr^{'}$] = 0

b. Sion opter pour celui qui aura comme conséquence NbTrajet[$Tr^{\epsilon}$] minimal

3.1.4 Si NbTrajet[$Tr^{\epsilon}$] > Capacité[$Ca^{\epsilon}$]
Alors $\longrightarrow$ NbTrajet[$Tr^{\epsilon}$] = NbTrajet[$Tr^{\epsilon}$] - Capacité[$Ca^{\epsilon}$]
Sinon $\longrightarrow$ NbTrajet[$Tr^{\epsilon}$] = 0

3.1.5 DateLibre[$Ca^{\epsilon}$] = DateLibre[$Ca^{\epsilon}$] + TrajetVide[$Tr^{\epsilon}$][$Ca^{\epsilon}$] + TrajetPlein[$Tr^{\epsilon}$][$Ca^{\epsilon}$]

3.1.6 Emplacement[$Ca^{\epsilon}$] = Arrivée[$Tr^{\epsilon}$]

3.1.7 Pour tous les trajets $tr$ tel que NbTrajet[$tr$] $\neq 0$,

3.1.7.1 Calculer TrajetVide[$tr$][$Ca^{\epsilon}$]. C'est le temps nécessaire au camion pour parcourir la distance entre Emplacement[$Ca^{\epsilon}$] et Départ[$tr$] en partant à la date DateLibre[$Ca^{\epsilon}$]

3.1.7.2 Calculer TrajetPlein[$tr$][$Ca^{\epsilon}$]. C'est le temps nécessaire au camion $Ca^{\epsilon}$ pour parcourir la distance entre le site du Départ[$tr$] et le site d'Arrivée[$tr$] à la date de départ DateLibre[$Ca^{\epsilon}$]+TrajetVide[$tr$][$Ca^{\epsilon}$]

3.1.7.3 Calculer FinTrajet[$tr$][$Ca^{\epsilon}$] = DateLibre[$Ca^{\epsilon}$] + TrajetVide[$tr$][$Ca^{\epsilon}$] + TrajetPlein[$tr$][$Ca^{\epsilon}$]

*Remarque générale pour les trois heuristiques :* Si deux camions (ou plus) peuvent accomplir un trajet dans les mêmes délais, le camion qui peut s'acquitter du maximum de déplacements sur ce trajet est choisi.

## 3.4 Test des heuristiques

Afin de tester les différentes heuristiques proposées, nous les mettons devant les mêmes situations. Nous observons ensuite leur efficacité face à ces situations, et nous analysons les résultats.

### 3.4.1 Description du programme

Pour que les situations ne soient pas biaisées, nous engendrons aléatoirement un nombre de sites $K$, K étant compris entre 2 et 10, ainsi que leurs positions dans l'espace à trois dimensions, tout en respectant l'inégalité triangulaire. En fait, la position de chaque site est générée aléatoirement, soit respectivement ses coordonnées $x$, $y$, et $z$, (les valeurs de $x$, $y$, et $z$ sont comprises ent res 0 et 10). En fonction de ces points, les distances $D_{ij}$ sont calculées par la formule :

$$D_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \qquad \forall i, j = 1, \cdots, K$$

Ensuite les durées nécessaires pour parcourir les distances entre chaque paire de sites sont calculées en se basant sur le fait qu'un camion roule à une vitesse de 50 km/h en moyenne en ville et que chaque période élémentaire $\Delta$ est une période de huit minutes. Le nombre total $N$

de voitures dont nous disposons, autrement dit les voitures qui sont garées dans les différents sites, est aussi généré aléatoirement, ($N$ est compris entre 0 et 100 voitures). La répartition défavorable est déterminée ainsi que la répartition favorable souhaitée. Ces données sont traitées à l'aide d'un programme linéaire et, de cette façon, nous obtenons les déplacements à effectuer afin de rééquilibrer la situation. L'étape suivante consiste à construire la flotte qui va servir au rééquilibrage, autrement dit le nombre de camions ainsi que leurs capacités respectives. Toutes les données sont préparées afin de tester les heuristiques.

### 3.4.2 Avantages et inconvénients des différentes heuristiques

Comme il nous est difficile de trancher et de dire que globalement une heuristique est meilleure qu'une autre, nous les testons dans différentes situations et nous étudions le comportement de chacune. En fait, nous ne pouvons pas nous baser sur la rapidité des différentes heuristiques pour la construction des solutions. Les trois heuristiques sont exécutées en moins d'une seconde. Le critère dominant de mesure est le temps total nécessaire pour le rééquilibrage en nombre de périodes élémentaires.

1. **Le nombre de déplacements est nettement inférieur à la somme de places disponibles dans les différents camions.** Dans cette situation, nous soulignons que la flotte chargée du rééquilibrage est constituée de plus d'un camion. Les camions de la flotte n'ont pas la même capacité, mais la capacité maximale d'un camion de cette flotte ne dépasse pas 4 voitures. Ceci sous entend que la flotte chargée du rééquilibrage est grande. Le nombre de stations et les durées des trajets entre les stations ne sont pas les mêmes pour les différents exemples listés ci-dessous puisque ceux-la sont générés aléatoirement. Dans le tableau 3.1, nous avons un échantillon des résultats obtenus. Nous observons que pour cette situation, toutes les heuristiques donnent une solution équivalente.

TAB. 3.1: *Résultats obtenus pour le cas 1.*

| | Capacité totale des camions | Nombre de déplacements à effectuer | Temps total nécessaire pour le rééquilibrage en nombre de périodes élémentaires | | |
|---|---|---|---|---|---|
| | | | Heuristique 1 | Heuristique 2 | Heuristique 3 |
| 1 | 33 | 6 | 1 | 1 | 1 |
| 2 | 22 | 2 | 2 | 2 | 2 |
| 3 | 17 | 16 | 4 | 4 | 4 |
| 4 | 12 | 2 | 2 | 2 | 2 |
| 5 | 14 | 4 | 1 | 1 | 1 |
| 6 | 20 | 5 | 1 | 1 | 1 |
| 7 | 22 | 11 | 4 | 4 | 4 |
| 8 | 16 | 8 | 2 | 2 | 2 |

2. **Un seul camion est chargé du rééquilibrage.** Un seul camion se charge des déplacements calculés par le Simplexe. La capacité de celui-ci diffère dans chacun des

exemples ci-dessous, (voir le tableau 3.2). Nous avons remarqué que l'heuristique 3 était la moins bien adaptée à cette situation.

TAB. 3.2: *Résultats obtenus pour le cas 2.*

| | Capacité du camion | Nombre de déplacements à effectuer | Temps total nécessaire pour le rééquilibrage en nombre de périodes élémentaires | | |
|---|---|---|---|---|---|
| | | | Heuristique 1 | Heuristique 2 | Heuristique 3 |
| 1 | 7 | 9 | 6 | 6 | 9 |
| 2 | 7 | 30 | 25 | 25 | 27 |
| 3 | 3 | 46 | 48 | 48 | 50 |
| 4 | 1 | 36 | 87 | 87 | 93 |
| 5 | 1 | 16 | 36 | 36 | 39 |
| 6 | 1 | 46 | 142 | 142 | 152 |
| 7 | 5 | 19 | 11 | 11 | 13 |

3. **Un seul site excédentaire.** Autrement dit, parmi tous les sites du système, un seul fourni les voitures aux autres. Le tableau 3.3 regroupe les résultats que l'on a obtenu pour cette situation. Nous avons remarqué que l'heuristique 2 était la mieux adaptée à cette situation.

TAB. 3.3: *Résultats obtenus pour le cas 3.*

| | Nombre de camions | Nombre de déplacements à effectuer | Temps total nécessaire pour le rééquilibrage en nombre de périodes élémentaires | | |
|---|---|---|---|---|---|
| | | | Heuristique 1 | Heuristique 2 | Heuristique 3 |
| 1 | 3 de capacité 2 | 38 | 15 | 14 | 16 |
| 2 | 1 de capacité 7 | 21 | 10 | 9 | 10 |
| 3 | 1 de capacité 4 | 8 | 14 | 13 | 15 |
| 4 | 1 de capacité 7 | 13 | 14 | 12 | 14 |
| 5 | 2 de capacité 3 | 9 | 7 | 7 | 8 |
| 6 | 7 de capacité 4 | 36 | 4 | 4 | 5 |
| 7 | 2 de capacité 2 | 20 | 19 | 19 | 18 |
| 8 | 5 de capacité 4 | 85 | 19 | 19 | 18 |
| 9 | 2 de capacité 2<br>1 de capacité 3 | 27 | 10 | 12 | 11 |

L'optique de l'heuristique 2 est de transporter le maximum de voitures sans repasser par le site fournisseur si ceci est plus rapide et bien sûr si la capacité du camion le permet.

4. **Un seul site déficitaire.** Un seul site du système sera en rupture de voitures si les autres ne lui en fournissent pas. Le tableau 3.4 résume la situation. En fait, nous observons

que les différentes heuristiques sont à peu près équivalentes, et que ce sont uniquement les positions initiales des camions qui affectent le résultat final.

TAB. 3.4: *Résultats obtenus pour le cas 4.*

| | Nombre de camions | Nombre de déplacements à effectuer | Temps total nécessaire pour le rééquilibrage en nombre de périodes élémentaires | | |
|---|---|---|---|---|---|
| | | | Heuristique 1 | Heuristique 2 | Heuristique 3 |
| 1 | 1 de capacité 4<br>1 de capacité 7 | 25 | 14 | 14 | 16 |
| 2 | 1 de capacité 2<br>5 de capacité 4 | 57 | 12 | 12 | 12 |
| 3 | 5 de capacité 4 | 67 | 9 | 9 | 9 |
| 4 | 3 de capacité 4<br>4 de capacité 3 | 51 | 10 | 10 | 10 |
| 5 | 1 de capacité 5<br>1 de capacité 2 | 38 | 22 | 20 | 22 |
| 6 | 1 de capacité 7 | 69 | 46 | 44 | 45 |
| 7 | 1 de capacité 3<br>1 de capacité 1 | 4 | 6 | 6 | 4 |
| 8 | 1 de capacité 1<br>1 de capacité 6 | 10 | 6 | 6 | 8 |

Des exemples précédents, nous déduisons que la position initiale des camions joue un rôle important sur les différents résultats des heuristiques.

## 3.5 Conclusion

A la fin de ce chapitre, nous avons fait le tour du problème de rééquilibrage en répondant aux questions *Quand, Comment, et Pourquoi* on effectue un rééquilibrage. Premièrement nous avons défini le moment où un rééquilibrage doit être lancé. Nous avons enchaîné sur le choix de la répartition qui garantiera le bon fonctionnement du système. Ensuite nous avons déterminé les déplacements de voitures à effectuer pour atteindre cette répartition, et finalement nous avons proposé plusieurs méthodes pour l'accomplir.

# Chapitre 4

# *Rééquilibrage : Les trajets sont cycliques*

## Résumé

*Ce chapitre présente une solution pour résoudre le problème de la redistribution des voitures dans les sites, différente des solutions présentées dans les chapitres précédents. Les camions, chargés de la redistribution des voitures, visitent cycliquement les différents sites du système. Une décision est à prendre lorsqu'un camion arrive dans un site. En fait, on a le choix à ce moment là entre plusieurs actions : soit charger des voitures dans le camion, soit décharger des voitures du camion, soit tout simplement continuer son chemin sans charger ni décharger.*

*Mots-clés : Simulation, Transport, Programmation Dynamique.*

## Sommaire

## 4.1 Introduction.

Dans ce chapitre, nous proposons une approche différente des précédentes pour la redistribution des voitures dans les sites. Les camions qui sont chargés du rééquilibrage fonctionnent en permanence. Ces camions parcourent les différents sites du système, et selon les besoins de chacun des sites, chargent ou déchargent des voitures. La différence majeure entre ce procédé de rééquilibrage et ceux que nous avons détaillés précédemment est que celui-ci n'est pas déclenché pour répondre à une situation défavorable. Le camion suit le chemin qui lui est attribué (ce trajet est spécifique et cyclique) et, s'il peut améliorer la situation actuelle, entreprend une action. Le camion est en service continu, autrement dit sans aucune interruption. Lorsque le camion arrive dans un site, celui-ci se trouve dans une des situations suivantes. Le site est (ou sera bientôt) en rupture de voitures; dans ce cas le camion décharge des voitures dans le site. La seconde situation est celle où le site a trop de voitures et, par conséquent, un encombrement dans le site peut se produire; dans ce cas, si la capacité du camion le permet, les voitures en surplus sont chargées dans le camion. Enfin, si le site contient un nombre acceptable de voitures, il n'a besoin ni de charger, ni de décharger des voitures, et peut continuer sa route. Avec cette méthode de rééquilibrage, les questions posées dans le premier chapitre de cette partie ne se posent plus. Ces questions étaient : *Quel est le trajet du moyen utilisé pour le rééquilibrage?* et *Quand un dispositif de rééquilibrage doit-il être lancé?* Par contre, dans ce chapitre, nous répondons à la question suivante *Combien de voitures doivent-être chargées ou déchargées lorsque le camion arrive dans un site?* En fait, dans ce chapitre, nous proposons deux solutions différentes : la première utilise la simulation, et la seconde utilise la programmation dynamique.

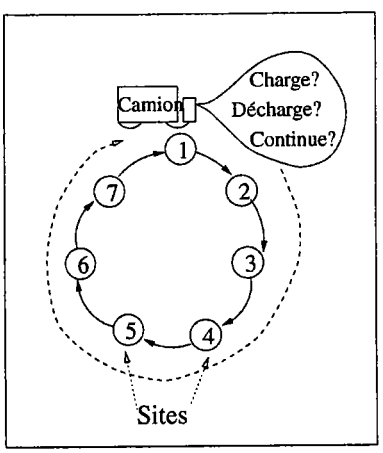



FIG. 4.1: *Rééquilibrage : les trajets sont cycliques.*

## 4.2 Méthode basée sur la simulation

Le rééquilibrage a lieu par l'intermédiaire d'un camion qui visite successivement et en permanence tous les sites. Son parcours est cyclique. Lorsque le camion passe dans un site, il a le choix entre trois attitudes : soit il décharge des voitures dans le site, soit il charge

des voitures dans le camion, soit il ne fait rien. Chaque camion suit un trajet prédéfini. La durée de ce trajet est constante. Autrement dit, le camion $c$ visite le site $i$ toutes les $x$ périodes élémentaires $\Delta$. Quand un camion arrive dans un site, la décision de chargement ou de déchargement de voitures prend en considération tout le cycle suivant. Autrement dit la décision doit tenter d'optimiser un cycle complet. Dans la suite de cette section, nous allons procéder de la manière suivante :

1. Déterminer le cycle du camion.
2. Déterminer les bornes inférieure et supérieure pour chaque site.

### 4.2.1 Cycle du camion

Prenons la situation où un seul camion est chargé de la redistribution des voitures dans les différents sites. Ce camion visite tous les sites du système les uns après les autres. Parmi tous les cycles hamiltoniens possibles, nous recherchons le plus court afin d'assurer des visites plus fréquentes du camion, dans les sites. Nous avons donc affaire à un problème du type *voyageur de commerce*. Nous avons opté pour une heuristique gloutonne (voir 1.3.1 de la partie 2) pour la détermination de ce cycle. En fait, l'idée de l'algorithme est de choisir un site quelconque comme point de départ et, à partir de celui-ci, construire petit à petit le cycle en minimisant le temps global du cycle, et ceci en choisissant, à chaque étape, le site le plus proche.

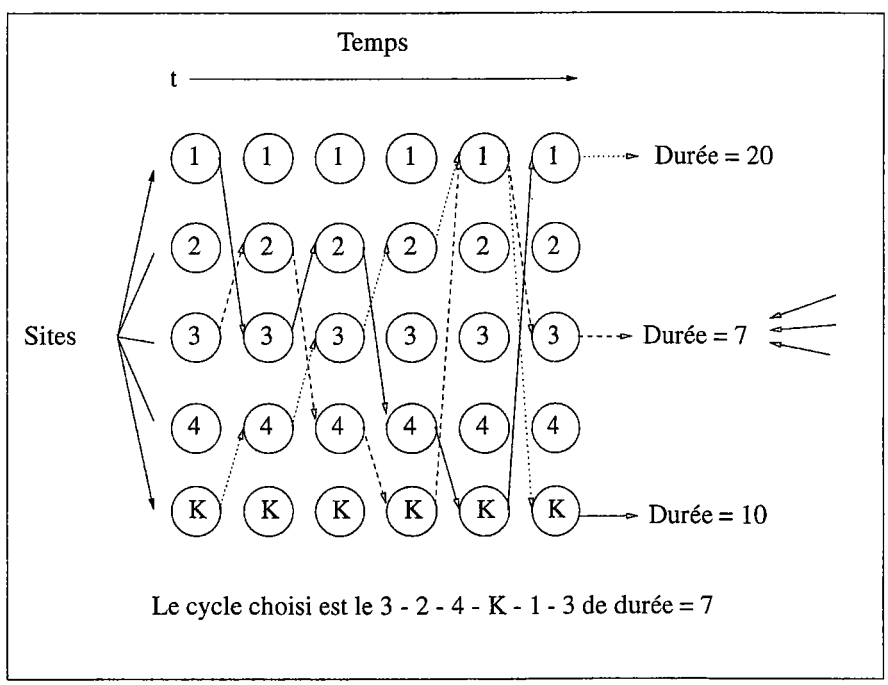



FIG. 4.2: *Les différents cycles possibles en partant des sites 1, 3 et K.*

La figure 4.2 illustre le fonctionnement de l'algorithme qui détermine le cycle du camion. Le camion se trouve dans le site initial $i$ à l'instant $t$, il visite le site qui est le plus proche parmi tous ceux qu'il n'a pas encore visité, l'atteint à l'instant $t + x\Delta$, et ainsi de suite. A

chaque étape, un site est choisi parmi les sites non visités de façon à minimiser la durée totale de la tournée. L'heuristique est testée avec des sites initiaux différents, et parmi toutes ces tournées construites, la plus rapide est choisie. L'algorithme est détaillé ci-dessous sachant que $K$ est le nombre total de sites :

**Algorithme** *Détermination du cycle du camion.*

DuréeMaxi = une journée.
Pour $i = 1$ jusqu'à ce que $i = K$
  deb = $i$.
  SiteInitial = deb.
  Durée = 0.
  Pour $j = 1$ jusqu'à ce que $j = K - 1$
    fin = le site le plus proche de deb.
    Durée = Durée + Distance entre deb et fin.
    deb = fin.
  Durée = Durée + Distance entre deb et SiteInitial
  Si DuréeMaxi > Durée
    DuréeMaxi = Durée.

Bien entendu nous aurions pu utiliser un des nombreux algorithmes disponibles dans la littérature.

### 4.2.2 Seuils de décision

Le but de cette procédure est de déterminer un intervalle inclus dans $[0, E_i]$, où $E_i$ est la capacité du site $i$. Si, à l'arrivée du camion, le site comporte un nombre de voitures inférieur

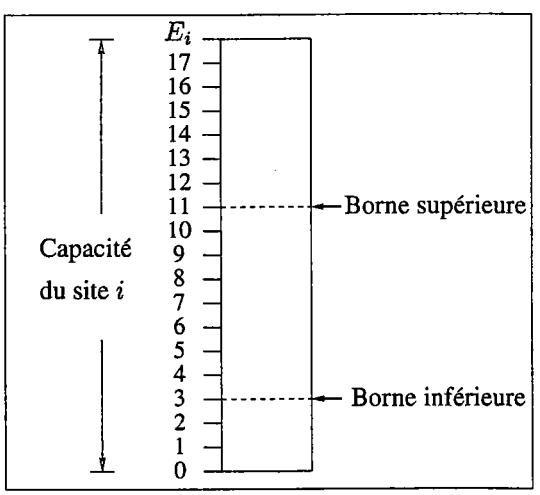



FIG. 4.3: *Les bornes calculées par simulation.*

à la borne inférieure de cet intervalle, des voitures seront déchargées (si possible) jusqu'à ramener le nombre de voitures présentes dans le site à la borne inférieure. Si le nombre de voitures dans le site est compris dans l'intervalle, aucune action n'est entreprise. Si le

nombre de voitures est supérieur à la borne supérieure de l'intervalle, alors le camion charge les voitures en excès si la capacité disponible le lui permet. La *borne inférieure* indique le nombre minimal de voitures nécessaires pour répondre aux demandes prévisionnelles avec un taux de satisfaction $\mu$ pendant un cycle du camion. La *borne supérieure* indique le nombre maximal de voitures présentes dans le site qui n'engendre pas, avec une probabilité $\mu$, un encombrement dans le site durant un cycle du camion.

Afin de calculer les deux bornes, nous suivons la même procédure que celle utilisée lors du calcul des états favorables (voir la section 1.4.2 de cette partie). Le même calcul est effectué pour chacun des sites du système, mais par simplification et pour faciliter l'explication de ce paragraphe, nous nous limitons à la description du calcul pour un seul site $i$. Nous effectuons $Z$ simulations, celles-ci commencent à la période $t$ et se terminent à la période $t + h$. L'horizon $h$ est égal au nombre de périodes élémentaires $\Delta$ nécessaires au camion pour effectuer son cycle, c'est à dire revenir à son point de départ qui est le site considéré. Pour chacune de ces simulations, nous reportons dans un tableau l'intervalle obtenu $[m_i, s_i]$.

Rappel de la section 1.4.2 : $m_i$ représente le nombre minimal de voitures qui, présentes dans le site $i$ à la période $t$, garantit la satisfaction des clients qui se présentent dans le site (pas de ruptures de voitures) pendant l'horizon déterminé. Quant à $s_i$, c'est le nombre maximal de voitures atteint dans le site considéré en supposant qu'il y ait eu initialement $m_i$ voitures dans le site $i$ au début de la période $t$.

Pour que $m_i$ et $s_i$ soient reportés dans le tableau 4.4, ces valeurs doivent être inférieures à la capacité du site $E_i$. De cette façon, au bout des $Z$ simulations nous obtenons le tableau 4.4 qui représente les différents $g^t(i, n)$. $g^t_{i,n}$ est le nombre de simulations pour lesquelles $n$ voitures se situaient dans l'intervalle $[m_i, s_i]$.

Site $i$

| Nb. de Voitures → | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | $E_i$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $g^t_{(i,n)}$ → | 0 | 10 | 160 | 590 | 803 | 890 | 1000 | 1000 | 957 | 790 | 400 |

$Z = 1000$

FIG. 4.4: *Illustration de l'algorithme pour la détermination des bornes.*

L'étape suivante consiste à calculer les différents $p^t_{(i,n)}$ qui représentent les probabilités de bon fonctionnement. $p^t_{(i,n)}$ indique la probabilité qu'il n'y ait ni rupture, ni encombrement tout le long de l'horizon $h$ dans le site $i$ si $n$ voitures s'y trouvaient (initialement) au début de la période $t$.

$$p^t_{(i,n)} = \frac{g^t_{(i,n)}}{Z} \qquad \forall n = 0, \cdots, Min[E_i, N]$$

*Remarque : $N$ est le nombre total de voitures.*

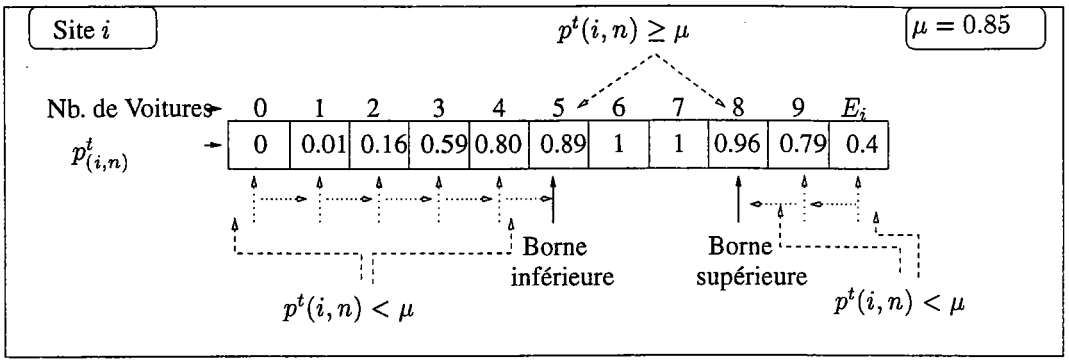



FIG. 4.5: *Le tableau des différents* $p^t(i,n)$.

Afin d'obtenir la borne inférieure pour un site $i$, nous testons si $p^t_{(i,n)} \geq \mu$. Nous commençons nos tests avec $n = 0$. Si le test est négatif, nous incrémentons la valeur de $n$, sachant que $n \in [0, E_i]$. Pour la détermination de la borne supérieure, nous effectuons le même test mais en commençant avec $n = E_i$. Si le test est négatif nous décrémentons la valeur de $n$. Ce paragraphe est expliqué par la figure 4.5. Ainsi, pour chacun des sites, nous obtenons les deux bornes supérieure et inférieure. Lorsqu'un camion arrive dans un site, il compare le nombre actuel de voitures présentes dans le site aux bornes calculées et, suivant le résultat, prend une décision.

## 4.3 Méthode basée sur la programmation dynamique

Notre but est de satisfaire les clients qui se présentent dans un site. Un client est satisfait lorsqu'il se présente dans un site pour prendre une voiture, s'il en trouve une de disponible, ou, à la rigueur, s'il n'attend qu'un petit moment avant d'en obtenir une. Dans cette section, nous n'utilisons plus la notion de bornes inférieures et supérieures décrite dans la section précédente. Lorsqu'un camion arrive dans un site, nous comparons le nombre de voitures présentes dans le site à l'espérance du nombre de voitures dans le site lors du prochain passage du camion. En fonction du résultat de cette comparaison, le camion charge ou décharge des voitures, ou bien, tout simplement continue sa route sans entreprendre d'action.

Dans cette section, nous allons suivre les étapes suivantes :

1. Déterminer le cycle du camion.
2. Les différentes étapes du processus de décision lorsqu'un camion arrive dans un site $i$ sont les suivantes:
   a. Déterminer les probabilités du flux entrant dans le site $i$.
   b. Calculer l'espérance $E[V]$ du nombre de voitures dans le site $i$.

**Les données dont nous disposons sont :**

- Le système est constitué de $K$ sites, et d'un nombre $V$ de voitures.

- Chaque site contient un nombre $E_i$ de places de parking.
- La journée est segmentée en un nombre fini de périodes élémentaires $\Delta$.
- Pour chaque période élémentaire et pour chaque site i, nous connaissons la borne $b_{ij}^t$ qui est la demande maximale vers chacun des sites.
- Nous possédons les probabilités de demandes $p_{ij}^t(n)$ dans chaque site pendant chacune des périodes élémentaires. $p_{ij}^t(n)$ est la probabilité que $n$ clients se présentent dans le site $i$ pendant la période élémentaire $t$ afin d'aller au site $j$, sachant que $n \in [0, b_{ij}^t]$, et $i, j \in [1, K]$
- La distance entre chaque couple de sites, donnée en nombre de périodes élémentaires, est connue.
- Nous connaissons le temps $r_{ij}$, donné en nombre de périodes élémentaires, nécessaire à un client pour parcourir la distance entre les deux sites $i$ et $j$ en partant du site $i$.
- Un seul camion est chargé de la redistribution des voitures dans les différents sites. Ce camion a un trajet prédéfini et cyclique. Toutes les $x$ périodes élémentaires, le camion passe dans le site $i$. Le temps, donné en nombre de périodes élémentaires, nécessaire au camion pour parcourir la distance qui sépare le site $i$ du site $j$ est $d_{ij}$. La capacité du camion est fixe. Le temps de chargement des voitures dans le camion ainsi que le temps de déchargement des voitures dans le site sont négligeables.
- La répartition initiale des voitures dans les différents sites est précisée.
- L'emplacement initial du camion chargé de la redistribution est fourni.

### 4.3.1 Cycle du camion

Nous suivons les mêmes étapes que celles décrites dans la section 4.2.1 pour déterminer le trajet du camion qui est cyclique et de durée connue. La durée du cycle du camion est donnée en nombre de périodes élémentaires.

### 4.3.2 Les différentes étapes du processus de décision

Lorsqu'un camion arrive dans un site $i$, celui-ci doit connaître l'action qu'il doit entreprendre qui est soit de charger des voitures, soit de décharger des voitures, ou soit tout simplement de continuer sa route. En fait, la décision dépend de deux paramètres qui sont l'espérance du nombre de voitures dans le site lors du prochain passage du camion dans celui-ci et le nombre de voitures qui sont effectivement présentes dans le site.

#### 4.3.2.1 Probabilités du flux entrant dans un site

La probabilité que le flux entrant dans le site $i$ soit de $n$ voitures, notée $\mathcal{P}_i^t(n)$, dépend des probabilités que des voitures arrivent (dans le) et partent du site. Celle-ci est formulée comme

suit :

$$\mathcal{P}_i^t(n) = \sum_{m=max(0,n)}^{\sum_{j=1}^{K}(b_{ji}^{t-r_{ji}})} p_{\rightarrow i}^{t-r_{ji}}(m) * p_{i\rightarrow}^{t}(n-m) \tag{4.1}$$

$$\text{sachant que } \sum_{j=1}^{K}(b_{ji}^{t-r_{ji}}) \geq n \geq -\sum_{j=1}^{K}(b_{ij}^{t})$$

L'équation 4.1 indique que nous devons commencer par le calcul des différentes probabilités d'arrivées dans le site $i$, noté $p_{\rightarrow i}^{t-r_{ji}}(n)$, puis ensuite enchaîner avec le calcul des différentes probabilités de départs du site $i$, noté $p_{i\rightarrow}^{t}(n)$, afin de déterminer les probabilités du flux entrant dans le site. *Remarque :* le flux entrant peut être négatif puisque $\sum_{j=1}^{K}(b_{ji}^{t-r_{ji}}) \geq n \geq -\sum_{j=1}^{K}(b_{ij}^{t})$ et il indique alors que le flux sortant est supérieur au flux entrant.

$p_{\rightarrow i}^{t}(n)$ qui est la probabilité que $n$ voitures arrivent dans le site $i$ à la période $t$ en provenance de tous les sites du système est formulée comme suit :

$$p_{i\rightarrow}^{t}(n) = [\sum_{n_1=0}^{min(n,b_{i1}^{t-r_{ji}})} (p_{i1}^{t-r_{ji}}(n_1))[\sum_{n_2=0}^{min(n-n_1,b_{i2}^{t-r_{ji}})} (p_{i2}^{t-r_{ji}}(n_2))\cdots$$

$$[\sum_{n_K=0}^{min(n-\sum_{j=0}^{K-1}(n_j),b_{iK}^{t-r_{ji}})} (p_{iK}^{t-r_{ji}}(n_K))]\cdots]]$$

$$\text{ou encore: } p_{i\rightarrow}^{t}(n) = \sum_{\{n_j\}/\sum_{j=1}^{K}(n_j)=n} \prod_{j=1}^{K} p_{ij}^{t-r_{ji}}(n_j) \tag{4.2}$$

$p_{i\rightarrow}^{t}(n)$ qui est la probabilité que $n$ voitures partent du site $i$ à la période $t$ quelque soit les destinations est formulée comme suit :

$$p_{i\rightarrow}^{t}(n) = [\sum_{n_1=0}^{min(n,b_{i1}^{t})} (p_{i1}^{t}(n_1))[\sum_{n_2=0}^{min(n-n_1,b_{i2}^{t})} (p_{i2}^{t}(n_2))\cdots[\sum_{n_K=0}^{min(n-\sum_{j=0}^{K-1}(n_j),b_{iK}^{t})} (p_{iK}^{t}(n_K))]\cdots]]$$

$$\text{ou encore: } p_{i\rightarrow}^{t}(n) = \sum_{\{n_j\}/\sum_{j=1}^{K}(n_j)=n} \prod_{j=1}^{K} p_{ij}^{t}(n_j) \tag{4.3}$$

*Détails de l'algorithme développé pour le calcul des probabilités de départs $p^t_{i\rightarrow}(n)$ du site $i$*

Comme données, nous connaissons les différentes probabilités de demandes du site $i$ vers chacun des autres sites du système. Nous utilisons l'équation 4.3 afin de calculer les probabilités de départs $p^t_{i\rightarrow}(n)$ du site $i$. *Remarque :* $n \in [0, \sum_{j=0}^{K} b^t_{ij}]$.

Pour simplifier l'explication de l'algorithme développé pour le calcul des probabilités de départs, nous avons recours à l'exemple numérique suivant :

*Données de l'exemple :*

Le système comprend 6 sites (K=6). Le camion arrive dans le site 1 au début de la période $t$. Les différents $b^t_{1j}$ sont respectivement 0, 2, 4, 3, 5, et 2 et les différentes probabilités de demandes dans le site 1 vers chacun des autres sites sont listées dans le tableau 4.1.

TAB. 4.1: *Les probabilités de demandes dans le site 1 pour chacun des sites du système*

| Site 1 | Site 2 | Site 3 | Site 4 | Site 5 | Site 6 |
|---|---|---|---|---|---|
| $p^t_{11}(0)$ | $p^t_{12}(0)$ | $p^t_{13}(0)$ | $p^t_{14}(0)$ | $p^t_{15}(0)$ | $p^t_{16}(0)$ |
| | $p^t_{12}(1)$ | $p^t_{13}(1)$ | $p^t_{14}(1)$ | $p^t_{15}(1)$ | $p^t_{16}(1)$ |
| | $p^t_{12}(2)$ | $p^t_{13}(2)$ | $p^t_{14}(2)$ | $p^t_{15}(2)$ | $p^t_{16}(2)$ |
| | | $p^t_{13}(3)$ | $p^t_{14}(3)$ | $p^t_{15}(3)$ | |
| | | $p^t_{13}(4)$ | | $p^t_{15}(4)$ | |
| | | | | $p^t_{15}(5)$ | |

*Algorithme :*

La première étape de l'algorithme consiste à lister toutes les possibilités de demandes dans le site tout en respectant les différents $b^t_{ij}$ (voir le tableau 4.2). Prenons par exemple la seconde entrée du tableau 4.2, celle-ci indique qu'une seule demande est formulée dans le site 1 et que c'est pour aller au site 2.

TAB. 4.2: *Les départs possibles du site 1 vers les autres sites*

| | Site 1 | Site 2 | Site 3 | Site 4 | Site 5 | Site 6 |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 | 0 | 0 |
| 3 | 0 | 2 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 1 | 0 | 0 | 0 |
| 5 | 0 | 1 | 1 | 0 | 0 | 0 |
| 6 | 0 | 2 | 1 | 0 | 0 | 0 |
| 7 | 0 | 0 | 2 | 0 | 0 | 0 |
| 8 | 0 | 1 | 2 | 0 | 0 | 0 |

*suite page suivante*

| *suite de la page précédente* | | | | | | |
|---|---|---|---|---|---|---|
| | Site 1 | Site 2 | Site 3 | Site 4 | Site 5 | Site 6 |
| ... | ... | ... | ... | ... | ... | ... |
| x | 0 | 2 | 4 | 3 | 5 | 2 |

Pour chaque entrée dans le tableau 4.2, nous calculons partiellement la probabilité de départ correspondante.

Prenons, par exemple, le calcul de la probabilité qu'il y ait un seul départ du site 1, nous procédons en plusieurs étapes :

1. La première étape, appelée $\mathcal{A}$, est calculée lors de la seconde entrée du tableau, ainsi :
$$\mathcal{A} = p^t_{11}(0) * p^t_{12}(1) * p^t_{13}(0) * p^t_{14}(0) * p^t_{15}(0) * p^t_{16}(0)$$
2. Lors de la quatrième entrée du tableau, nous additionnons à $\mathcal{A}$ le produit des probabilités correspondantes, nous obtenons alors :
$$p^t_{1\rightarrow}(1) = \mathcal{A} + p^t_{11}(0) * p^t_{12}(0) * p^t_{13}(1) * p^t_{14}(0) * p^t_{15}(0) * p^t_{16}(0)$$
3. et ainsi de suite.

La probabilité que l'on ait aucun départ du site est calculée lors de la première combinaison du tableau 4.2, ainsi :

$$p^t_{1\rightarrow}(0) = p^t_{11}(0) * p^t_{12}(0) * p^t_{13}(0) * p^t_{14}(0) * p^t_{15}(0) * p^t_{16}(0)$$

La probabilité que l'on ait un maximum de départs du site, autrement dit $n = \sum_{j=0}^{K} b^t_{ij}$, est calculée lors de la dernière entrée du tableau 4.2. Celle-ci est égale à :

$$p^t_{i\rightarrow}(16) = p^t_{11}(0) * p^t_{12}(2) * p^t_{13}(4) * p^t_{14}(3) * p^t_{15}(5) * p^t_{16}(2)$$

Le même algorithme est utilisé pour le calcul des différentes probabilités d'arrivées $p^t_{\rightarrow i}(n)$ dans le site $i$.

Ayant calculé les différentes probabilités de départs et d'arrivées, nous appliquons l'équation 4.1, et nous obtenons les probabilités du flux entrant dans le site lors de la période $t$.

Pour calculer les probabilités du flux entrant dans un site lors du prochain passage du camion dans le site, nous appliquons plusieurs fois l'équation 4.4. Le nombre de fois où l'équation 4.4 est appliquée dépend du nombre de périodes élémentaires nécessaires au camion pour effectuer son cycle.

$$\mathcal{P}_i^{t\rightarrow t+1}(n) = \sum_{n_1=-\sum_{j=1}^{K}(b^t_{ij})}^{\sum_{j=1}^{K}(b_{ji}^{t-r_{ji}})} \mathcal{P}_i^t(n_1) * \mathcal{P}_i^{t+1}(n-n_1) \tag{4.4}$$

sachant que : $\sum_{j=1}^{K}(b_{ji}^{t-r_{ji}}) + \sum_{j=1}^{K}(b_{ji}^{t+1-r_{ji}}) \geq n \geq -(\sum_{j=1}^{K}(b^t_{ij}) + \sum_{j=1}^{K}(b^{t+1}_{ij}))$

#### 4.3.2.2 Espérance du nombre de voitures dans un site

L'étape suivante consiste à calculer l'espérance $E[V]$ du nombre de voitures présentes dans le site $i$ lors du prochain passage du camion dans le site. Nous appliquons l'équation 4.5.

$$E[V] = \sum_{x=-\sum_{j=1}^{K}(b_{ij}^{t\rightarrow t+cycle})}^{\sum_{j=1}^{K}(b_{ji}^{t-r_{ji}\rightarrow t+cycle-r_{ji}})} x\mathcal{P}_i^{t\rightarrow t+cycle}(x) \tag{4.5}$$

Par hypothèse, nous connaissons $v_i^t$ qui est le nombre de voitures présentes dans le site $i$ lors du passage du camion dans le site à la période $t$. L'étape suivante consiste à estimer $S_i^{t\rightarrow t+cycle}$ qui est le nombre de voitures qui seront présentes dans le site lors du prochain passage du camion, autrement dit, après une durée équivalente à la durée du cycle du camion.

$$S_i^{t\rightarrow t+cycle} = E[V] + v_i^t$$

Lorsque $S_i^{t\rightarrow t+cycle}$ est inférieur à zéro, cela indique qu'une rupture a lieu dans le site pendant que le camion visite les autres sites du système. Afin d'éviter cette rupture, le camion décharge des voitures dans le site. Par contre, si le nombre $S_i^{t\rightarrow t+cycle}$ est supérieur à la capacité du site, ceci sous-entend qu'un encombrement peut avoir lieu dans le site. Le camion, si sa capacité le lui permet, charge des voitures pour empêcher cet encombrement. Lorsque le camion s'est acquitté de sa mission dans un site, celui-ci se dirige vers le site suivant et les mêmes calculs sont effectués pour le site $i+1$. De plus, compte tenu du nombre de voitures présentes dans le camion et du nombre de voitures présentes dans le site, une action est décidée.

## 4.4 Partitionnement du cycle hamiltonien en plusieurs cycles

Dans ce chapitre, par hypothèse, tous les camions chargés de la redistribution des voitures empruntent un seul cycle hamiltonien. Cette section a pour but de proposer des heuristiques afin de partager ce cycle hamiltonien en plusieurs cycles dans le cas où plusieurs camions sont utilisés.

### 4.4.1 Heuristique 1

La première étape est la construction du cycle hamiltonien le plus court comme le décrit la section 4.2.1. Par hypothèse, nous possédons plusieurs camions et nous voulons leur attribuer des cycles différents à raison d'un camion par cycle. Pour cela, nous divisons le nombre de sites par le nombre de camions. Ainsi, nous déterminons le nombre moyen de sites visités par chaque camion. Il se peut que le premier camion visite un nombre de sites supérieur aux autres.

L'étape suivante consiste à supprimer les arcs en trop du cycle hamiltonien comme le montre le second shéma de la figure 4.6.

Ensuite, en ajoutant de nouveaux arcs, nous obtenons les cycles souhaités. *Remarque :* on prévoit une connexion entre les cycles afin de pouvoir déplacer une voiture d'un site vers tout autre site du système.

1. Un premier arc est ajouté entre l'extrémité d'un cycle et le site initial du cycle suivant, (exemple : les arcs (7, 1), (3, 4) et (5, 6) de la figure 4.6.(c)).

2. Pour obtenir un cycle, il faut revenir à son point de départ. Pour cela, un arc est ajouté pour obtenir une boucle et relier le dernier site visité du point précédent au site initial du cycle considéré, (exemple : les arcs (1, 6), (4, 1) et (6, 4) de la figure 4.6.(c)).

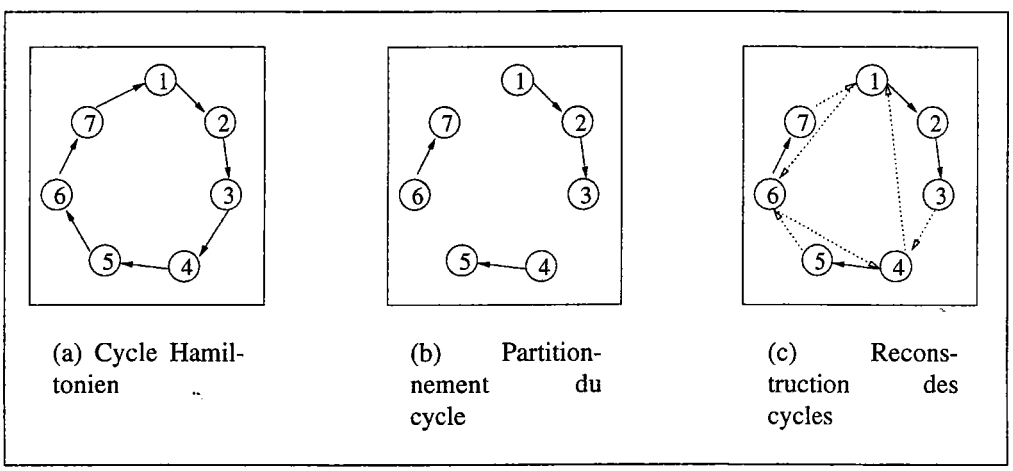



FIG. 4.6: *Illustration de la première heuristique*

*Remarque :* l'intérêt de cette heuristique réside dans le peu de comparaisons à effectuer.

### 4.4.2 Heuristique 2

Cette heuristique partage le cycle hamiltonien initial en plusieurs cycles, et relie les différents cyles obtenus en un site, appelé *site étoile*. Le site étoile est le point de connexion entre les différents cycles et facilite ainsi les passages de voitures entre les différents cycles.

Le site étoile est le site dont la somme des distances qui le relie à chacun des autres sites du système est la plus petite.

### 4.4.3 Heuristique 3

L'objectif de la dernière heuristique présentée est d'obtenir des cycles de durées à peu près équivalentes. Connaissant la durée totale du cycle hamiltonien, et le nombre de camions, nous connaissons la durée moyenne souhaitée pour chaque cycle. Afin de construire les nouveaux cycles, nous suivons les étapes décrites dans la première heuristique.

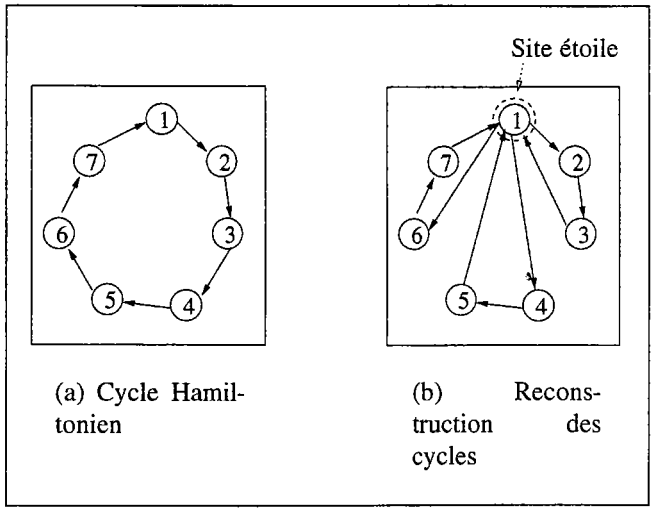



FIG. 4.7: *Illustration de la seconde heuristique*

## 4.5 Conclusion

Avec ce chapitre, nous terminons la partie de ce mémoire consacrée à la redistribution des voitures dans les sites sachant que le but de la redistribution est de pouvoir répondre au maximum de demandes formulées dans les différents sites. Dans la partie suivante, nous traitons le problème de la recharge des voitures électriques.

# Quatrième partie

# Recharge des voitures électriques

# Chapitre 1

# *Détermination du seuil de disponibilité*


## Résumé

*Ce chapitre concerne la gestion de la charge des voitures électriques. Dans chaque site, des décisions sont prises concernant la disponibilité des voitures. On suppose qu'une voiture est disponible quand son niveau de charge est supérieur à un certain seuil. Notre but est d'optimiser ce seuil, c'est à dire de choisir un seuil qui minimise un coût lié à l'insatisfaction des clients.*


*Mots-clés : Logistique, Voitures Electriques, Voitures en Libre Service.*

## Sommaire

## 1.1 Introduction

Notre système est un nouveau mode de transport public. Il s'agit de voitures électriques mises en libre service en complément des transports en commun. Ce système devrait inciter la population à modérer l'utilisation des voitures privées et, par conséquent, remédier à certains problèmes importants tels que les encombrements (bouchons), les places de parking introuvables, et la pollution dans les centres-villes. Initialement, les voitures sont réparties sur plusieurs sites. Les clients se présentent dans ces sites afin d'emprunter une voiture pour une durée limitée. Ils ont la possibilité de la rendre au site de leur choix. Une voiture sera donc utilisée plusieurs fois par différents clients au cours d'une seule journée, d'où l'intérêt de notre système [ISA96]. Notre but est d'obtenir un taux de satisfaction élevé.

Les raisons pour lesquelles le système utilise des voitures électriques sont nombreuses. La plus importante est une raison écologique. L'utilisation de voitures électriques diminue la pollution dans les villes. Effectivement une voiture électrique est silencieuse et non polluante en cours d'utilisation [14t97a]. De plus, une voiture électrique est nerveuse, amusante à conduire, et sa vitesse maximale se situe entre 60 et 80 km/h, ce qui est l'idéal pour une voiture de citadins. Enfin, une voiture électrique est plus économique d'utilisation qu'une voiture à essence. Jusqu'à maintenant, nous avons uniquement cité les avantages d'une voitures électrique, mais elle a également des inconvénients. En effet, une voiture électrique a besoin de huit heures pour recharger complètement ses batteries [14t97b], de même qu'elle a seulement une autonomie d'une centaine de kilomètres environ [Can96]. De plus, de temps en temps, afin de réajuster les jauges, une décharge complète des batteries est nécessaire [Can96].

Pour que notre système ait le succès attendu, nous devons satisfaire les exigences de nos clients. En premier lieu, les clients veulent que des voitures soient disponibles lorsqu'ils en ont besoin. Un client peut attendre, à la rigueur, cinq minutes [Haf97], mais rarement plus. De même, ils exigent que les voitures soient bien entretenues, du point de vue de la propreté et de la mécanique. Enfin, un client n'accepte pas que la voiture utilisée s'arrête au cours d'un trajet, faute d'énergie ou pour cause de panne mécanique.

Il est évident qu'au bout d'un certain nombre de courses, nous pourrons peut-être observer que certains sites contiennent un nombre excessif de voitures, alors que d'autres en manquent; d'où la nécessité d'installer un dispositif de rééquilibrage du système [CHP97]. Ce dispositif a pour rôle de faire en sorte qu'il n'y ait ni rupture ni pléthore de voitures dans les différents sites [CHPS97]. A ce stade de l'étude, nous supposons que le système de rééquilibrage est parfait. Aussitôt qu'une voiture quitte un site, elle est remplacée par une voiture qui vient d'un autre site. Autrement dit, le nombre de voitures présentes dans le site est constant. C'est une façon de séparer le problème de rééquilibrage de celui de l'évaluation du seuil de disponibilité. Comme nous l'avons précédemment noté, une voiture est disponible quand son niveau de charge est supérieur à un certain seuil. Ce chapitre a pour objectif de déterminer ce seuil : c'est un problème qui se situe en amont de celui du rééquilibrage.

## 1.2 Formulation du problème

### 1.2.1 Fonction objectif

Le taux d'insatisfaction des clients dépend du seuil $s$, en dessous duquel les voitures ne sont pas autorisées à quitter le site. Le critère, le taux d'insatisfaction, que nous voulons minimiser est évalué de la façon suivante :

$$O(s) = c_M \, E[M(s)] + c_R \, E[R(s)] \tag{1.1}$$

où :

- $s$ est le seuil que nous voulons déterminer. Les voitures dont le niveau de charge est au-dessus de ce seuil, sont disponibles.
- $M(s)$ est la variable aléatoire qui représente le nombre de clients mécontents présents dans un site. Les valeurs de cette variable aléatoire se situe dans l'intervalle $[0, N]$. Comme chaque voiture qui quitte le site est automatiquement remplacée par une autre voiture, $N$ est constant. Dans la suite de ce chapitre, les valeurs de toute variable discrète représentant un nombre de voitures appartient à l'ensemble $\{0, 1, \cdots, N\}$.
- $E[M(s)]$ est l'espérance de $M(s)$. Le mécontentement d'un client peut être dû à :
  1. L'absence de voitures dans le site, ou à la présence d'un nombre insuffisant de voitures compte tenu de la demande. Ce cas n'est pas considéré dans ce chapitre puisque, comme nous l'avons précisé plus haut, le système de rééquilibrage est supposé parfait.
  2. Le site contient un nombre suffisant de voitures, mais certaines d'entre elles sont insuffisamment chargées pour être mises à la disposition des utilisateurs. Dans ce chapitre, $E[M(s)]$ reflète uniquement le mécontentement des clients dû à cette raison.
- $R(s)$ est la variable aléatoire qui représente le nombre de voitures qui vont tomber en panne d'énergie pendant une course.
- $E[R(s)]$ est l'espérance de $R(s)$. Pour simplifier, dans la suite du chapitre, nous l'appelerons, "espérance de rupture d'énergie". Bien entendu, cette espérance dépend du seuil que nous voulons déterminer.
- $c_M$ est le coût lié à l'insatisfaction d'un client qui ne peut disposer d'une voiture, les voitures situées dans le site étant insuffisamment chargées.
- $c_R$ est le coût lié au fait qu'une voiture tombe en panne en cours d'utilisation pour cause de rupture d'énergie. Quand un client tombe en panne pendant une course, sa course doit lui être remboursée ainsi que le taxi qui viendra le dépanner. De plus, il va falloir remorquer la voiture jusqu'au garage. Enfin, le mécontentement du client a un prix.

Les coûts $c_M$ et $c_R$ doivent être considérés comme des paramètres que l'utilisateur peut faire varier en fonction de ses observations. La forme du critère est logique de notre point de vue, car la décision à prendre est nécessairement un compromis entre le risque d'affecter des voitures insuffisamment chargées, et le mécontentement suscité en refusant aux clients certaintes voitures en recharge dans une station.

L'étape suivante consiste à montrer comment nous évaluons le taux d'insatisfaction des clients.

### 1.2.2 Calcul de l'espérance du nombre de clients mécontents présents dans un site

Nous faisons l'hypothèse qu'une voiture est disponible uniquement si son niveau de charge est supérieur à un seuil $s$ donné. Le niveau de charge $y$ d'une voiture disponible se situe dans l'intervalle $[s, C]$, où $C$ est la charge maximale qu'une voiture électrique peut atteindre.

Dans un site, nous avons deux catégories de voitures : les voitures disponibles et les voitures indisponibles. Nous supposons que la charge des voitures présentes dans un site est uniformément distribuée entre 0 et $C$.
La proportion de voitures disponibles dans un site est donc :

$$B(D) = \frac{C - s}{C}$$

La proportion de voitures indisponibles est par conséquent :

$$B(I) = 1 - B(D) = \frac{s}{C}$$

Nous désignons par $N$ le nombre total de voitures qui se trouvent dans un site. Ce nombre inclut les voitures disponibles et indisponibles.

La probabilité liée à la variable aléatoire $D$ qui représente le nombre de voitures disponibles dans un site est donc donnée par une loi binomiale :

$$P(D = d) = C_N^d \, (B(D))^d \, (1 - B(D))^{N-d}$$

$$= C_N^d \, (\frac{C - s}{C})^d \, (\frac{s}{C})^{N-d}$$

où $$C_N^d = \frac{N!}{d!(N - d)!}$$

L'espérance $E[M(s)]$ du nombre de clients mécontents présents dans un site est l'espérance du nombre de clients insatisfaits du fait de l'immobilisation de certaines voitures. Rappelons que notre hypothèse initiale indique que la procédure de rééquilibrage est parfaitement établie, ce qui implique que le nombre de voitures présentes dans un site est toujours supérieur ou égal aux demandes. Si $k$ est la demande, elle sera toujours inférieure au nombre de

voitures présentes dans le site, $k \leq N$.

$$E[M(s)] = \sum_{k=0}^{N} \sum_{d=0}^{k} P(K = k)\, P(D = d)\, (k - d)$$

posons w = k - d.

$$E[M(s)] = \sum_{k=0}^{N} \sum_{w=0}^{k} P(K = k)\, P(D = k - w)\, w$$

où :

- $K$ est la variable aléatoire qui représente le nombre de demandes qui sont formulées dans un site. Nous supposons que cette probabilité est connue. La valeur de $K$ se situe dans l'intervalle $[0, N]$ puisque le rééquilibrage est parfait.
- $w = k - d$ est le nombre de clients insatisfaits dans un site parce que les voitures qui leur sont destinées ne sont pas suffisamment chargées.

Finalement, l'espérance du nombre de clients mécontents présents dans un site s'écrit :

$$E[M(s)] = \sum_{k=0}^{N} \sum_{w=0}^{k} C_N^{k-w} (\frac{C - s}{C})^{k-w} (\frac{s}{C})^{(N-k+w)}\, P(K = k)\, w \tag{1.2}$$

### 1.2.3 Calcul de l'espérance de rupture d'énergie

Une voiture dont le niveau de charge maximal est $C$ peut parcourir une distance $L$. Supposons qu'une voiture n'ait jamais à parcourir cette distance, mais que la distance parcourue soit toujours inférieure ou égale à $G$, avec $G \leq L$. Autrement dit, afin de parcourir une distance maximale $G$, une voiture a besoin d'un niveau de charge, représenté par la variable aléatoire $Y$, égal à $\frac{CG}{L}$. De même pour parcourir une distance, représentée par la variable aléatoire $X$, le niveau de charge $Y$ nécessaire est égal à :

$$Y = \frac{C}{L} X \tag{1.3}$$

Les valeurs de $X$ se situent dans l'intervalle $[0, G]$.

Une voiture est prête à l'utilisation si son niveau de charge est supérieur à un certain seuil $s$. Une telle voiture est dite disponible. Le niveau de charge de celle-ci se situe obligatoirement entre $s$ et $C$ :

$$s \leq Y \leq C \tag{1.4}$$

Une voiture de charge $Y$ tombe en panne d'énergie si sa charge n'est pas suffisante pour parcourir la distance souhaitée ou, en d'autres termes, si la charge nécessaire pour parcourir cette distance est supérieure à $Y$ :

$$Y < \frac{C}{L}X \quad ou \quad X > \frac{L}{C}Y$$

La probabilité qu'une voiture de charge $Y = y$ connaisse une rupture d'énergie pendant un trajet est égale à la probabilité que la distance à parcourir $X = x$ soit supérieure à la distance que la voiture est effectivement capable de parcourir connaissant sa charge. Nous la calculons comme suit :

$$P(X > \frac{L}{C}y/Y = y) \quad = \quad \int_{\frac{L}{C}y}^{\infty} g_X(x)dx \tag{1.5}$$

où $g_X(x)$ est la densité de probabilité de la variable aléatoire X qui représente la distance à parcourir.

$q_R(s)$ est la probabilité qu'une voiture tombe en panne d'énergie pendant une course. Pour calculer $q_R(s)$, nous considérons chaque niveau de charge $y$ possible pour une voiture disponible, et nous calculons la probabilité que cette voiture tombe en panne, multipliée par la densité de probabilité de la charge. Nous exprimons ceci comme suit :

$$q_R(s) = \int_s^C (P(X > \frac{yL}{C}/Y = y))\, f_Y(y)\, dy \tag{1.6}$$

où $f_Y(y)$ est la densité de probabilité de la variable aléatoire $Y$ qui représente le niveau de charge d'une voiture disponible.

*Remarque :*
Les limites de l'intégrale sont $s$ et $C$, car les valeurs de la variable aléatoire $Y$ qui représente le niveau de charge d'une voiture disponible appartiennent à l'intervalle $[s, C]$ (voir 1.4). Sachant que $\frac{CG}{L}$ est la charge nécessaire pour parcourir la distance maximale $G$, nous reformulons l'équation 1.6 comme suit :

$$q_R(s) = \int_s^{\frac{CG}{L}} (P(X > \frac{yL}{C}/Y = y))\, f_Y(y)\, dy$$

$$+ \int_{\frac{CG}{L}}^{C} (P(X > \frac{yL}{C}/Y = y))\, f_Y(y)\, dy$$

Le second terme de l'équation précédente est nul puisque $P(X > \frac{yL}{C}/Y = y)$ est égal à zéro lorsque $y$ est supérieur à $\frac{CG}{L}$.

L'espérance de rupture d'énergie, comme nous l'avons déjà dit, est l'espérance du nombre de voitures qui tombent en panne d'énergie pendant une course. Nous la formulons comme suit :

$$E[R(s)] = \sum_{v=0}^{N} \sum_{r=0}^{v} P(V = v)\, P(R(s) = r/V = v)\, r \tag{1.7}$$

où $V$ est la variable aléatoire qui représente le nombre de voitures disponibles qui quittent le site.

Quand une voiture quitte un site, alors soit elle termine sa course et arrive à destination sans encombre, soit elle subit une panne d'énergie parce que sa charge était insuffisante.

A l'aide d'une distribution binomiale, nous déterminons la probabilité $P(R(s) = r/V = v)$ d'avoir $r$ des $v$ voitures (attribuées aux clients) qui tombent en panne lors d'une course :

$$P(R(s) = r/V = v) = C_v^r \, (q_R(s))^r \, (1 - q_R(s))^{v-r}$$

Si nous remplaçons $P(R(s) = r/V = v)$ par sa valeur dans l'équation 1.7, nous obtenons :

$$E[R(s)] = \sum_{v=0}^{N} \sum_{r=o}^{v} P(V = v) \, C_v^r (q_R(s)^r)(1 - q_R(s))^{v-r} r$$

Mais : $\sum_{r=0}^{v} C_v^r \, (q_R(s))^r \, (1 - q_R(s))^{v-r} \, r \; = \; v \, q_R(s)$

Par conséquent : $E[R(s)] \; = \; q_R(s) \sum_{v=0}^{N} v \, P(V = v)$

Si : $E[V(s)] = \sum_{v=0}^{N} v \, P(V = v)$ sachant que $E[V(s)]$ est l'espérance du nombre de voitures disponibles qui quittent le site

Finalement nous obtenons :

$$E[R(s)] = q_R(s) \, E[V(s)] \tag{1.8}$$

Nous supposons que le nombre total de voitures présentes dans un site couvre les demandes de $k$ clients. Quelques unes de ces $k$ demandes seront refusées parce que certaines voitures sont indisponibles. L'espérance E[V(s)] du nombre de voitures qui quittent le site peut être exprimée de la manière suivante : *(La démonstration de l'équation 1.9 se trouve en annexe).*

$$E[V(s)] = E[K] - E[M(s)], \tag{1.9}$$

$E[M(s)]$ est calculé en 1.2. L'espérance $E[K]$ du nombre de demandes des clients est égal à :

$$E[K] = \sum_{k=0}^{N} kP(K = k)$$

où $N$ est le nombre maximal de demandes.

En considérant la relation 1.8, nous pouvons donc expliciter $E[R(s)]$, espérance d'être en rupture d'énergie lors d'une course, de la façon suivante :

$$\begin{aligned} E[R(s)] &= q_R(s)\, E[V(s)] \\ &= q_R(s)\, (E[K] - E[M(s)]) \\ E[R(s)] &= q_R(s)\, E[K] - q_R(s)\, E[M(s)] \end{aligned} \tag{1.10}$$

Si nous remplaçons chaque terme par sa valeur, nous obtenons :

$$E[R(s)] = \int_s^C \left( \int_{\frac{vL}{C}}^{\infty} g_X(x)\, dx \right) f_Y(y)\, dy \left[ \sum_k kP(K=k) - \sum_{k=0}^{N} \sum_{w=0}^{k} P(K=k)\, P(D=k-w)\, w \right]$$

### 1.2.4 Expression de la fonction objectif

La fonction objectif introduite en 1.2 est :

$$O(s) = c_M\, E[M(s)] + c_R\, E[R(s)]$$

En tenant compte de la relation 1.8 :

$$O(s) = c_M\, E[M(s)] + c_R\, (q_R(s)\, E[V(s)])$$

et, en considérant l'équation 1.9 :

$$O(s) = c_M\, E[M(s)] + c_R\, (q_R(s)\, (E[K] - E[M(s)]))$$

Finalement :

$$O(s) = c_M\, E[M(s)] + c_R\, q_R(s)\, E[K] - c_R\, q_R(s)\, E[M(s)]$$

$$O(s) = \underbrace{E[M(s)](c_M - c_R\, q_R(s))}_{\alpha(s)} + \underbrace{c_R\, q_R(s)\, E[K]}_{\beta(s)} \tag{1.11}$$

## 1.3 Exemples numériques

Notre objectif est la détermination du seuil qui minimise notre critère O(s). La fonction O(s) est trop complexe pour être analysée. C'est la raison pour laquelle nous utilisons le calcul numérique. Nous allons d'abord considérer un cas particulier dans lequel les différentes probabilités sont uniformément distribuées.

### 1.3.1 Cas particulier

La longueur d'un trajet est inférieure ou égale à $G$. Nous supposons que la densité de probabilité correspondant à cette longueur est uniformément distribuée entre 0 et $G$. Par conséquent la densité de probabilité de la variable aléatoire X qui représente la distance qu'un client peut parcourir est déterminée de la manière suivante :

$$g_X(x) = \begin{cases} \frac{1}{G} & \text{si } x \in [0, G] \\ 0 & \text{Sinon} \end{cases} \tag{1.12}$$

Si nous remplaçons $g_X(x)$ par sa valeur dans l'équation 1.5, nous obtenons :

$$\begin{aligned} P(X > \tfrac{yL}{C}/Y = y) &= \int_{\frac{yL}{C}}^{G} \tfrac{1}{G} dx \\ &= \begin{cases} 1 - \frac{yL}{GC} & \text{si } y \in [s, \frac{GC}{L}] \\ 0 & \text{Sinon} \end{cases} \end{aligned} \tag{1.13}$$

*Remarque :* La limite supérieure de l'intégrale est $G$ qui est la longueur maximale d'une course.

Nous supposons que la densité de probabilité $f_Y(y)$ de la variable aléatoire $Y$, qui correspond au niveau de charge d'une voiture disponible, est uniforme :

$$f_Y(y) = \begin{cases} \frac{1}{C-s} & \text{si } y \in [s, C] \\ 0 & \text{Sinon} \end{cases} \tag{1.14}$$

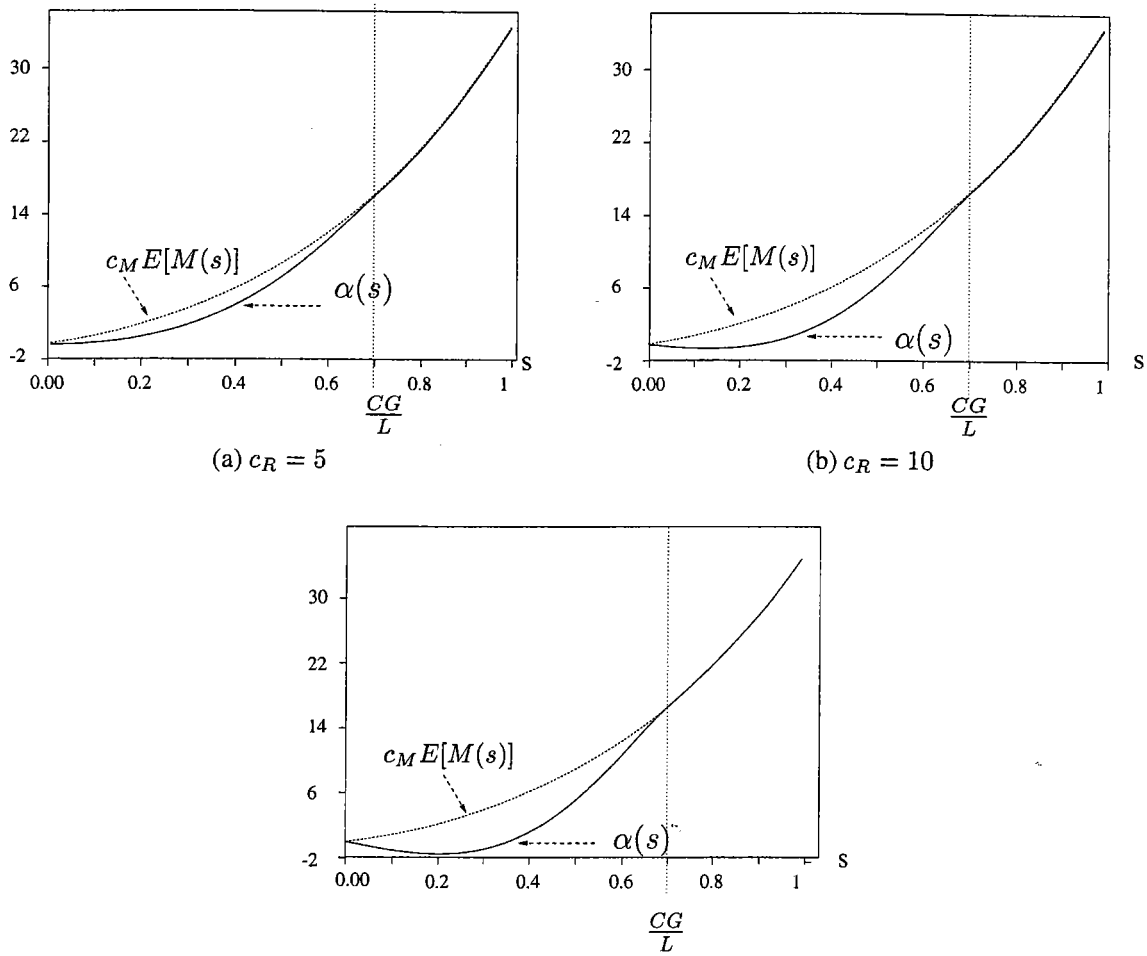
$q_R(s)$ est alors égal à :

$$\begin{aligned} q_R(s) &= \int_s^{\frac{CG}{L}} \left(P\left(X > \tfrac{yL}{C}/Y = y\right)\right) f_Y(y)\, dy \\ q_R(s) &= \int_s^{\frac{CG}{L}} \left(1 - \tfrac{yL}{GC}\right) \left(\tfrac{1}{C-s}\right) dy \\ q_R(s) &= \begin{cases} \frac{(GC-sL)^2}{2LGC(C-s)} & \text{si } y \in [s, \frac{CG}{L}] \\ 0 & \text{Sinon} \end{cases} \end{aligned} \tag{1.15}$$

### 1.3.2 Application numérique

Voici les valeurs choisies pour nos calculs :

- La distance maximale $G$ d'un trajet est 70 km.
- La distance maximale $L$ qu'une voiture, complètement chargée, peut parcourir est 100 km.




(a) $c_R = 5$ (b) $c_R = 10$

(c) $c_R = 15$

FIG. 1.1: *L'évolution du terme $\alpha(s)$ relativement au seuil $s$.*

- Le niveau de charge maximal $C$ qu'une voiture électrique peut atteindre est 1.
- Le coût unitaire $c_M$ dû à l'insatisfaction d'un client du fait de l'indisponibilité des voitures est 2 unités monétaires.
- Le coût unitaire $c_R$ appliqué lorsqu'une voiture tombe en panne d'énergie pendant une course est 5 unités monétaires.
- Le nombre de voitures $N$ qui se trouvent dans le site est 10.
- Le nombre maximal de demandes $k$ formulées dans un site est 10.
- Les probabilités d'avoir $k$ demandes dans un site sont listées dans un ordre croissant commençant par zéro demande et se terminant par 10 demandes: [0.1, 0.2, 0.2, 0.1, 0.1, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05].

Nous allons considérer chaque terme de l'équation 1.11 séparément et voir à quoi ressemble la courbe correspondante. Pour les figures, nous avons utilisé le logiciel Scilab.
Considérons le premier terme $\alpha(s)$

$$\alpha(s) = E[M(s)](c_M - c_R\, q_R(s)) = c_M\, E[M(s)] - c_R\, q_R(s)\, E[M(s)]$$

où :

- $c_M\, E[M(s)]$ est le coût moyen considéré lorsqu'un client ne trouve pas de voiture disponible dans un site.
- $c_R\, q_R(s)\, E[M(s)]$ représente le coût moyen qui serait à payer si des voitures indisponibles étaient autorisées à quitter le site et tombaient en panne en cours d'utilisation.
- $\alpha(s)$ qui est la différence entre les deux termes précédents, représente le gain obtenu en empêchant les voitures indisponibles (qui tomberaient en panne) de quitter le site.

Les figures 1.1(a, b, et c) montrent comment le terme $\alpha(s)$ évolue en fonction du seuil pour différentes valeurs de $c_R$. Si le seuil est fixé à une valeur supérieure à $\frac{CG}{L}$, les deux courbes $c_M\, E[M(s)]$ et $\alpha(s)$ sont superposées, puisque la probabilité qu'une voiture tombe en panne d'énergie est nulle dans ce cas. Nous remarquons que le gain augmente quand nous augmentons le coût unitaire d'une panne lors d'un trajet.

Maintenant, nous allons considérer le second terme $\beta(s)$ qui est égal à $c_R\, q_R(s)\, E[K]$. Ce

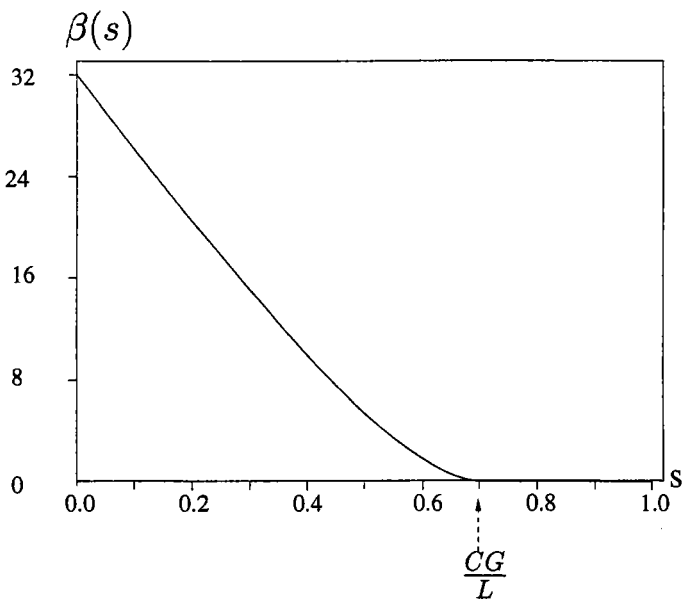



FIG. 1.2: *L'évolution du terme $\beta(s)$ relativement au seuil $s$.*

terme représente le coût à payer lorsque toutes les demandes des clients sont satisfaites. Nous supposons que toutes les voitures qui se trouvent dans le site sont disponibles, autrement dit, que leur niveau de charge est supérieur au seuil établi. Quelques clients vont tomber en panne. La figure 1.3 représente l'évolution de $\beta(s)$ en fonction du seuil $s$. La courbe montre que lorsque le seuil est bas, le coût $\beta(s)$ est élevé, et il diminue lorsque le seuil $s$ augmente.

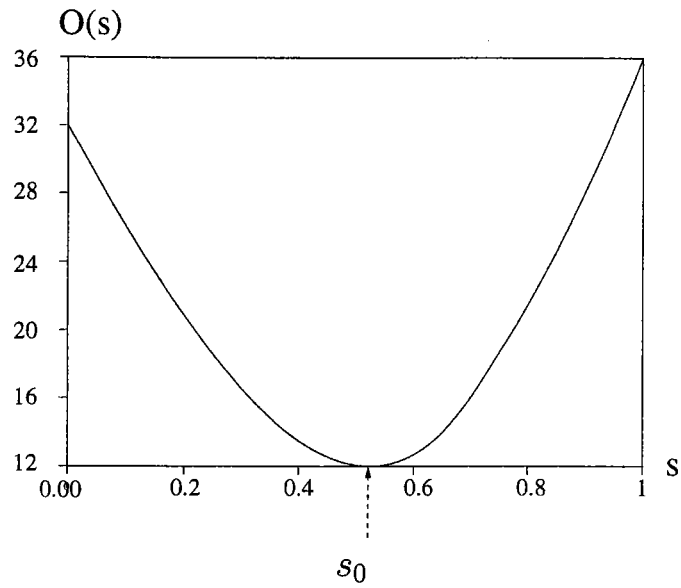



FIG. 1.3: *L'évolution de la fonction O(s) relativement au seuil s.*

Ce coût $\beta(s)$ devient nul quand le seuil est supérieur à $\frac{CG}{L}$ car la probabilité de tomber en panne d'énergie est nulle dans ce cas. Si nous combinons les deux termes $\alpha(s)$ et $\beta(s)$, nous obtenons la figure 1.3. Les résultats numériques ainsi que le bon sens préconisent de considérer que cette courbe est convexe. Nous supposons donc (provisoirement) la convexité. C'est pour cela que nous utilisons la méthode de dichotomie afin de trouver le seuil minimal $s_0$. La première étape de notre procédure est de déterminer l'intervalle d'investigation. Celui-ci sera initialement $[0, C]$. $C$ est le niveau de charge maximal d'une voiture. Nous calculons la dérivée de $O(s)$ pour le point qui se situe au centre de l'intervalle$[0, C]$, autrement dit pour le point $\frac{C}{2}$. Si la dérivée est nulle, nous avons donc trouvé le seuil minimal et la procédure se termine. Autrement, nous examinons la pente, si celle-ci est croissante, le point $\frac{C}{2}$ devient la limite supérieure de l'intervalle, sinon il devient la limite inférieure de l'intervalle.

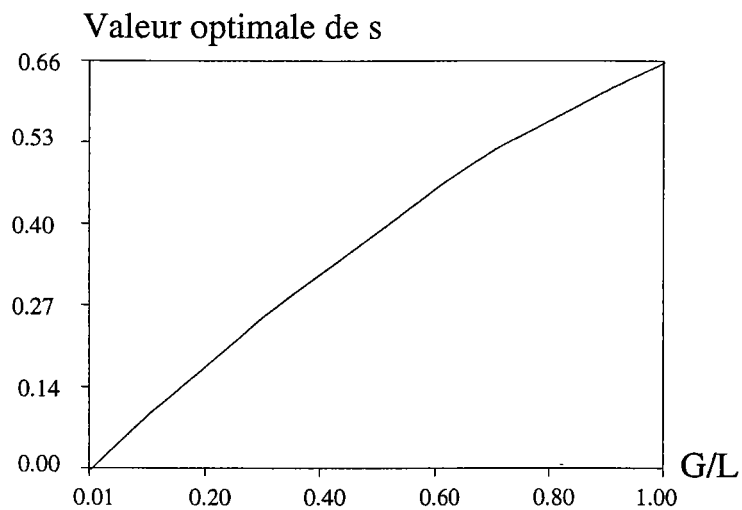



FIG. 1.4: *La relation entre le rapport $\frac{G}{L}$ et le seuil s.*

Le test suivant consiste à déterminer les différents seuils $s$ qui minimisent le critère O(s) pour différentes valeurs de $\frac{G}{L}$. Nous remarquons que lorsque le rapport augmente, autrement dit quand $G$ s'approche de L, le seuil augmente aussi (voir figure 1.4).

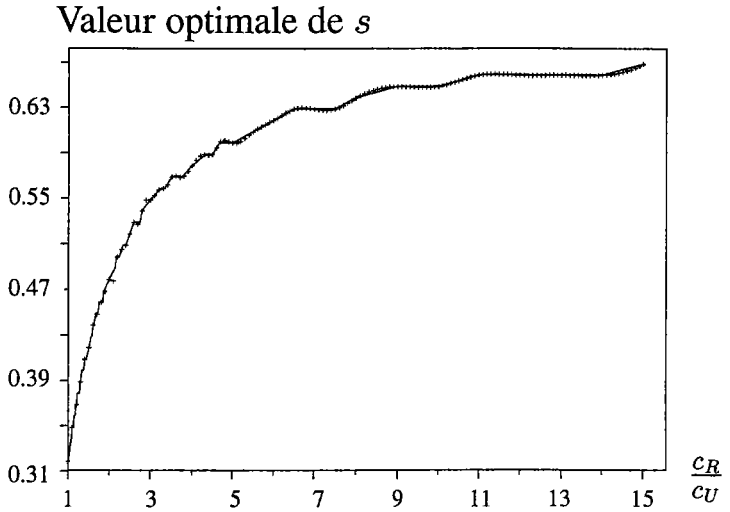



FIG. 1.5: *La relation entre le rapport $\frac{c_R}{c_M}$ et le seuil s.*

La figure 1.5 représente les différents seuils $s$ qui minimisent le critère O(s) pour différentes valeurs de $\frac{c_R}{c_M}$. Nous admettons que $c_R$ est supérieur à $c_M$. En effet, le coût induit par une panne en cours de trajet, du fait de ses nombreuses conséquences, est largement supérieur au coût induit par le mécontentement d'un client qui ne trouve pas de voiture disponible dans un site. Nous remarquons que lorsque le rapport $\frac{c_R}{c_M}$ augmente, le seuil augmente afin de réduire les ruptures d'énergie en cours de courses.

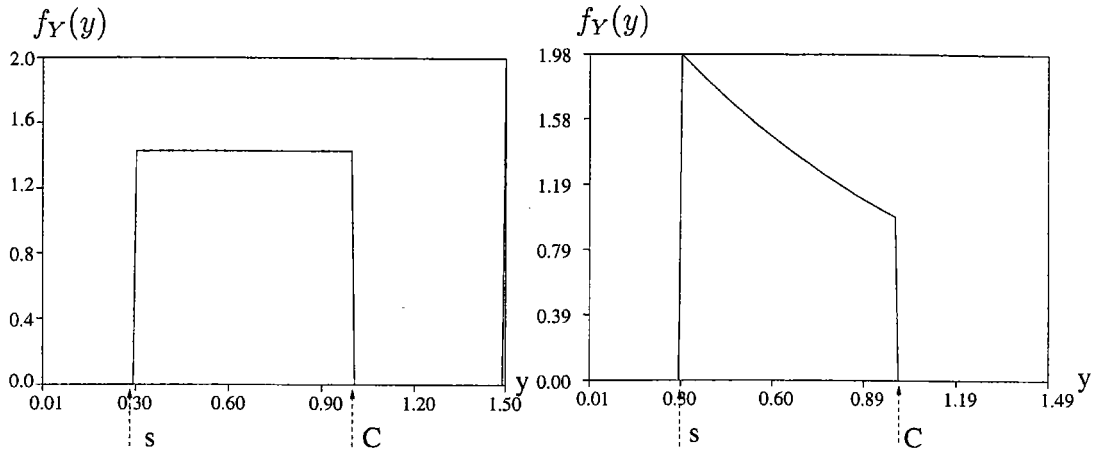
### 1.3.3 Cas plus réels

Nous faisons varier la densité de probabilité $f_Y(y)$, et nous étudions son effet sur le critère O(s) ainsi que sur le seuil.
La nouvelle densité de probabilité de la variable aléatoire $Y$ qui représente le niveau de charge d'une voiture disponible est :

$$f_Y(y) = \begin{cases} \frac{e^{-y}}{(e^{-s}-e^{-C})} & \text{si } y \in [s,\ C] \\ 0 & \text{Sinon} \end{cases} \tag{1.16}$$

La figure (1.6) compare les différentes densités de probabilité $f_Y(y)$ de la variable aléatoire $Y$ proposées en 1.14 et en 1.16. La nouvelle densité, dans le cadre des hypothèses de notre problème, est plus réaliste pour les raisons suivantes.

1. Pour atteindre un niveau de charge $C$, une voiture a besoin d'être immobilisée très longtemps. Pour cette raison, la proportion de voitures dont la charge se situe près du seuil est plus grande que celle dont la charge se trouve près de la capacité maximale $C$.




(a) $f_Y(y)$ = uniforme (b) $f_Y(y)$ = exponentielle

FIG. 1.6: *Une comparaison entre les deux densités $f_Y(y)$ considérées.*

2. Supposons que nous ayons un nombre limité de chargeurs de batteries. Nous pouvons penser qu'aussitôt le niveau de seuil légèrement dépassé, la procédure de charge est arrêtée afin de favoriser la recharge d'une autre voiture indisponible, et donc les charges des voitures seront proches du seuil.

Par rapport a nos calculs précédents, seul $q_R(s)$, la probabilité qu'une voiture tombe en rupture d'énergie lors d'un parcours, va changer.

$$q_R(s) = \int_s^{\frac{CG}{L}} \left(P\left(X > \frac{yL}{C}/Y = y\right)\right) f_Y(y)\, dy$$

$$q_R(s) = \int_s^{\frac{CG}{L}} \left(1 - \frac{yL}{GC}\right) \left(\frac{e^{-y}}{(e^{-s}-e^{-C})}\right) dy$$

$$q_R(s) = \begin{cases} \frac{GCe^{-s}+L(e^{\frac{-CG}{L}}-se^{-s}-e^{-s})}{GC(e^{-s}-e^{-C})} & \text{si } y \in [s, \frac{CG}{L}] \\ 0 & \text{Sinon} \end{cases} \tag{1.17}$$

Dans la figure (1.7), Nous superposons l'évolution de la fonction $O(s)$ lorsque $f_Y(y) = \frac{1}{C-s}$ et l'évolution de la fonction $O(s)$ lorsque $f_Y(y) = \frac{e^{-y}}{(e^{-s}-e^{-C})}$, afin de comparer l'effet de ces différentes densités de probabilité $f_Y(y)$ sur notre critère O(s).

La probabilité qu'une voiture tombe en panne lors d'une course est plus grande lorsque nous utilisons la probabilité de densité exponentielle puisque :

$$\int_s^{\frac{CG}{L}} \frac{e^{-y}}{e^{-s}-e^{-C}} dy > \int_s^{\frac{CG}{L}} \frac{1}{C-s} dy$$

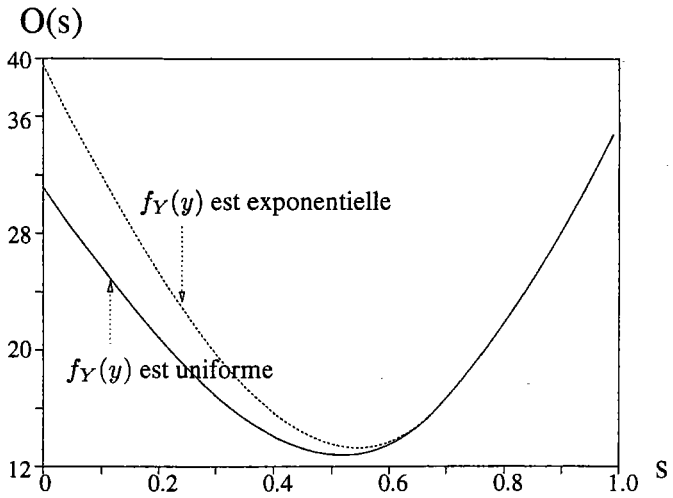



FIG. 1.7: *Une comparaison entre deux tracés de O(s) en variant leur* $f_Y(y)$.

De plus nous modifions la densité de probabilité de la variable aléatoire X qui représente la distance qu'un client peut parcourir.

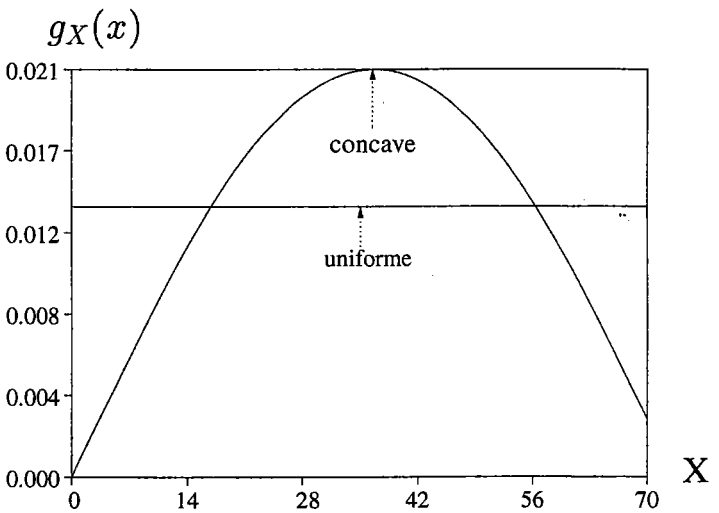


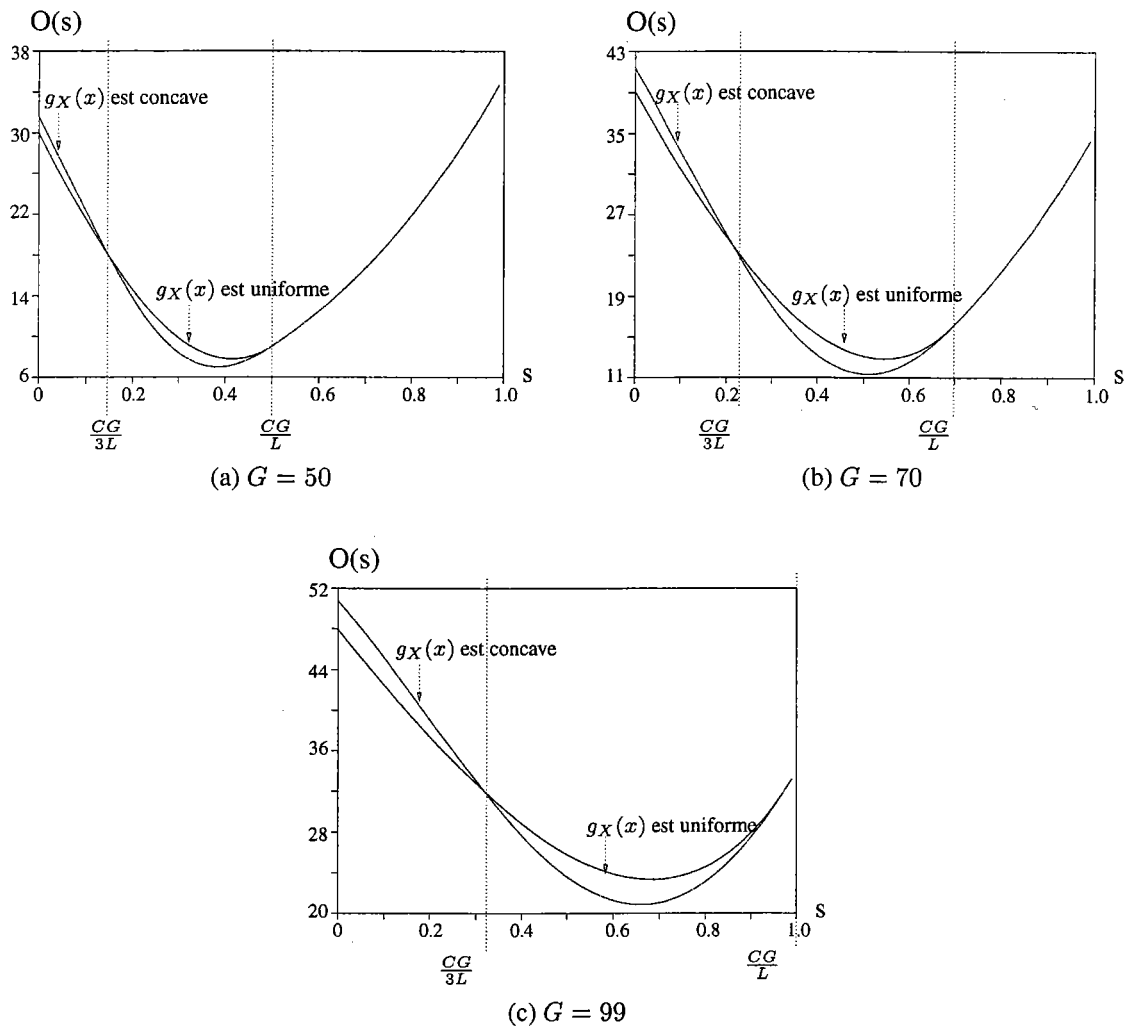

FIG. 1.8: *Les tracés de deux densités* $g_X(x)$ *considérées*

Nous avons opté pour la forme concave représentée dans la figure 1.8 pour les raisons suivantes :

- Un client ne va jamais emprunter une voiture afin d'effectuer une course de zéro km.
- Il est également rare qu'il emprunte à l'intérieur d'une ville, une voiture pour faire une course très courte. Par conséquent, la probabilité qu'un client emprunte une voiture commence à croître avec la longueur de la course envisagée.
- Ceci est vrai jusqu'à une certaine distance, ensuite cette probabilité diminue. En effet, la facturation incite à ne pas utiliser la voiture sur de trop longues distances.
- Finalement, d'après notre hypothèse initiale, la probabilité qu'un client effectue un trajet de distance $G$ n'est pas nulle.

Cette nouvelle fonction de densité $g_X(x)$ s'écrit :

$$g_X(x) = \begin{cases} \frac{3sin(\frac{3x}{G})}{G(1-cos(3))} & \text{si } x \in [0, G] \\ 0 & \text{Sinon} \end{cases} \tag{1.18}$$




(a) $G = 50$ (b) $G = 70$

(c) $G = 99$

FIG. 1.9: *Une comparaison entre deux tracés de O(s) en faisant varier leur $g_X(x)$, de même qu'en faisant varier le paramètre G.*

La probabilité qu'une voiture tombe en panne sachant que sa charge est $y$ s'écrit :

$$P(X > \frac{yL}{C}/Y = y) \quad = \quad \int_{\frac{yL}{C}}^{G} \frac{3sin(\frac{3x}{G})}{G(1-cos(3))} dx$$

$$= \begin{cases} \frac{cos(\frac{3yL}{GC})-cos(3)}{1-cos(3)} & \text{si } y \in [s, \frac{CG}{L}] \\ 0 & \text{Sinon} \end{cases} \quad (1.19)$$

Si nous gardons la fonction de densité de charge exponentielle déterminée en 1.16, la probabilité qu'une voiture tombe en panne notée $q_R(s)$, sera :

$$q_R(s) = \int_s^{\frac{CG}{L}} (\int_{\frac{yL}{C}}^{G} g_X(x)dx) f_Y(y)dy$$

$$= \begin{cases} \frac{e^{-s}[cos(3)[C^2G^2+9L^2]+CG[3Lsin(\frac{3sL}{CG})-CGcos(\frac{3sL}{CG})]]}{(C^2G^2+9L^2)(cos(3)-1)(e^{-s}-e^{-C})} & \\ \frac{-3Le^{(\frac{-CG}{L})}[3Lcos(3)+CGsin(3)]}{(C^2G^2+9L^2)(cos(3)-1)(e^{-s}-e^{-C})} & \text{si } y \in [s, \frac{GC}{L}] \\ 0 & \text{Sinon} \end{cases} \quad (1.20)$$

Pour différentes valeurs de $G$, distance maximale d'une course, (voir figure 1.9) nous comparons deux tracés du critère O(s). Le premier tracé, est calculé avec une densité de probabilité $g_X(x)$ uniformément distribuée et une densité de probabilité $f_Y(y)$ exponentielle. Le second est calculé avec une densité $g_X(x)$ sinusoïdale tandis que $f_Y(y)$ reste exponentielle.

## 1.4 Conclusion

Nous avons vu dans ce chapitre que déterminer un seuil de disponibilité n'est pas une tâche facile. Pour calculer ce seuil, nous avons défini un critère basé sur différents coûts, en faisant l'hypothèse d'un rééquilibrage optimal. Ce rééquilibrage optimal a pour conséquence que le nombre de voitures présentes dans un site est toujours supérieur ou égal à la demande, mais la charge de certaines de ces voitures n'est pas suffisante pour qu'elles soient remises aux clients. Notre but était de trouver le seuil qui minimise le coût global. Nous avons observé que nous devions prendre en considération deux coûts différents. Le premier est la conséquence de l'insuffisance de voitures suffisamment chargées pour satisfaire la demande. Le second est le coût résultant des voitures défaillantes en cours d'utilisation. Nous remarquons que ces deux coûts concernent des phénomènes antinomiques. Autrement dit, si nous réduisons un coût en optant pour un certain seuil, nous augmentons l'autre coût. Par conséquent, nous devons établir un compromis et trouver le seuil qui minimise le critère en son global. C'était l'objectif de ce chapitre.

# Chapitre 2

# *Gestion des plots*

## Résumé

*Ce chapitre concerne la gestion de la recharge des voitures dans un site. En effet, seules quelques places de parking sont munies d'un système de recharge. De ce fait, il arrive que toutes les voitures présentes dans le site ne puissent pas être rechargées simultanément. Pour cette raison, nous avons développé plusieurs politiques adaptées aux différentes situations.*

*Mots-clés : Voiture Electriques, Dispositifs de Recharge, Simulation.*

## Sommaire

## 2.1 Introduction

Le nouveau mode de transport que nous étudions est, comme nous l'avons précédemment décrit, constitué d'un parc de voitures électriques et d'un nombre défini de sites. Une voiture électrique a besoin de recharger régulièrement ses batteries. Pour cette raison, les sites sont munis de systèmes de recharge. Le système de recharge utilisé est basé sur la technologie inductive. Lorsqu'un client arrive dans un site et se gare, le système de recharge fonctionne automatiquement sans aucune intervention de la part de l'utilisateur [BN96]. Autrement dit, le fonctionnement du système de recharge est transparent pour l'utilisateur. La sécurité ainsi que la robustesse du chargeur sont d'autres qualités souhaitées. La robustesse du chargeur par induction est une caractéristique importante puisque celui-ci est intégré dans le milieu urbain, mis en libre service, et par conséquent peut être l'objet de vandalisme. Le prix d'un tel dispositif est plus élevé que les dispositifs de recharge standards, ce qui explique que toutes les places de parking n'en sont pas équipées.

Notre but est la gestion efficace de ce mode de transport public innovant. Le client est satisfait du service offert lorsqu'il trouve une voiture à sa disposition quand il se présente dans un site, et ceci quelle que soit l'heure. Une voiture est disponible lorsque sa charge est supérieure au seuil de disponibilité déterminé dans le chapitre précédent. Comme les voitures ne peuvent pas toutes être rechargées simultanément, et que notre but est d'obtenir le taux de satisfaction des clients le plus élevé possible, une politique de recharge efficace est nécessaire.

Cette politique de recharge prend en compte le nombre de voitures présentes dans le site, le niveau de charge de chacune de ces voitures, le seuil de disponibilité propre au site, ainsi que les demandes prévisionnelles dans ce même site. La politique détermine les voitures qui sont en charge à chaque instant.

Dans ce chapitre, nous développons plusieurs politiques que nous testons par simulation. Nous terminons le chapitre par une analyse des résultats obtenus.

## 2.2 Les politiques de recharge

Avant de détailler les différentes politiques développées, nous rappelons plusieurs définitions qui seront souvent utilisées dans ce chapitre.

- ⋆ *Une voiture indisponible* est une voiture qui n'est pas utilisable par un client parce que son niveau de charge est inférieur au seuil de disponibilité. Elle peut être en cours ou en attente de chargement.

- ⋆ *Un plot* est une place de parking munie d'un système de recharge (ou chargeur).

- ⋆ *Une place de parking simple* est une place de parking qui n'est pas équipée du système de recharge. Par conséquent, une voiture garée à cette place ne peut pas recharger ses batteries.

### 2.2.1 Politique de recharge 1

La première politique de recharge exposée est la plus simple. La figure 2.1 présente l'algorithme de fonctionnement.

Noter que :

– $N_v^x$ est le niveau de charge de la voiture x.

– $s$ est le seuil de disponibilité.

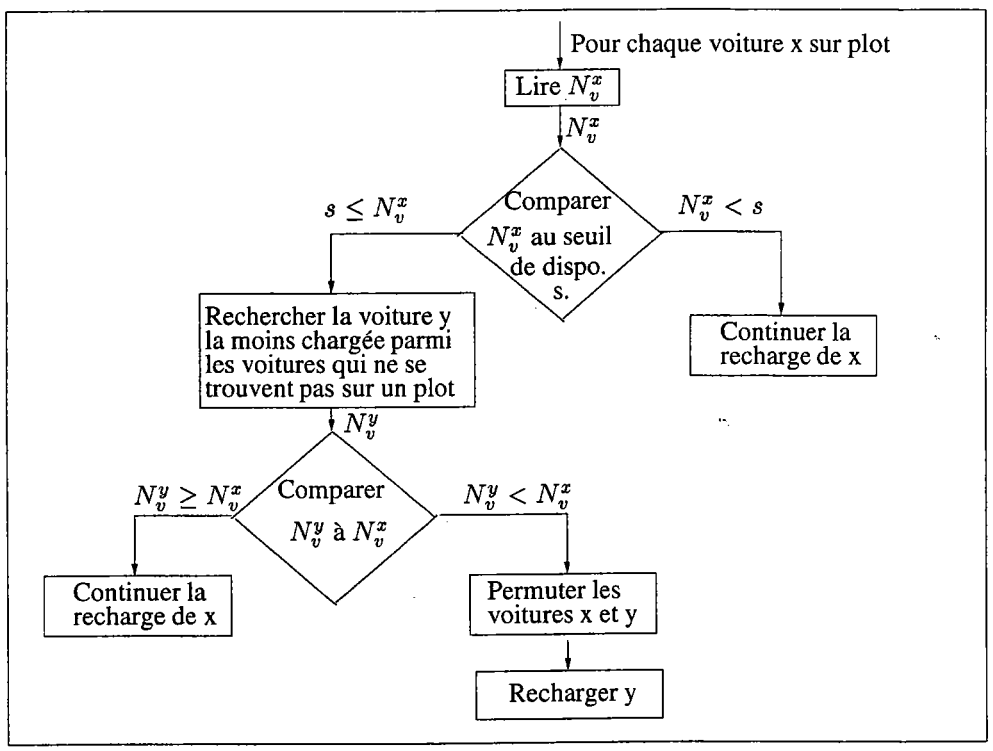



FIG. 2.1: *Procédure d'action no.1 pour une voiture qui se trouve sur un plot.*

Les voitures qui sont sur un plot et qui ont un niveau de charge inférieur au seuil de disponibilité restent sur le plot et sont en cours de recharge. Si tous les plots sont occupés, et que des voitures indisponibles se trouvent sur des places de parking simples, alors un échange de places peut avoir lieu. La voiture indisponible (dont la charge est inférieure au seuil) la moins chargée qui ne se trouve pas sur un plot prend la place d'une voiture disponible en cours de recharge. De cette façon, cette voiture peut à son tour se recharger. De même, si un plot se libère (un client a pris la voiture qui s'y trouvait) ou est déjà libre, la place est prise par la voiture indisponible de charge minimale. En fait, cette politique fait en sorte que les niveaux de charge des différentes voitures présentes dans le site tournent autour du seuil de disponibilité.

### 2.2.2 Politique de recharge 2

La deuxième politique développée (voir figure 2.2) a pour but d'augmenter le nombre de voitures disponibles dans le site en favorisant la recharge des voitures indisponibles qui ont un niveau de charge proche du seuil de disponibilité. Autrement dit, les voitures qui seront le plus rapidement disponibles ont la priorité pour utiliser les plots.

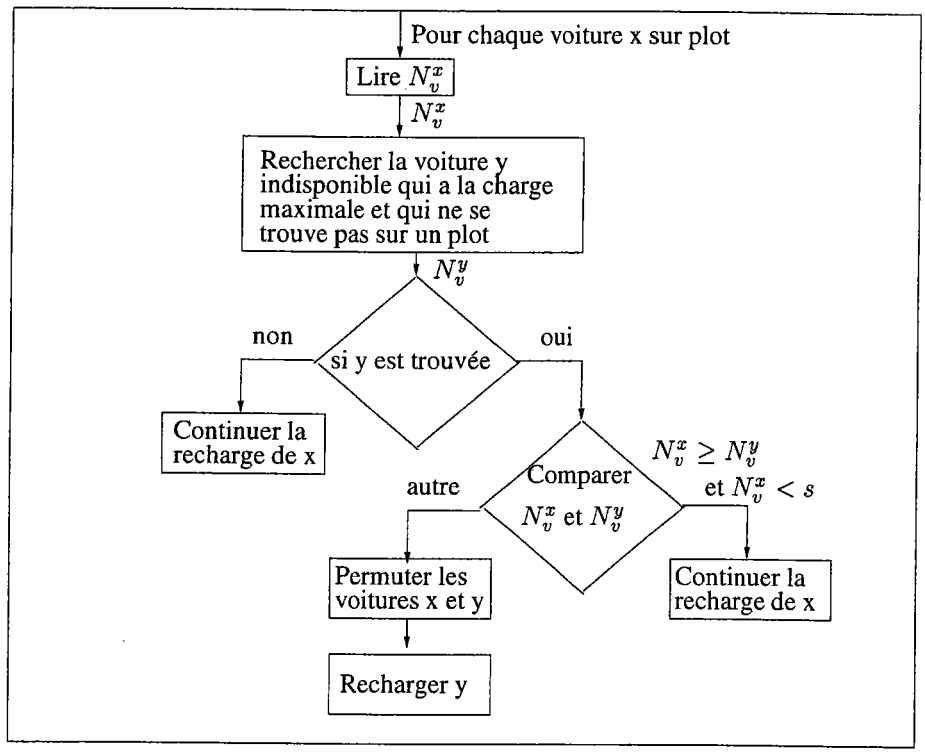



FIG. 2.2: *Procédure d'action no.2 pour une voiture qui se trouve sur un plot.*

L'intérêt de cette politique est de maximiser le nombre de voitures disponibles dans un site. Cette politique est adaptée à la situation présentant un nombre important de demandes et beaucoup de voitures indisponibles. Comme les voitures indisponibles qui ont un niveau de charge proche du seuil de disponibilité ont la priorité pour l'utilisation des plots, les clients seront ainsi plus vite servis.

La figure 2.3 est un exemple de situation que l'on peut trouver dans un site. Dans cet exemple, nous avons dix voitures dont les charges varient de 0 à 1. Le seuil de disponibilité pour ce site est fixé à 0.44.

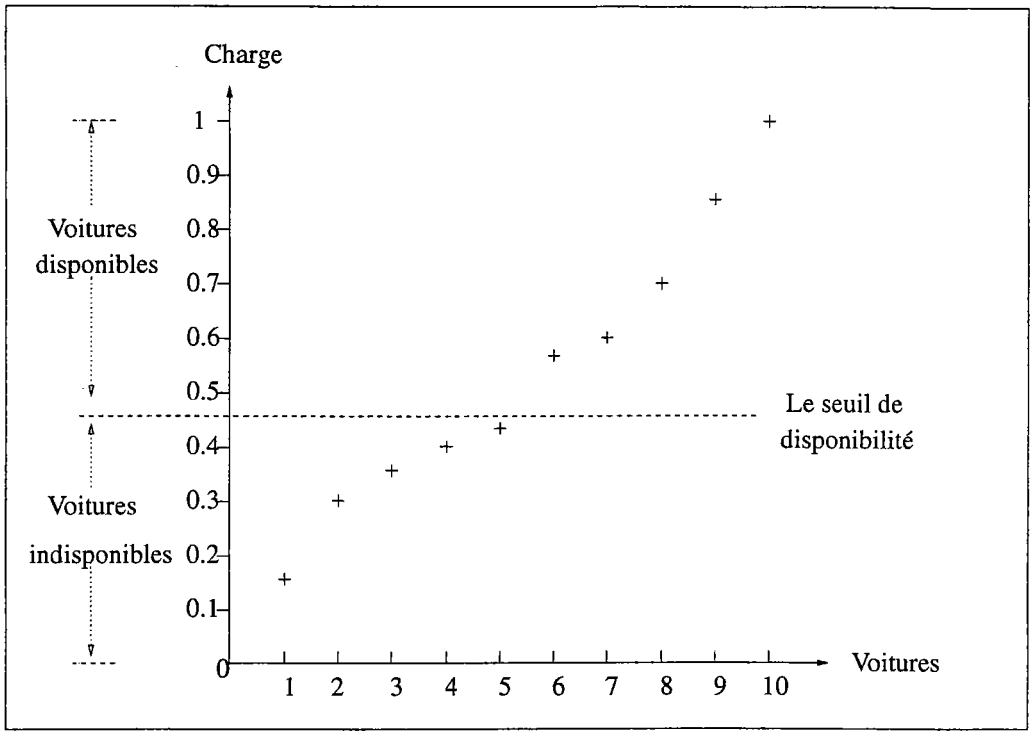



FIG. 2.3: *Les différents niveaux de charge des voitures présentes dans le site.*

Si nous supposons que les voitures ne sont pas sur des plots, l'ordre d'attribution des plots aux voitures selon la deuxième politique est donné par la figure 2.4.

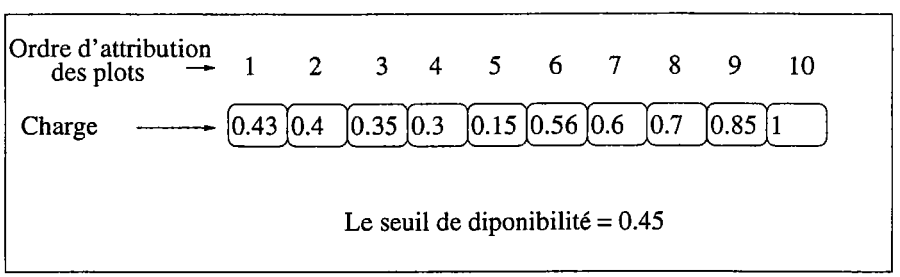



FIG. 2.4: *L'ordre d'attribution des plots aux différentes voitures présentes dans le site.*

### 2.2.3 Politique de recharge 3

Afin de favoriser la recharge totale des batteries et, ainsi, optimiser la capacité et la vie des batteries [Can96], une troisième politique a été développée. La politique de recharge 3 stipule que si une voiture se trouve sur un plot, elle y reste jusqu'à ce qu'elle soit complètement rechargée. Le seuil de disponibilité dans ce cas ne joue aucun rôle dans la décision de stopper la recharge ou de recharger une voiture. La voiture la plus chargée est attribuée au client qui se présente dans le site. De cette façon, l'emplacement muni d'un système de recharge est libre et peut être attribué à la voiture la plus chargée présente dans le site. La figure 2.5 résume la logique de cette politique.

Noter que $C$ est la charge maximale qu'une voiture peut atteindre.

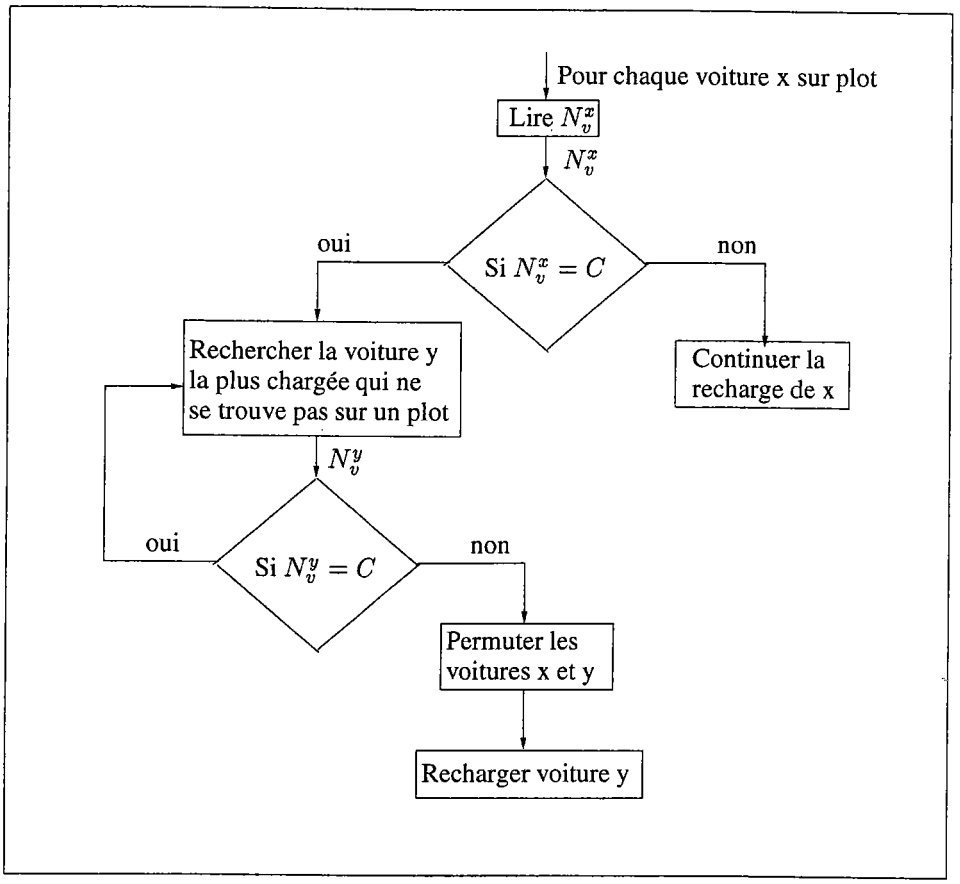



FIG. 2.5: *Procédure d'action no.3 pour une voiture qui se trouve sur un plot.*

## 2.3 Simulation

Afin de tester les différentes politiques de recharge présentées dans ce chapitre, nous avons eu recours à la simulation. Nous séparons le problème de la recharge du problème de la redistribution des voitures sur les sites. Pour cela, nous supposons que notre système est parfaitement rééquilibré, et ainsi, chaque voiture qui quitte un site est automatiquement remplacée par une autre voiture dont la charge est simulée suivant différentes lois de répartitions. Lors des simulations, nous étudions l'effet de ces différentes politiques sur le taux de satisfaction des clients. Dans notre cas, le système se limite à un seul site. Une voiture qui quitte ce site, effectue un déplacement d'une durée plus ou moins longue pour enfin revenir dans son site de départ. La décharge de la voiture est proportionnelle à son temps d'utilisation. Nous utilisons le simulateur décrit dans les paragraphes suivants. Mais avant de procéder dans la description du programme de simulation, nous commençons par déterminer le seuil de disponibilité. Ce seuil autorise une voiture à quitter le site, ou le lui interdit. Si le niveau de charge d'une voiture est supérieur ou égal au seuil de disponibilité, cette voiture devient

disponible et par conséquent, est autorisée à quitter le site si un client en a besoin. Ce seuil est calculé de façon à obtenir un taux de satisfaction des clients le plus élevé possible. En fait, c'est un compromis entre le désir de satisfaire les clients qui arrivent dans le site, c'est à dire ne pas les faire attendre, et entre le risque qu'ils tombent en panne lors d'une course. Le calcul du seuil est expliqué dans le chapitre précédent. Ce seuil est un paramètre de notre simulation.

### 2.3.1 Description du programme de simulation

Les paragraphes suivants présentent les différents paramètres nécessaires à la simulation, ainsi que les instances choisies.

Les principaux paramètres sont :

- Le nombre de voitures présentes dans le site ainsi que leur état (en cours d'utilisation, disponible, en panne ou en cours de chargement).
  - Dix voitures sont présentes dans le site au début de la simulation. Celles-ci sont complètement chargées et par conséquent disponibles.
- Le nombre total de places de parking dans le site. Ce nombre comprend les places de parking munies ou non d'un système de recharge.
- Le nombre de places de parking équipées d'un système de recharge.
  - Dans le site, il y a dix emplacements fixes. Le nombre de places de parking munies d'un système de recharge est compris entre 1 et 10 emplacements, mais celui-ci dépendra des différentes simulations effectuées. Lors des simulations, nous étudions l'impact du nombre de plots sur le taux de satisfaction des clients.
- La durée de chaque période élémentaire $\Delta$ sachant qu'une journée est segmentée en un nombre fini de périodes élémentaires.
  - La période élémentaire $\Delta$ est égale à huit minutes.
- Les probabilités de demandes pendant chacune des périodes élémentaires. Une demande est formulée lorsqu'un client se présente dans le site.
  - Pendant une période élémentaire, deux clients (tout au plus) peuvent se présenter dans le site. Les probabilités de demandes varient selon les différentes simulations afin d'évaluer leurs impacts sur les politiques testées.
- Les durées minimale et maximale des déplacements. Une voiture en cours d'utilisation consomme de l'énergie proportionnellement à la durée d'utilisation.
  - La durée minimale d'une course est équivalente à une période élémentaire. La durée maximale d'une course, peut être égale à deux périodes élémentaires.

Les paramètres concernant une voiture électrique ainsi que ceux nécessaires au calcul du seuil de disponibilité sont :

- Le niveau de charge maximal $C$ qu'une voiture peut atteindre.
  - $C = 1$.
- La distance maximale $L$ qu'une voiture complètement chargée peut parcourir.
  - $L = 40\ km$.
- La distance maximale $G$ qu'une voiture parcourt lors d'une course.
  - $G = 14\ km$.
- Spécifications sur la recharge et décharge d'une voiture électrique :
  - En ville, une voiture a en moyenne une vitesse de 50 $km/h$ ; autrement dit, elle parcourt une distance de 6.6 $km$ par période élémentaire $\Delta$.
  - Nous supposons que la recharge des batteries d'une voiture électrique est linéaire. Une voiture électrique a besoin de huit heures pour se recharger totalement [Can96]. Par conséquent, le niveau de charge des batteries d'une voiture se trouvant sur un plot augmente de 0.0167 par période élémentaire $\Delta$.
  - Une voiture électrique consomme de l'énergie lors de ses déplacements. Nous supposons que cette consommation d'énergie est une fonction linéaire du temps d'utilisation et de la distance parcourue. Autrement dit, une voiture électrique en cours de déplacement consomme 0.165 de la valeur de $C$ par période élémentaire $\Delta$.

Les paramètres propres à la simulation sont :

- Le nombre $Z$ de simulations.
  - $Z = 1000$.
- La durée de la simulation, autrement dit l'horizon $h$ d'étude sur lequel on teste les différentes politiques. Les politiques peuvent être testées sur plusieurs heures ou même sur plusieurs jours.
  - L'horizon $h$ est fixé à 75 périodes, autrement dit, à dix heures.
- Le client qui ne trouve pas de voiture disponible lorsqu'il se présente dans un site tolère une attente de quelques minutes.
  - Dans notre cas, le client tolère une attente tout au plus équivalente à 2 périodes élémentaires $\Delta$.

A chaque période élémentaire $\Delta$, les évènements suivants peuvent se produire :

- L'arrivée d'une demande dans le site.

- Le départ d'une voiture du site.
- L'arrivée d'une voiture dans le site.
- La fin de chargement des batteries d'une voiture.
- L'atteinte du seuil de disponibilité par le niveau de charge d'une voiture.
- L'arrivée du personnel chargé des déplacements locaux. Les déplacements locaux sont les déplacements intra-site, comme par exemple, déplacer une voiture qui se trouve sur un emplacement simple vers un emplacement muni d'un plot.

Ces différents évènements produisent plusieurs mises à jour, à savoir :

- L'emplacement des différentes voitures (une voiture peut se trouver sur un emplacement muni d'un plot ou pas).
- La file d'attente des demandes dans le site. Si un client n'a pas de voiture disponible au bout d'un certain temps d'attente, il s'en va mécontent.
- Le nombre de systèmes de recharge en service.

### 2.3.2 Résultats des simulations

Afin de tester les différentes politiques, nous observons l'effet de ces politiques de recharge sur le pourcentage de clients satisfaits dans ce site. Le pourcentage de clients satisfaits est le rapport entre les clients servis, autrement dit, ceux qui ont trouvé une voiture disponible même si c'est après une attente limitée, et la totalité des demandes formulées dans ce site.

Le tableau 2.1 présente une comparaison entre les différentes politiques de recharge lorsque nous faisons varier le nombre de plots dans le site. Le but est de connaître l'impact du nombre de plots sur le pourcentage de clients satisfaits. Lors de ces simulations, le personnel chargé des déplacements locaux est présent dans le site en permanence.

TAB. 2.1: *Pourcentage de satisfaction des clients lorsque nous faisons varier le nombre de plots dans le site*

| Nombre de plots | Politique 1 | Politique 2 | Politique 3 |
|---|---|---|---|
| 1 | 62.52% | 65.88% | 58.48% |
| 2 | 71.80% | 73.79% | 66.65% |
| 5 | 98.02% | 98.25% | 95.29% |
| 10 | 100% | 100% | 100% |

Le tableau 2.2 présente les pourcentages de clients satisfaits obtenus lorsque le personnel chargé du rééquilibrage intra-site n'est pas en permanence présent dans le site. Dans ces

simulations, le nombre de places de parking munies d'un chargeur est fixé à deux. Chaque $x$ périodes (voir la première colonne du tableau 2.2), le personnel effectue les déplacements locaux déterminés par les différentes politiques.

TAB. 2.2: *Pourcentage de satisfaction des clients lorsque le personnel n'est pas en permanence présent dans le site*

| Fréquence des visites | Politique 1 | Politique 2 | Politique 3 |
|---|---|---|---|
| 1 | 71.80% | 73.79% | 66.65% |
| 5 | 70.91% | 73.25% | 65.02% |
| 10 | 67.18% | 67.60% | 65.47% |
| 20 | 66.20% | 66.53% | 64.84% |

Le tableau 2.3 présente les résultats obtenus lorsque les demandes varient dans le site. La première colonne du tableau représente l'espérance du nombre de demandes dans le site pendant une durée équivalente à six périodes élémentaires.

TAB. 2.3: *Pourcentage de satisfaction des clients*

| Espérance de demandes | Politique 1 | Politique 2 | Politique 3 |
|---|---|---|---|
| 3.45 | 97.20% | 98.36% | 92.74% |
| 4.75 | 71.80% | 73.79% | 66.65% |
| 5.00 | 66.84% | 69.54% | 62.76% |
| 5.80 | 56.95% | 59.15% | 54.27% |

*Remarques :*

1. En règle générale, la seconde politique obtient les meilleurs pourcentages de clients satisfaits, alors que la troisième obtient les plus mauvais résultats. Remarque : le but de la troisième politique n'étant pas la satisfaction des clients mais l'optimisation des batteries, le résultat était prévisible.

2. Lorsque le nombre de plots est maximal, les résultats des différentes politiques se rapprochent.

3. Lorsque le nombre de plots augmente, le nombre de voitures disponibles augmente et, par conséquent, le pourcentage de satisfaction des clients augmente aussi. Ceci est vrai pour les trois politiques.

4. Pour ce qui concerne la seconde politique, plus le nombre de plots dans le site est petit, meilleur est le pourcentage de satisfaction des clients (en comparaison avec les résultats des autres politiques). Ceci est dû au fait que comme toutes les voitures ne peuvent pas être rechargées simultanément, plus rapidement les voitures sont disponibles, plus vite les clients sont satisfaits.

5 Lors des simulations que l'on a effectuées en modifiant la fréquence des visites du personnel chargé du rééquilibrage intra-site, plus on espaçait les visites, plus le pourcentage de satisfaction des clients diminuait. L'effet de l'espacement des visites du personnel sur le pourcentage de clients satisfaits est moins évident lorsqu'on applique la troisième politique. Etant donné que la troisième politique favorise le rechargement total des batteries, la voiture la plus chargée se situe (généralement) sur un plot. Comme c'est la voiture la plus chargée du site qui est attribuée au client, la voiture libère un plot et par conséquent, le rééquilibrage se fait naturellement.

6. Plus le nombre de demandes est petit, meilleur sont les résultats. La seconde politique obtient les meilleurs résultats.

## 2.4 Conclusion

Ce chapitre termine la partie de l'étude concernant la gestion de la recharge. A ce stade de l'étude, nous avons défini le critère qui détermine le seuil de disponibilité. Ce seuil est un compromis entre la satisfaction des clients et le risque que les voitures tombent en panne pendant les courses. Ensuite nous avons développé et testé plusieurs politiques pour la gestion des plots lorsque ceux-ci sont insuffisants pour recharger simultanément toutes les voitures présentes dans le site. Notre conclusion globale est que le calcul du seuil de disponibilité ainsi que la seconde politique de recharge ont comme conséquence un meilleur taux de satisfaction de clients.

# Cinquième partie

# Intégration des deux parties précédentes

# Chapitre 1

# *Intégration du rééquilibrage et de la recharge*

## Résumé

*Dans ce chapitre, nous présentons le simulateur développé. Celui-ci regroupe toutes les politiques détaillées dans le mémoire. L'utilisateur paramètre le système, et il choisit les différentes politiques qu'il veut tester.*

*Mots-clés : Simulation.*

## Sommaire

## 1.1 Introduction

L'objectif de ce chapitre est de présenter le simulateur développé afin de faciliter son utilisation et d'aider à l'introduction de nouvelles politiques, si nécessaire. Le langage $C$ est utilisé pour le développement des programmes du simulateur. Dans les précédents chapitres, nous avons séparé le problème de la redistribution du problème de la recharge, et nous avons testé chaque politique individuellement. Dans ce chapitre, nous intégrons l'ensemble. Le programme simule le fonctionnement du système de voitures électriques mises en libre service en complément des transports en commun suivant les paramètres donnés par l'utilisateur. L'utilisateur choisit la politique de redistribution souhaitée ainsi que la politique de recharge qui lui convient. Dans ce chapitre, nous commençons par détailler les différents paramètres que l'utilisateur fournit au simulateur. Nous enchaînons par la description des différents évènements qui influencent la situation des différentes entités telles que : les voitures, les sites et les clients. Nous poursuivons ce chapitre en listant les différentes structures des données utilisées et en résumant les principales procédures du programme de simulation. Nous terminons ce chapitre par la description de plusieurs graphiques issus des résultats des simulations.

## 1.2 Le Simulateur

L'utilisateur du simulateur a la possibilité de paramétrer le système comme il lui convient. Les paramètres du simulateur sont listés ci-dessous.

### 1.2.1 Paramètres

Les paramètres du simulateur sont :

*Remarque : le nom des variables sont entre les parenthèses.*

1. Le nombre de sites, (*NbSite*).
2. Le nombre de places de parking dans chaque site, (*Capacite*).
3. Le nombre de plots dans chaque site. En fait, une place de parking peut être équipée d'un chargeur ou pas. Le nombre de plots dans un site est inférieur ou égal au nombre de places de parking, (*NbBorne*).
4. Le nombre de voitures électriques qui constituent la flotte, (*NbVoit*).
5. La répartition initiale des voitures dans les différents sites, autrement dit, le nombre de voitures présentes dans chaque site au début de la simulation, (*EtatDep*).
6. Le nombre de périodes élémentaires constituant un cycle. Prenons l'exemple d'une journée, celle-ci peut être segmentée en un nombre fini de périodes élémentaires, chacune d'elle représentant une durée, (*TailleCycle*).
7. La demande maximale possible dans chaque site à destination des autres sites du système pour chacune des périodes élémentaires, (*NbMaxDem*).

8. Les probabilités de demandes entre tout couple de sites pour chaque période élémentaire. *Remarque :* les deux sites peuvent être identiques, (*Proba*).

9. Les durées minimales des déplacements, en nombre de périodes élémentaires, entre tout couple de sites en connaissant la période élémentaire du départ, (*DureeMin*).

10. L'écart, en nombre de périodes élémentaires, entre les durées minimales et les durées maximales des déplacements entre tout couple de sites, connaissant la période élémentaire du départ, (*DureeEcart*).

11. Le moment où la simulation commence, (*DateDeb*).

12. La durée de la simulation, autrement dit la période sur laquelle on teste les différentes politiques. Cette durée peut représenter plusieurs heures, plusieurs journées, ou même plusieurs mois, (*DureeSim*).

13. Le nombre de simulations, (*NbSim*).

14. Le temps d'attente, en nombre de périodes élémentaires, qu'un client tolère avant de quitter le système insatisfait. Lorsqu'un client se présente dans un site, il souhaite trouver une voiture disponible, mais il peut attendre un certain temps avant de s'en aller mécontent, (*NbPerMec*).

15. Le nombre de voitures de service (camions) chargées du rééquilibrage, (*NbCam*).

16. La capacité de chacun des camions, (*CapaCam*).

17. L'emplacement de chacun des camions, (*EmplCam*).

18. Le temps moyen, en nombre de périodes élémentaires, nécessaire aux camions pour parcourir la distance qui sépare les sites, (*DureeCam*).

19. Le seuil de disponibilité dans chacun des sites, c'est à dire le niveau de charge au delà duquel une voiture est disponible (*Seuils*).

20. Les paramètres concernant une voiture électrique sont :

    a. La quantité de charge reçue, par période élémentaire, lorsque la voiture se trouve sur un plot, (*Recharge*).

    b. La quantité de charge qu'une voiture électrique en cours de déplacement consomme par période élémentaire, (*PourFatigue*).

21. La politique de redistribution choisie, (*Distrib*), sachant que l'on a le choix entre :

    a. *Priorité au site le plus proche d'un camion.*

    b. *Service groupés.*

    c. *Priorité au site le plus éloigné d'un camion.*

    d. *Trajet cyclique du camion.*

e. *Pas de politique de redistribution.*

22. La politique de recharge choisie, (*Rech*), sachant que l'on a le choix entre :
    a. *Politique de recharge 1.* Lorsqu'une voiture atteint le seuil de disponibilité, elle cède sa place à la voiture la moins chargée présente dans le site.
    b. *Politique de recharge 2.* Lorsqu'une voiture atteint le seuil de disponibilité, elle cède sa place à la voiture indisponible qui a la charge la plus proche du seuil.
    c. *Politique de recharge 3.* Lorsqu'une voiture se trouve sur un plot, elle cède la place uniquement lorsque son niveau de charge atteint la charge maximale.
    d. *Pas de politique de recharge.* Dans ce cas, le niveau de charge d'une voiture, le processus de recharge, le nombre de plots dans un site, la charge consommée par une voiture pendant une période élémentaire et les différents seuils de disponibilité ne sont pas pris en considération.
23. La fréquence des visites du personnel chargé des déplacements locaux. Les déplacements locaux sont les déplacements intra-site, comme par exemple, déplacer une voiture qui se trouve sur un emplacement simple vers un emplacement muni d'un plot, (*JockeyPres*).
24. L'horizon minimal pour qu'un état soit considéré favorable, (*HorizonMin*).
25. L'horizon nécessaire pour le calcul des états défavorables. Cet horizon est fixé supérieur ou égal au temps nécessaire pour passer d'un état défavorable à un état favorable, (*HorizonDef*).
26. Le seuil de satisfaction souhaité dans chaque site, (*SeuildeSatisf*).

### 1.2.2 Hypothèses

Nous avons posé quelques hypothèses qui sont listées ci-dessous.

1. Les voitures initialement présentes dans le site sont complètement chargées et disponibles.
2. Les temps de chargements des voitures dans le camion ainsi que le temps de déchargement des voitures dans les sites sont négligeables.
3. Lorsqu'une voiture est en stationnement dans un site, elle ne consomme pas d'énergie. Autrement dit, phares oubliés et consommation du calculateur ne sont pas pris en considération.
4. Lorsqu'une voiture tombe en panne, elle est remorquée à son site de provenance.
5. Lorsque un camion est au repos, il se trouve dans le dernier site visité.
6. Les camions ont besoin d'un temps identique pour parcourir une distance indépendamment de leurs capacités.
7. Le niveau de charge maximal qu'une voiture peut atteindre est 1.

### 1.2.3 Evènements

A chaque période élémentaire, des évènements suivants sont considérés :

1. L'arrivée d'une demande dans un site.
2. Le départ d'un client mécontent à cause d'une trop longue attente.
3. Le départ d'une voiture d'un site.
4. L'arrivée d'une voiture dans un site.
5. Une voiture atteint la charge maximale.
6. Une voiture tombe en panne lors d'une course.
7. L'atteinte du seuil de disponibilité par le niveau de charge d'une voiture.
8. L'arrivée d'un camion dans un site.
9. L'arrivée du personnel chargé des déplacements locaux.

### 1.2.4 Les mises à jour

Ces différents évènements produisent plusieurs mises à jour, à savoir :

1. L'emplacement des différentes voitures ainsi que leur état (en cours d'utilisation, disponible, en panne, ou en cours de chargement).
2. Le nombre de voitures dans chaque site ainsi que leur emplacement (une voiture peut se trouver sur un emplacement muni d'un plot ou pas).
3. Le nombre de systèmes de recharge en service.
4. La file d'attente des demandes dans le site. Au bout d'un certain temps d'attente, si un client n'a pas de voiture disponible, il s'en va mécontent.

### 1.2.5 Structures des données

Dans les paragraphes suivants, nous listons les plus importantes structures des données que nous utilisons dans le simulateur. Afin de limiter la place mémoire utilisée, nous avons recours a des structures des données dynamiques, autrement dit, les variables sont crées et détruites pendant l'exécution du programme. Les listes utilisées sont des listes double chaînages.

**A.** La structure des données *DonneesStructure* contient les paramètres du système, à savoir :

TAB. 1.1: *La structure des données de DonneesStructure*

| Nom de la variable | Description | Commentaires |
|---|---|---|
| *SeuildeSatisf* | Nombre réel | Seuil de satisfaction souhaité |
| *NbSite* | Nombre entier | Nombre de sites |
| *Capacite[i]* | Tableau de dimension 1 de nombres entiers | Capacité de chaque site |
| *NbBorne[i]* | Tableau de dimension 1 de nombres entiers | Nombre de plots dans chaque site |
| *NbVoit* | Nombre entier | Nombres de voitures |
| *EtatDep[i]* | Tableau de dimension 1 de nombres entiers | Répartition initiale des voitures dans les sites |
| *TailleCycle* | Nombre entier | Nombre de périodes élémentaires |
| *NbMaxDem[i][j][t]* | Tableau de dimension 3 de nombres entiers | Nombre maximal de demandes dans chaque site pour aller du site $i$ au site $j$ en partant à la période élémentaire $t$ |
| *Proba[i][j][n][t]* | Tableau de dimension 4 de nombres réels | Probabilités de demandes entre chaque couple de sites $(i, j)$ pour chacune des périodes élémentaires |
| *DureeMin[i][j][t]* | Tableau de dimension 3 de nombres entiers | Durée minimale d'une course entre le site $i$ et le site $j$ lorsque la voiture part à la période $t$ |
| *DureeEcart[i][j][t]* | Tableau de dimension 3 de nombres entiers | Durée maximale d'une course = DureeMin + DureeEcart |
| *DateDeb* | Nombre entier | Date du début de la simulation |
| *DureeSim* | Nombre entier | Durée de la simulation |
| *NbSim* | Nombre entier | Nombre de simulations |
| *NbPerMec* | Nombre entier | Attente maximale qu'un client tolère |
| *NbCam* | Nombre entier | Nombre de camions |
| *CapaCam[i]* | Tableau de dimension 1 de nombres entiers | Capacité de chacun des camions |
| *EmplCam[c][0/1]* | Tableau de dimension 2 de nombres entiers | EmplCam[$c$][0] est l'emplacement du camion $c$, et EmplCam[$c$][1] est le moment où il l'atteindra |
| *DureeCam[i][j]* | Tableau de dimension 2 de nombres entiers | Durée moyenne nécessaire aux camions pour parcourir la distance |

*suite page suivante*

| *suite de la page précédente* | | |
|---|---|---|
| Nom de la variable | Description | Commentaires |
| | | entre chaque couple de sites |
| *Seuils[i]* | Tableau de dimension 1 de nombres réels | Les seuils de disponibilité pour chaque site |
| *PourFatigue* | Nombre réel | Charge consommée par une voiture en mouvement par période élémentaire |
| *Recharge* | Nombre réel | Charge ajoutée à la charge d'une voiture lorsque celle-ci se situe sur un plot pendant une période élémentaire |
| *Distrib* | Nombre entier | Politique de redistribution choisie |
| *Rech* | Nombre entier | Politique de recharge choisie |
| *JockeyPres* | Nombre entier | Fréquence des passages des jockeys dans les sites pour les déplacements locaux |
| *HorizonDef* | Nombre entier | Horizon nécessaire pour le calcul des états défavorables |
| *HorizonMin* | Nombre entier | Horizon nécessaire pour le calcul des états favorables |

**B.** La structure des données *DemandeStructure* caractérisant une demande émise par un client, à savoir :

TAB. 1.2: *La structure des données DemandeStructure*

| Nom de la variable | Description | Commentaires |
|---|---|---|
| *SiteArrivee* | Nombre entier | Le site de destination d'une course |
| *DateDemande* | Nombre entier | Date où le client s'est présenté dans le site |

**C.** La structure des données *VoitureStructure* contient les informations concernant une voiture, à savoir :

TAB. 1.3: *La structure des données VoitureStructure*

| Nom de la variable | Description | Commentaires |
|---|---|---|
| *SurBorne* | Booléen | Booléen indiquant qu'une voiture se trouve sur une place munie d'un chargeur ou pas |
| *Mode* | Nombre entier | Les différents modes d'une voiture sont :<br>*MODEINITIA* → au début de la simulation, les voitures sont complètement chargées et disponibles<br>*MODESTATIO* → voiture stationnée<br>*MODEDEPLAC* → voiture en course |
| *suite page suivante* | | |

| *suite de la page précédente* | | |
|---|---|---|
| Nom de la variable | Description | Commentaires |
| | | *MODECHARGE* → voiture en train se recharger<br>*MODEENCOMB* → voiture stationnée en dehors des places de parking<br>*MODEPANNE* → voiture en panne<br>*MODETRANSP* → voiture transportée par un camion<br>*MODEPLEINE* → voiture complètement chargée |
| *Charge* | Nombre réel | Niveau de charge de la voiture |
| *SiteApresPanne* | Nombre entier | Site dans lequel la voiture est remorquée après une panne |
| *TempsMonopolise* | Nombre entier | Durée pendant laquelle la voiture est monopolisée, le temps de la rapatrier |

**D.** La structure des données *RééquilibrageStructure* contient les résultats du Simplexe, à savoir :

TAB. 1.4: *La structure des données RééquilibrageStructure*

| Nom de la variable | Description | Commentaires |
|---|---|---|
| *SiteArrivee* | Nombre entier | Site de destination d'un rééquilibrage |
| *NbVoitures* | Nombre entier | Nombre de voitures à déplacer |

**E.** La structure des données *EtatDefavorableStructure* contient les bornes minimales et maximales d'après lesquelles un état est classé défavorable ou non. Lorsque le nombre de voitures dans un site est inférieur à la borne minimale, celui-ci indique que la situation actuelle est classée défavorable. Un encombrement est fort probable lorsque le nombre de voitures dans un site est supérieur à la borne maximale.

TAB. 1.5: *La structure des données des EtatDefavorableStructure*

| Nom de la variable | Description | Commentaires |
|---|---|---|
| *NiveauMinimal[i]* | Tableau de dimension 1 de nombres entiers | Borne minimale de chacun des sites |
| *NiveauMaximal[i]* | Tableau de dimension 1 de nombres entiers | Borne maximale de chacun des sites |

**F.** La structure des données *EtatFavorableStructure* contient les résultats obtenus lors du calcul des états favorables, à savoir :

TAB. 1.6: *La structure des données EtatFavorableStructure*

| Nom de la variable | Description | Commentaires |
|---|---|---|
| *NbMinimal* | Nombre entier | Plus petit nombre de voitures réparties dans les sites pour lesquelles un état favorable est possible |
| *NbMaximal* | Nombre entier | Plus grand nombre de voitures réparties dans les sites pour lesquelles un état favorable est possible |
| *NbMinMeilleur* | Nombre entier | Nombre minimal de voitures réparties dans les sites pour lesquelles un état favorable est obtenu avec la plus forte probabilité |
| *NbMaxMeilleur* | Nombre entier | Nombre maximal de voitures réparties dans les sites pour lesquelles un état favorable est obtenu avec la plus forte probabilité |
| *Horizon[i]* | Tableau de dimension 1 de nombres entiers | Les différents horizons pour lesquels un état favorable est possible |
| *Répartition[n][i]* | Tableau de dimension 2 de nombres entiers | La répartition de $n$ voitures dans les sites du système |

**G.** La structure des données $La-SimulationStructure$ contient les paramètres nécessaires à la simulation, à savoir :

TAB. 1.7: *La structure des données de* $La - SimulationStructure$

| Nom de la variable | Description | Commentaires |
|---|---|---|
| *DateCourante* | Nombre entier | Date à laquelle on se situe au cours d'une simulation |
| *Mouvement* | Nombre entier | Nombre de voitures qui effectuent un trajet entre deux sites |
| *VoitAjoutees[i]* | Tableau de dimension 1 de nombres entiers | Nombre de voitures ajoutées à chaque site lors du calcul des bornes minimales et maximales |
| *VoitAffectable* | Liste chaînée contenant des éléments *VoitureStructure* | Liste *Stock* des voitures qui ne sont pas affectées et qui peuvent être mises en service |

*suite page suivante*

| *suite de la page précédente* | | |
|---|---|---|
| Nom de la variable | Description | Commentaires |
| *VoitPerdues* | Liste chaînée contenant des éléments *VoitureStructure* | Liste des voitures qui sont tombées en panne lors d'une course |
| *EtatCourant[i]* | Tableau de dimension 1 de listes chaînées contenant des éléments *VoitureStructure* | Liste des voitures présentes dans chaque site à la date courante |
| *Flux[i][t]* | Tableau de dimension 2 de listes chaînées contenant des éléments *VoitureStructure* | Liste des voitures qui sont en déplacement et qui se dirigent vers les sites du système |
| *ListeDemande[i]* | Tableau de dimension 1 de listes chaînées contenant des éléments *DemandeStructure* | Liste des demandes formulées dans chaque site. Cette liste comprend les clients en attente |
| *VoitDsCams[c]* | Tableau de dimension 1 de listes chaînées contenant des éléments *VoitureStructure* | Liste des voitures qui sont transportées dans chacun des camions |
| *CycleDuCam[i]* | Tableau de dimension 1 de nombres entiers | Ordre de visite des différentes sites lors de la politique *Trajets cycliques des camions* |
| *DureeCycleCam* | Nombre entier | Durée nécessaire aux camions pour la visite cyclique des différents sites |
| *RepFav[t]* | Tableau de dimension 1 contenant des éléments de *EtatFavorableStructure* | Différentes répartitions favorables pour chacune des périodes élémentaires du cycle |
| *RepDefav[t]* | Tableau de dimension 1 contenant des éléments de *EtatDefavorableStructure* | Les bornes minimales et maximales concernant les voitures présentes dans les sites pour chaque période élémentaire du cycle |
| *DiffTrajets[c]* | Tableau de dimension 1 de listes chaînées contenant des éléments *RééquilibrageStructure* | Résultats du Simplexe : les trajets de rééquilibrage à effectuer lors des politiques de redistribution 1, 2, et 3 |

**H.** La structure des données *SimulationResultats* contient les résultats obtenus à la suite des différentes simulations, à savoir :

TAB. 1.8: *La structure des données de SimulationResultats*

| Nom de la variable | Description | Commentaires |
|---|---|---|
| *NbTotalDem[i]* | Tableau de dimension 1 de nombres entiers | Nombre total de demandes formulées dans chaque site |
| *NbTotalArriv[i]* | Tableau de dimension 1 de nombres entiers | Nombre total d'arrivées dans chaque site |
| *suite page suivante* | | |

| *suite de la page précédente* | | |
|---|---|---|
| Nom de la variable | Description | Commentaires |
| *NbVoitPanne[i]* | Tableau de dimension 1 de nombres entiers | Nombre de voitures qui sont en panne dans chaque site |
| *NbDemRefParPeriode[i]* | Tableau de dimension 1 de nombres entiers | Nombre de demandes refusées dans chaque site |
| *NbClientsEncomb[i]* | Tableau de dimension 1 de nombres entiers | Nombre de clients qui ont été obligés de laisser leur voiture en dehors des places de parking |
| *NbClPrt[i]* | Tableau de dimension 1 de nombres entiers | Nombre de clients qui sont partis faute de voitures disponibles |
| *NbClientsMecontents[i]* | Tableau de dimension 1 de nombres entiers | Total des clients insatisfaits |
| *NbClServis[i]* | Tableau de dimension 1 de nombres entiers | Total des clients satisfaits dans chaque site |
| *NbVoitDisp[i]* | Tableau de dimension 1 de nombres entiers | Nombre de voitures disponibles dans chaque site |

### 1.2.6 Fonctionnalité des principales procédures

Cette section contient les explications de quelques fonctions du programme de simulation. Nous détaillons la fonctionnalité de chacune, ainsi que ses entrées et ses sorties.

1. *LectureAllocationDonneesSysteme* : le rôle de cette fonction est de saisir tous les paramètres entrés par l'utilisateur, et de renvoyer un booléen indiquant le bon déroulement de la fonction. Dans cette fonction, les places mémoires sont allouées dynamiquement aux différentes variables.

TAB. 1.9: *LectureAllocationDonneesSysteme*

| **Nom** | *LectureAllocationDonneesSysteme* |
|---|---|
| **Entrées** | Nom du fichier dans lequel l'utilisateur a paramétré le système |
| **Sorties** | * Booléen indiquant que la lecture est correctement effectuée<br>* Structure des données de type *DonneesStructure* contenant tous les paramètres |

2. *VerificationDesDonnees* : cette fonction vérifie si les paramètres du système sont complets et cohérents. Si la réponse est négative, le programme de simulation est stoppé.

TAB. 1.10: *VerificationDesDonnees*

| **Nom** | *VerificationDesDonnees* |
|---|---|
| **Entrées** | Structure des données de type *DonneesStructure* |
| **Sorties** | Booléen indiquant que les paramètres sont cohérents |

3. *DesalocationDonneesSysteme*: à la fin de la simulation, la fonction libère toutes les places mémoires allouées pour les variables de la structure des données *DonneesStructure* .

TAB. 1.11: *DesalocationDonneesSysteme*

| **Nom** | *DesalocationDonneesSysteme* |
|---|---|
| **Entrées** | Structure des données de type *DonneesStructure* |

4. *Calcul-Cycle-Camion-Pour-Journee*: cette fonction détermine le trajet cyclique des camions lorsque la politique de redistribution *Trajets cycliques des camions* est appliquée.

TAB. 1.12: *Calcul-Cycle-Camion-Pour-Journee*

| **Nom** | *Calcul-Cycle-Camion-Pour-Journee* |
|---|---|
| **Entrées** | * Nombre de sites<br>* Durées moyennes entre tout couple de sites |
| **Sorties** | * Ordre de visites des différents sites, autrement dit trajet cyclique<br>* Durée du cycle |

5. *CalculSeuils*: cette procédure calcule les bornes minimales et maximales pour la politique de redistribution *Trajets cycliques des camions*. Ainsi, lorsqu'un camion arrive dans un site, suivant le nombre de voitures présentes et les bornes calculées, des voitures sont chargées (dans le) ou déchargées du camion.

TAB. 1.13: *CalculSeuils*

| **Nom** | *CalculSeuils* |
|---|---|
| **Entrées** | Durée nécessaire aux camions pour effectuer un cycle |
| **Sorties** | Les bornes minimales et maximales pour chacun des sites et pour chacune des périodes élémentaires |

6. *CalculDiffEtats* : cette fonction calcule les différents états favorables ainsi que les différents états défavorables pour chacune des périodes élémentaires. Ces états sont nécessaires lors de l'application des politiques de redistribution *Priorité au site le plus proche d'un camion*, *Service groupés*, et *Priorité au site le plus éloigné d'un camion*.

TAB. 1.14: *CalculDiffEtats*

| **Nom** | *CalculDiffEtats* |
|---|---|
| **Entrées** | Structure des données de type *DonneesStructure* |
| **Sorties** | * Les états favorables pour chaque période élémentaire<br>* Les états défavorables pour chaque période élémentaire |

7. *La-Simulation-main* : cette fonction est la fonction principale qui gère la simulation, autrement dit, elle appelle la fonction qui initialise la structure des données de type *SimulationStructure*, ainsi que la structure des données de type *SimulationResultats*. La simulation est effectuée, les résultats sont écrits dans les fichiers correspondants et, enfin, toutes les places mémoires sont libérées.

TAB. 1.15: *La-Simulation-main*

| **Nom** | *La-Simulation-main* |
|---|---|
| **Entrées** | * Structure des données de type *DonneesStructure*<br>* Durée nécessaire aux camions pour effectuer un trajet cyclique<br>* Ordre des visites des différents sites, autrement dit le trajet cyclique<br>* Les états favorables pour toutes les périodes élémentaires<br>* Les états défavorables pour toutes les périodes élémentaires |

8. *Simulation* : cette fonction considère les différents évènements décrits précédemment qui ont lieu pendant la période élémentaire considérée de la simulation, et met à jour les différentes listes.

TAB. 1.16: *Simulation*

| **Nom** | *Simulation* |
|---|---|
| **Entrées** | * Booléen qui indique si nous sommes dans la phase du calcul des états favorables et défavorables ou non<br>* La structure des données *DonneesStructure* |
| **Sorties** | * Mise à jour de la structure des données *SimulationStructure*<br>* Mise à jour de la structure des données *SimulationResultats* |

9. *LesStatistiques* : cette fonction établit les statistiques, et calcule le taux de satisfaction des clients qui ont utilisé le système.

TAB. 1.17: *LesStatistiques*

| **Nom** | *LesStatistiques* |
|---|---|
| **Entrées** | * Nombre de sites<br>* Durée de la simulation<br>* Nombre de simulations<br>* Structure des données *SimulationRésultats*<br>* Nom du fichier de sortie |
| **Sorties** | Les statistiques écrites dans un fichier |

10. *ResoudPL* : cette fonction appelle le programme du simplexe qui détermine les trajets de rééquilibrage nécessaires à faire basculer la situation actuelle, qui est classée défavorable, vers une situation classée favorable.

TAB. 1.18: *ResoudPL*

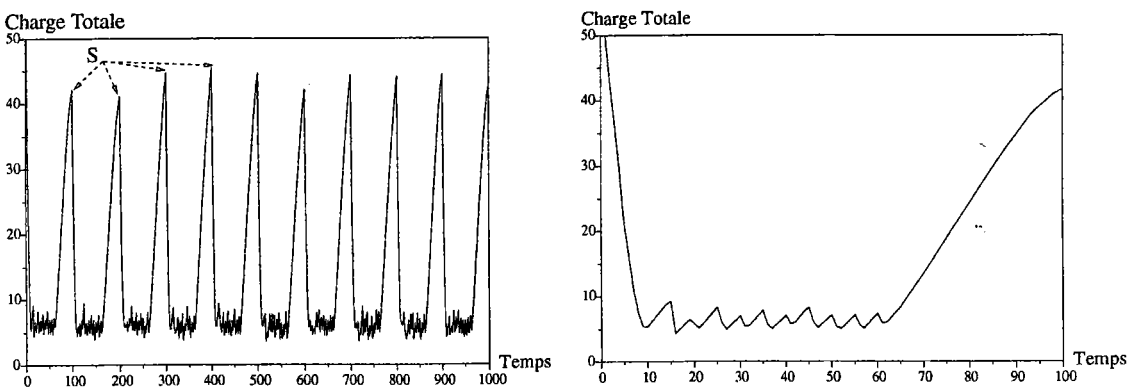
| **Nom** | *ResoudPL* |
|---|---|
| **Entrées** | * Nombre de sites<br>* Répartition actuelle qui est classée défavorable<br>* Répartition favorable correspondante<br>* Durées nécessaires pour parcourir la distance entre tout couple de sites |
| **Sorties** | Déplacements de voitures à effectuer afin de ramener une situation classée défavorable vers une situation favorable |

### 1.2.7 Résultats des simulations

A la fin des simulations, pour chacun des sites nous obtenons :

1. Le nombre total de demandes formulées dans chacun des sites.
2. Le nombre de clients qui ont été servis, autrement dit, qui ont trouvé une voiture disponible dans le site.
3. Le nombre de clients qui au bout d'une attente sont partis mécontents.
4. Le nombre de clients qui sont tombés en panne.
5. Le pourcentage de clients satisfaits. Un client est satisfait lorsqu'il trouve une voiture disponible et lorsqu'il effectue sa course sans encombres.

A partir des résultats obtenus, nous avons tracé différentes courbes, voir figures 1.1, 1.2 et 1.3. La figure 1.1 représente la courbe de la charge totale répartie sur les différentes voitures du système, tout le long de la simulation. Dans la suite de ce chapitre, nous l'appelons *'la courbe des charges'*. Pour l'exemple étudié, la simulation se fait pour dix jours, et les demandes sont nulles pendant la nuit. Notre système est constitué de six sites et de cinquante voitures. Une journée comprend 100 périodes élémentaires. Trois camions, de capacité un, sont chargés du rééquilibrage. La politique de rééquilibrage suivie est la politique appelée *Priorité au site le plus proche d'un camion.* Concernant la recharge, c'est la seconde politique de recharge qui est implémentée. Les sites comprennent respectivement 10, 10, 5, 5, 5, 5 bornes de recharge. Le personnel chargé du rééquilibrage inter-sites est présent toutes les demie-heures. La figure 1.1 (b) détaille la courbe des charges obtenue pendant la première journée. Une voiture est complètement chargée lorsque son niveau de charge atteint le niveau '1'. Par conséquent, la charge totale maximale pouvant être atteinte dans cet exemple est 50 car notre système comprend cinquante voitures.




(a) La période de simulation est dix jours

(b) Détails du premier jour

FIG. 1.1: *Charge totale répartie sur les voitures du système*

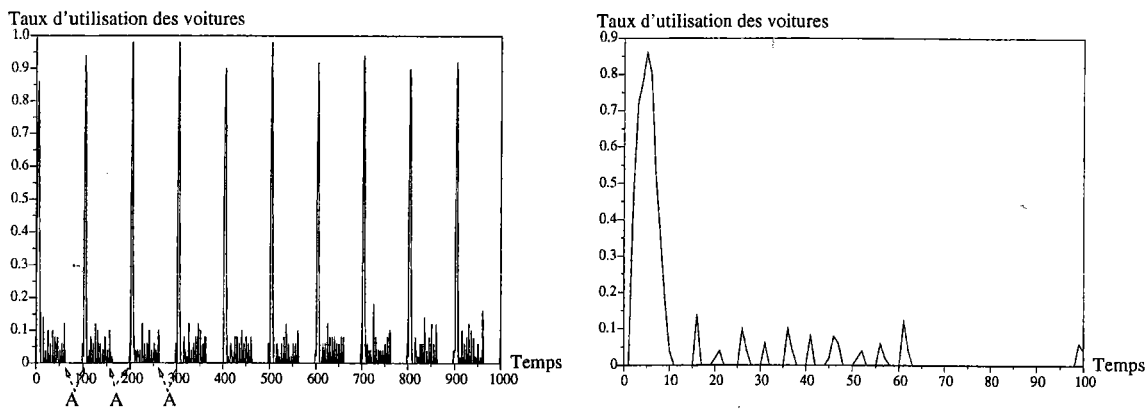
Comme les demandes sont nulles pendant la nuit, les voitures ont le temps de se recharger, d'où les différents sommets (identifiés par 'S') que nous distinguons dans la figure 1.1 (a). Nous remarquons que ces sommets n'atteignent jamais la charge totale maximale mais, se situent autour d'un niveau inférieur. Ceci est dû au fait qu'il y a un nombre limité de plots dans les différents sites, et qu'une voiture électrique a besoin de huit heures pour se recharger complètement.

Au cours de la journée, la charge totale tourne autour d'une valeur inférieure au seuil de disponibilité. Dans l'exemple considéré, une voiture est autorisée à quitter un site, si son niveau de charge est supérieur ou égal à 0.3. Comme les demandes sont fortes, dès que le niveau de charge d'une voiture atteint le seuil de disponibilité, celle-ci devient disponible pour tout client qui se présente. Il est donc normal que la charge totale se situe en dessous du seuil de disponibilité puisque les voitures, dès qu'elles sont disponibles, sont utilisées et consomment de l'énergie durant les différentes courses.

La seconde courbe extraite des résultats obtenus, voir figure 1.2, est la courbe qui reflète le taux d'utilisation des voitures. Autrement dit, cette courbe est le résultat du rapport suivant :

$$\frac{\text{Nombre de voitures en course + Nombre de voitures en rééquilibrage}}{\text{Nombre total de voitures}}$$

Les voitures en course sont les voitures utilisées par les clients du système de transport en libre service. Les voitures en rééquilibrage sont les voitures présentes dans les camions et qui sont transportées d'un site vers un autre. Le nombre total de voitures comprend les voitures disponibles (dont le niveau de charge est supérieur au seuil de disponibilité) et les voitures indisponibles du système.




(a) La période de simulation est dix jours

(b) Détails du premier jour

FIG. 1.2: *Taux d'utilisation des voitures*

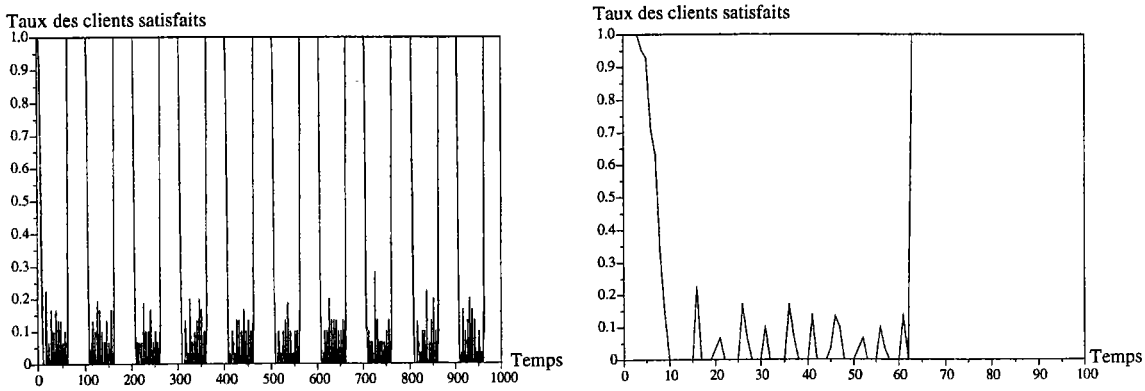
Dans la figure 1.2, les tranches horaires $A$ correspondent à la nuit, où le taux d'utilisation des voitures est nul puisqu'aucune demande n'est formulée. La nuit, les voitures ont le temps de se recharger, ainsi, les premiers clients qui arrivent le matin trouvent des voitures disponibles. C'est pour cette raison que le taux d'utilisation des voitures est élevé au début de la journée.

La dernière courbe détaillée est la courbe qui reflète le taux de satisfaction des clients. Cette courbe résulte du rapport suivant :

$$\frac{\text{Clients satisfaits}}{\text{Clients satisfaits + clients insatisfaits}}$$

Les clients satisfaits sont les clients qui se sont présentés dans un site et ont trouvé une voiture disponible qui les a amenés à destination sans encombres. Ces clients ont peut être

attendu un petit moment avant d'être servis. Les clients insatisfaits sont soit les clients qui sont tombés en panne pour cause d'insuffisance d'autonomie, soit les clients qui sont partis mécontents au bout d'un certain temps d'attente sans avoir eu accès à une voiture disponible.




(a) La période de simulation est dix jours (b) Détails du premier jour

FIG. 1.3: *Taux de satisfaction des clients*

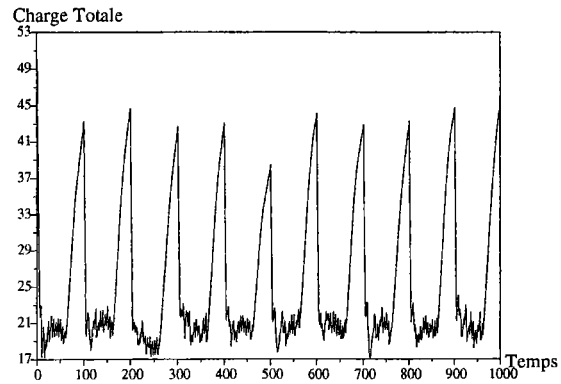
Le taux de satisfaction des clients est maximal pendant les périodes qui correspondent à la nuit. Ceci est évident puisqu'aucune demande n'est formulée. Le taux de satisfaction des clients est élevé au début de la journée et connait une baisse avec des fluctuations pour le restant de la journée. En fait, comme la majorité des voitures sont disponibles au bout d'une nuit de recharge, les permiers clients qui arrivent sont satisfaits. Ce taux de satisfaction des clients baisse le long de la journée puisque le nombre de voitures disponibles baisse aussi.

En examinant la figure 1.2 (b), nous constatons qu'au cours de la journée, le taux d'utilisation peut atteindre le niveau zéro. En comparant cette courbe aux deux autres courbes, voir figures 1.1 (b) et 1.3 (b), nous remarquons que lorsque le taux d'utilisation des voitures est nul, le taux de satisfaction des clients est nul aussi, de même que la charge totale des voitures est faible. Ceci indique que les clients sont insatisfaits parce qu'ils ne trouvent pas de voitures disponibles.

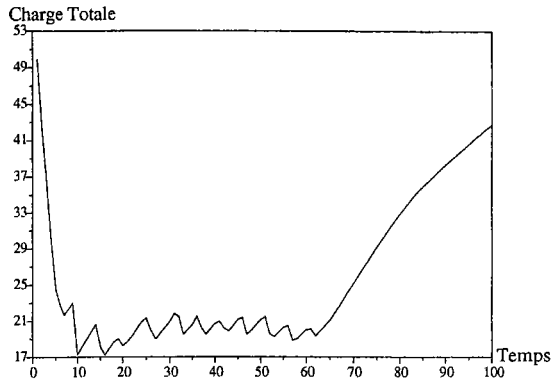
Dans la simulation suivante, nous faisons varier le seuil de disponibilité qui est fixé dans les différents sites du système. Le seuil de disponibilité devient 0.6 au lieu du 0.3 de la simulation précédente. La figure 1.4 regroupe toutes les courbes obtenues. La première remarque que nous effectuons est que la courbe des charges atteint un niveau minimal nettement supérieur au niveau atteint dans la simulation précédente, voir figure 1.1. En fait, ce détail est logique puisque les voitures qui partent (disponibles) ont un niveau de charge nettement supérieur.

Lorsque nous comparons les deux courbes 1.3 (a, b) et 1.4 (e, f) qui représentent le taux de satisfaction des clients, nous remarquons que la seconde courbe comprend des fluctuations plus grandes. Ceci est dû au fait de l'élévation du seuil de disponibilité, et par conséquent,

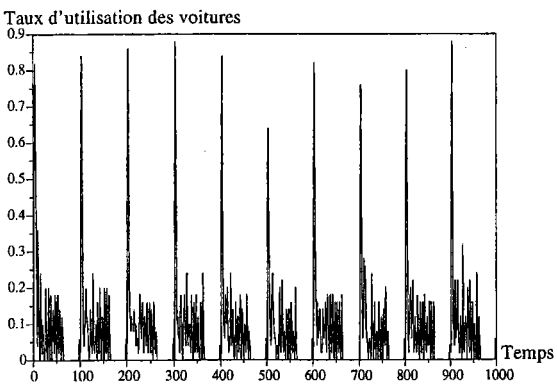
de la baisse des pannes d'énergie. Lorsqu'une voiture tombe en panne, celle-ci est mobilisée le temps qu'une dépanneuse aille la récupérer en plus du temps nécessaire pour la recharger.



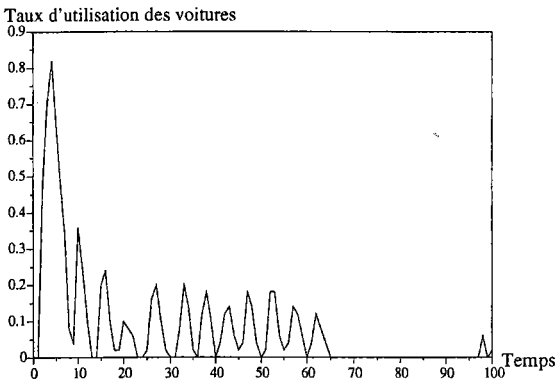

(a) Charge répartie sur les voitures



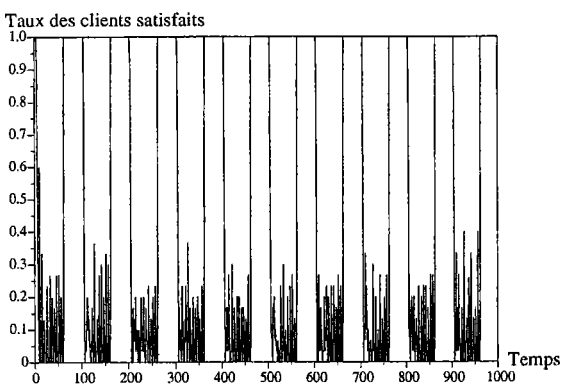

(b) Détails du premier jour



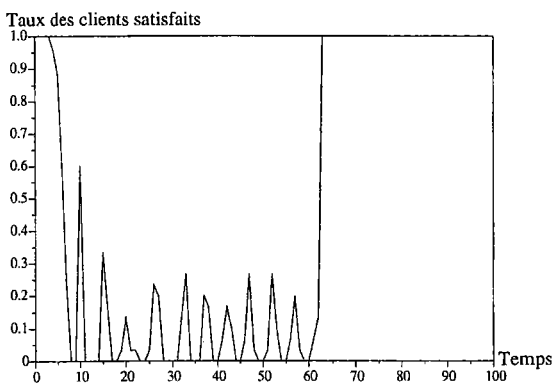

(c) Taux d'utilisation des voitures




(d) Détails du premier jour

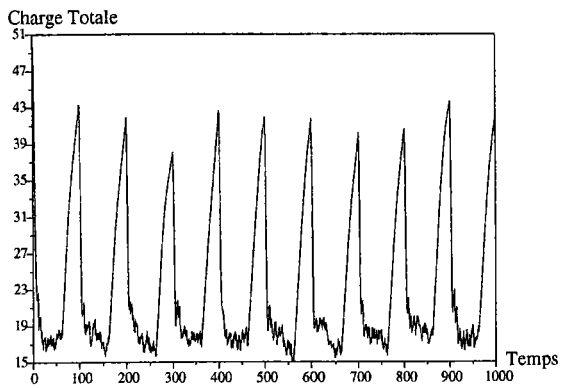



(e) Le taux de satisfaction des clients

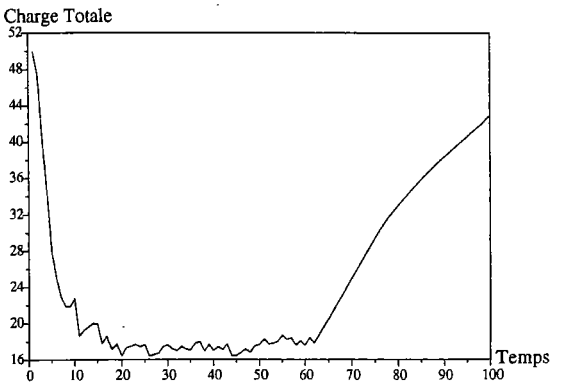



(f) Détails du premier jour

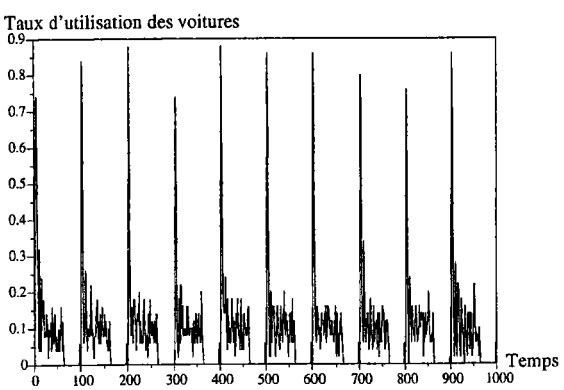
FIG. 1.4: *Modification du seuil de disponibilité = 0.6*



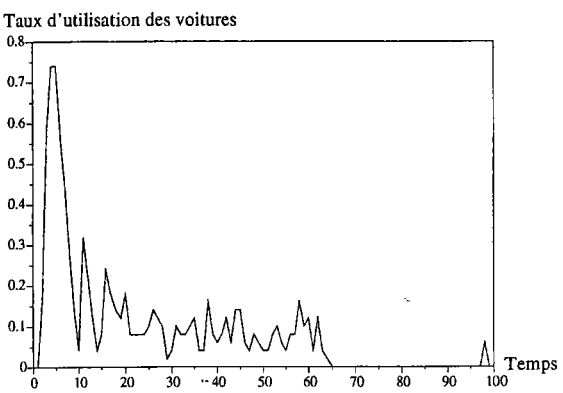

(a) Charge répartie sur les voitures



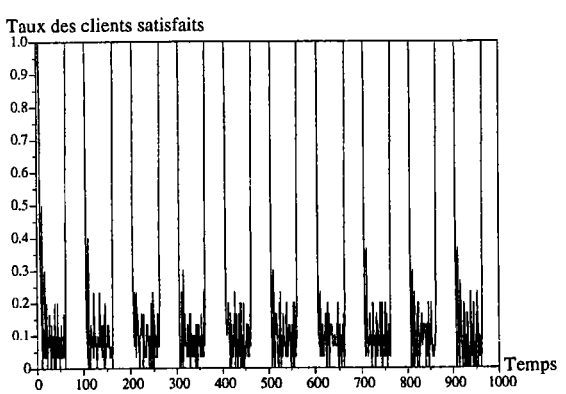

(b) Détails du premier jour



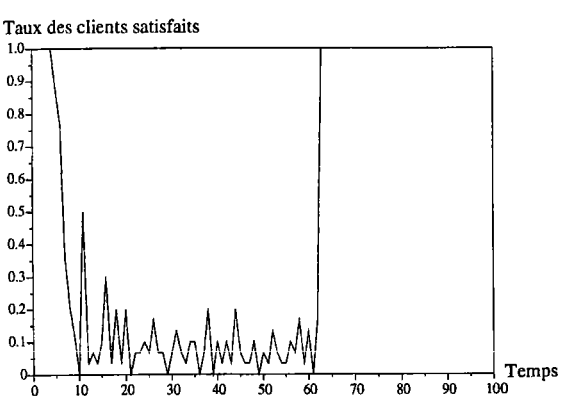

(c) Taux d'utilisation des voitures




(d) Détails du premier jour




(e) Le taux de satisfaction des clients




(f) Détails du premier jour

FIG. 1.5: *Modification des demandes formulées dans les différents sites*

Si nous regardons attentivement la première fluctuation des courbes 1.3 (b) et 1.4 (f), nous remarquons que le taux de satisfaction des clients dans la figure 1.4 (f) atteint le niveau zéro plus rapidement. En fait, lors de la seconde simulation, comme le seuil de satisfaction est plus élevé, les voitures sont plus rapidement mobilisée pour cause de niveau de charge.

Par hypothèse, les camions chargés de la redistribution transportent des voitures disponibles. Dans la seconde simulation, les voitures tombent moins souvent en panne, et par conséquent, ont plus souvent l'occasion de se recharger. Les camions peuvent ainsi transporter plus de voitures, et satisfaire plus de clients.

Dans la simulation suivante, nous avons diminué considérablement les demandes formulées dans les différents sites, et nous avons gardé le plus haut seuil de disponibilité des deux simulations précédentes. La figure 1.5 reflètent les résultats obtenus le long de la simulation.

Dans ce paragraphe, nous comparons la figure qui représente le taux de satisfaction des clients de cette simulation avec celle de la simulation précédente. Nous remarquons que la courbe de celle-ci comprend de plus petites fluctuations, de même qu'elle atteint moins souvent le taux nul. Dans la simulation précédente, beaucoup de demandes de clients sont refusées parce que les voitures ne sont pas disponibles, et c'est pour cela que nous avons des tranches horaires où le taux est égal à zéro. Lorsque nous comparons les deux courbes qui représentent le taux d'utilisation des voitures, nous remarquons que celle-ci est plus élevée lorsque les demandes sont plus faibles, et c'est parce que les voitures ont plus de temps pour se recharger.

## 1.3 Conclusion

Dans ce chapitre, nous avons présenté le simulateur regroupant toutes les politiques décrites dans ce mémoire. Le simulateur a été utilisé pour simuler la gestion du projet *Praxitèle* instauré à Saint Quentin en Yvelines et établir quelques comparaisons. Quelques fonctionnalités de ce simulateur ont effectivement été introduites dans la programme de gestion de *Praxitèle* tel que le calcul des bornes minimales et maximales, et ceci afin de signaler à l'exploitant qu'un problème va probablement avoir lieu sur l'un ou plusieurs des sites. Pour Saint Quentin en Yvelines, nous avons proposé l'interface homme-machine, voir l'annexe, qui a été partiellement intégré dans le logiciel de gestion de *Praxitèle*.

# Sixième partie

# Conclusion

# *Conclusion et perspectives*

Dans ce mémoire, nous avons présenté différentes méthodes pour gérer efficacement un système de voitures en libre service en complément des transports en commun. L'idée innovante du système consiste à mettre des voitures à la disposition des clients dans plusieurs sites. Après un certain nombre de courses, notre système peut être complètement déséquilibré. Autrement dit, des sites se trouvent submergés par des voitures alors que d'autres, au contraire, se retrouvent sans voitures. Nous avons proposé deux méthodes différentes pour rééquilibrer le système. La première est le rééquilibrage préventif, et la seconde est le rééquilibrage cyclique.

*Le rééquilibrage préventif* comprend trois étapes. La première, grâce à la simulation, détermine le moment où il faut initialiser une procédure de rééquilibrage. La seconde, utilisant le modèle de transport, calcule le nombre de voitures qu'il faut déplacer ainsi que les différentes destinations. La dernière étape construit les tournées que devront effectuer les camions ou voitures de service afin de s'acquitter du rééquilibrage dans les meilleurs délais. Nous avons utilisé des heuristiques gloutonnes lors de la construction de ces tournées.

*Le rééquilibrage cyclique* consiste à faire visiter les sites de façon continue par les différents camions chargés de la redistribution des voitures. Lorsqu'un camion arrive dans un site déficitaire, il décharge des voitures. Si, au contraire, il existe un excédent de voitures dans le site, des voitures seront chargées dans le camion. Aucune action n'est entreprise lorsque le nombre de voitures présentes dans le site est satisfaisant et suffisant. Nous avons proposé deux méthodes différentes, la première est basée sur la simulation et la seconde, quant à elle, est basée sur la programmation dynamique.

Ce service innovant utilise des voitures électriques qui sont non polluantes et qui vont dans le sens de la loi qui impose aux administrations et entreprises publiques d'acheter au moins une voiture non polluante sur cinq. Les voitures électriques présentent cependant un certain nombre d'inconvénients tels qu'une autonomie limitée, un temps de recharge et un coût de l'infrastructure correspondante élevé. Nous avons proposé un critère dont la minimisation assure un bon compromis entre la disponibilité des voitures dans les sites et leur capacité à assurer des trajets sans tomber en panne. Ainsi nous avons calculé le seuil de disponibilité. Une voiture peut être utilisée par un client si son niveau de charge est supérieur au seuil de disponibilité. Nous avons ensuite répondu au problème posé lorsque le nombre de chargeurs est insuffisant pour traiter les voitures présentes dans un site. Pour ce problème, nous avons proposé plusieurs politiques dont les objectifs sont différents, comme par exemple satisfaire le maximum de clients ou optimiser la vie des batteries des voitures.

Le mémoire se termine par un simulateur qui regroupe toutes les politiques développées. Une simulation conduit au taux de satisfaction des clients, sachant que le but d'une bonne gestion du système est d'obtenir le plus haut taux de satisfaction possible au moindre coût.

Le paragraphe suivant liste quelques paramètres supplémentaires qui pourraient être pris en considération lors de la formulation des différents problèmes :

- ⋆ le coût de l'électricité : le prix de l'électricité peut être différent suivant l'heure de la journée. Par exemple, il est plus avantageux de charger les voitures pendant la nuit où les coûts sont minima.

- ⋆ La tarification des courses : ce paramètre peut influencer la redistribution des voitures sur le site. En fait l'idée serait de faire varier le prix des courses des clients selon l'horaire du début de la course. Il peut même être envisageable de créer des courses gratuites entre deux sites à certaines périodes de la journée, ce qui reviendrait, peut être, moins cher que d'utiliser le système de rééquilibrage qui monopolise des camions et des conducteurs.

- ⋆ La possibilité de réservations : une voiture peut être disponible dans un site mais non utilisable parce qu'un client l'a réservée. Une nouvelle contrainte est ajoutée lors du lancement du processus de la redistribution des voitures.

- ⋆ L'uniformité de l'usure de la flotte : veiller à distribuer les voitures aux clients de manière à ce que les kilométrages des voitures restent voisins.

- ⋆ Les fenêtres de temps de la redistribution : les jockeys chargés de la redistribution des voitures peuvent avoir un emploi du temps et ne pas être disponibles à longueur de journée.

# Perspectives

1. Lors du rééquilibrage préventif, nous avons procédé par étapes. Autrement dit, nous avons commencé par l'optimisation du coût de rééquilibrage où nous avons déterminé le nombre de voitures à déplacer. Ensuite nous avons enchaîné par l'optimisation du temps de rééquilibrage, où nous avons attribué à chaque camion une tournée à effectuer. En fait, il se pourrait qu'avec cette démarche, nous nous éloignions de l'optimum. D'autres méthodes (peut être plus efficaces) peuvent être investiguées.

2. Nous avons utilisé un système qui comprend plusieurs sites dans lesquels sont réparties des voitures dont le nombre total est donné. Quelle sera la situation si plusieurs systèmes similaires cohabitent dans une même ville et qu'il existe une interface entre les différents systèmes? Comment gérer la redistribution sachant que les voitures n'appartiennent pas obligatoirement au système dans lequel elles se trouvent? Comment créer l'interaction entre les différents systèmes?

3. Lors de la redistribution des voitures dans les sites, nous avons proposé un rééquilibrage cyclique. Un camion visite de manière périodique les différents sites du système. Un avantage serait de donner au camion la liberté de choisir sa trajectoire, ou bien d'être réactif. Autrement dit, si passer dans un site n'apporte rien au processus de rééquilibrage, mieux vaut le court-circuiter.

4. Comme nous l'avons dit précédemment, le principe de notre système consiste à offrir une voiture disponible au client qui se présente dans un site. Celui-ci l'utilise et, ensuite, la gare dans un site. Le problème se complique si le client est autorisé à laisser sa voiture n'importe où dans un périmètre donné. Ce problème reste ouvert.

# Septième partie

# Annexes

# Annexe A

# Démonstration : L'espérance du nombre de demandes à un site est la somme des espérances du nombre de clients servis et de ceux mécontents

La relation $E[V(s)] = E[K] - E[M(s)]$ peut être établie comme suit.

Deux cas sont à considérer quand on veut déterminer le nombre de voitures qui quittent un site, à savoir :

1. Toutes les demandes de clients sont satisfaites car le nombre de voitures disponibles dans le site est suffisant pour les couvrir toutes.

2. Toutes les demandes des clients ne sont pas satisfaites car le nombre de voitures disponibles dans le site n'est pas suffisant. Seules les voitures disponibles sont autorisées à quitter le site.

Par conséquent :

$$E[V(s)] = \sum_{v=0}^{N} v\, P(K=v) \sum_{d=0\ d>v}^{N} P(D=d) + \sum_{d=0}^{N} d\, P(D=d) \sum_{v=0\ d\leq v}^{N} P(K=v)$$

En prenant le complémentaire du dernier facteur du premier terme, nous obtenons :

$$E[V(s)] = \sum_{v=0}^{N} v\, P(K=v)\, [1 - \sum_{d=0\ d\leq v}^{N} P(D=d)] + \sum_{d=0}^{N} d\, P(D=d) \sum_{v=0\ d\leq v}^{N} P(K=v)$$

En développant la formule précédante, nous obtenons :

$$E[V(s)] = \sum_{v=0}^{N} v\, P(K=v) - \sum_{v=0}^{N} v\, P(K=v) \sum_{d=0\ d\leq v}^{N} P(D=d) + \sum_{d=0}^{N} d\, P(D=d) \sum_{v=0\ d\leq v}^{N} P(K=v)$$

Ou bien :

$$E[V(s)] = \sum_{v=0}^{N} v\, P(K=v) - \sum_{v=0}^{N} \sum_{d=0\ d\leq v}^{N} v\, P(K=v)\, P(D=d) + \sum_{v=0}^{N} \sum_{d=0\ d\leq v}^{N} d\, P(D=d)\, P(K=v)$$

En regroupant le second et troisième termes, nous obtenons :

$$E[V(s)] = \sum_{v=0}^{N} v\, P(K=v) - \sum_{v=0}^{N} \sum_{d=0\ d\leq v}^{N} P(D=d)\, P(K=v)\, (v-d)$$

Si w = v - d, alors :

$$E[V(s)] = \sum_{v=0}^{N} v\, P(K=v) - \sum_{v=0}^{N} \sum_{w=0}^{v} P(D=v-w)\, P(K=v)\, w$$

Finalement, nous obtenons :

$$E[V(s)] = E[K] - E[M(s)]$$

# Annexe B

# L'interface homme-machine proposé

L'interface pour la gestion de la redistribution des voitures sur les site comprend deux écrans principaux. Le premier représente la situation actuelle du système et le second est une estimation du futur pour aider l'exploitant du système innovant de voitures en libre service.

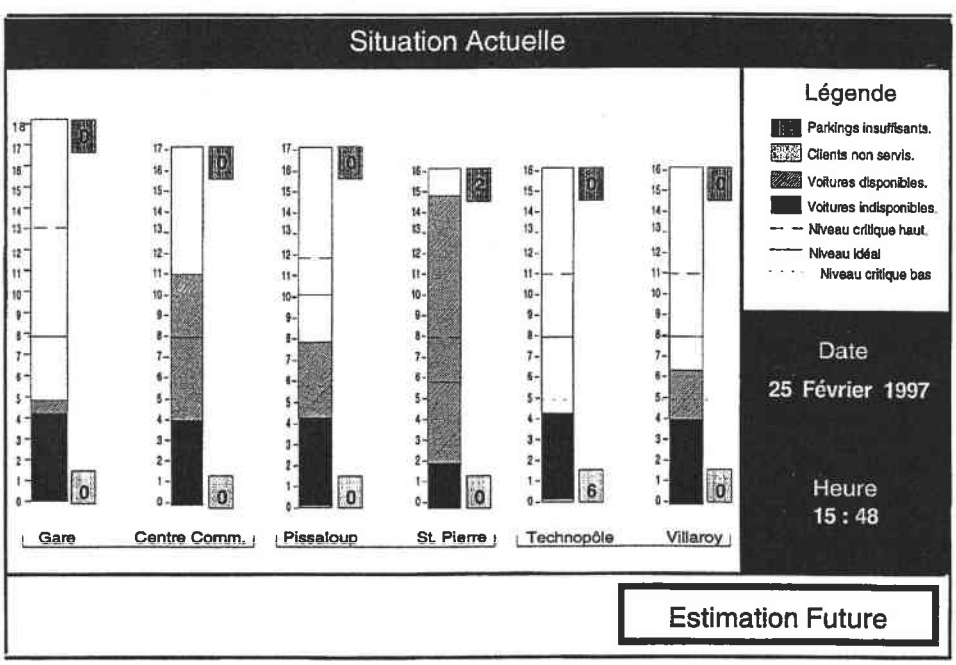



FIG. B.1: *Premier écran de l'interface de la redistribution*

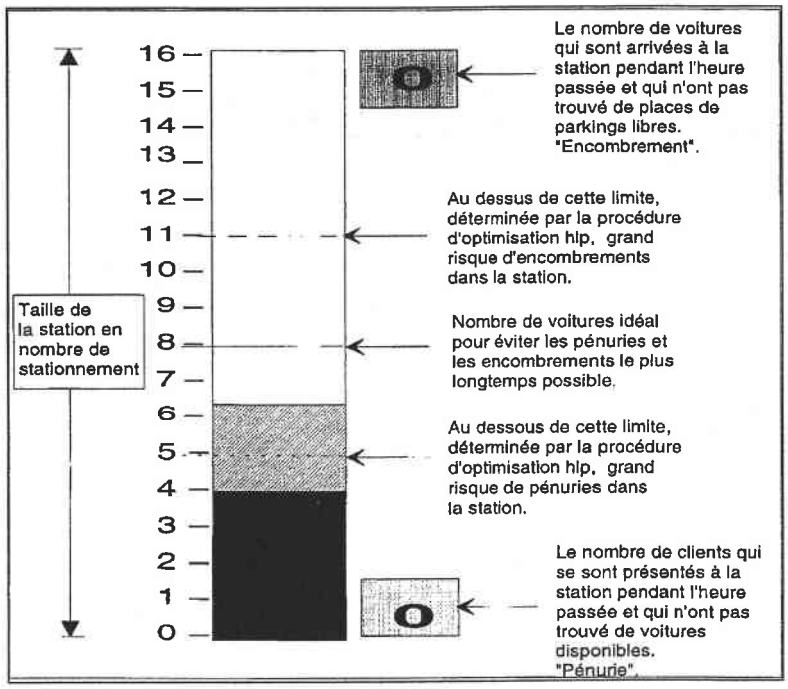



FIG. B.2: *Détails du premier écran de l'interface de la redistribution*

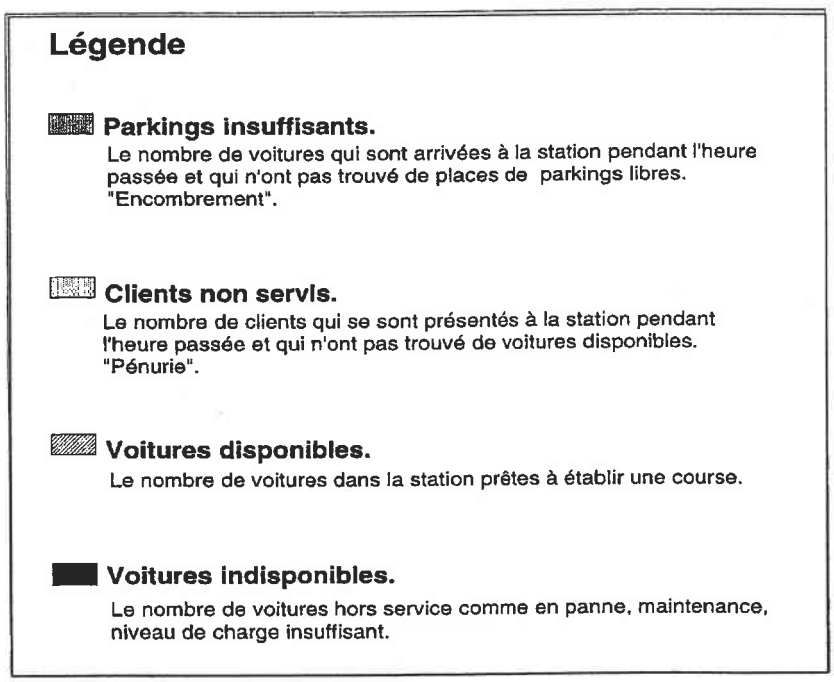



FIG. B.3: *La légende utilisée*

L'écran suivant est une estimation du la situation future dans les différents sites du système. Le côté droit de l'écran affiche les résultats obtenus par le problème linéaire détaillé dans le second chapitre de la troisième partie du mémoire. Le côté gauche est une aide à la gestion de la flotte. L'exploitant peut saisir un déplacement à effectuer et voir l'impact qu'il a sur le système dans les prochaines périodes.

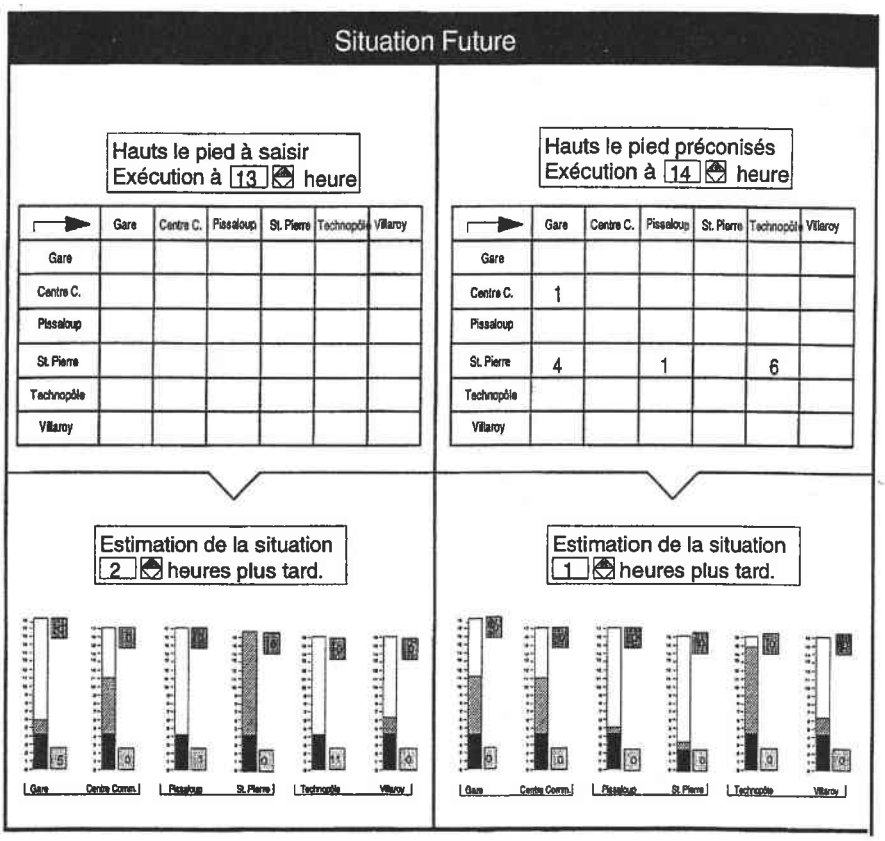



FIG. B.4: *Le second écran de l'interface*

la figure B.5 détaille une partie de la figure précédente.

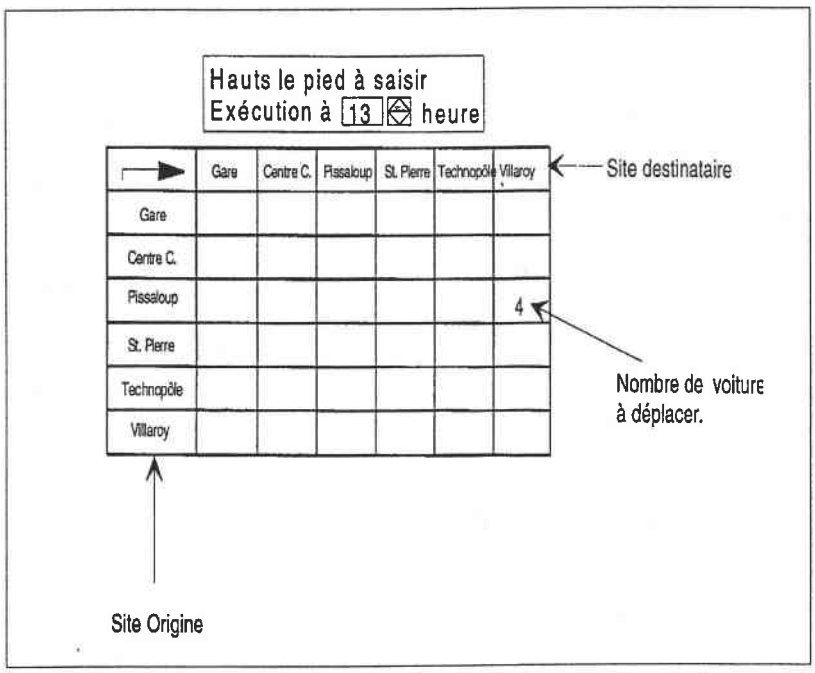



FIG. B.5: *Détail du second écran*

# Références

[14t97a] 14th International Electric Vehicle Symposium and Exposition. *Evaluation of an Electric Vehicle Demonstration Project*, Florida, December 1997.

[14t97b] 14th International Electric Vehicle Symposium and Exposition. *Where, When, How Fast and How Much? Questions about Consumer Demand for Home, Away From Home, Time of Day, and Speed of Recharging for Electric Vehicles*, Florida, December 1997.

[All98] Chafik Allal. *Commande d'un réseau de transport urbain*. PhD thesis, Université IX Dauphine, 1998.

[Ass88] A. Assad. *Vehicle Routing : Methods and Studies*, chapter Modeling and Implementation Issues in Vehicle Routing, pages 7–45. Elsevier Science Publishers, Amsterdam, 1988.

[Bak74] Kenneth R. Baker. *Introduction to Sequencing and Scheduling*. J. Wiley and sons, 1974.

[BG81] Lawrence Bodin and Bruce Golden. Classification in vehicle routing and scheduling. *Networks*, 11:97–108, 1981.

[BHM77] Stephen Bradley, Arnoldo Hax, and Thomas Magnanti. *Applied Mathematical Programming*. Addison-Wesley Publishing Company, Menlo Park, 1977.

[BN96] C. Bleijs and O. Normand. A fully automatic station using inductive charging techniques. Technical report, EDF, Electricité de France, Clamart, 1996.

[Bod90] Lawrence Bodin. Twenty years of routing and scheduling. *Operations Research*, 38(4):571–579, 1990.

[Can96] Olivier Canzler. Rapport interne praxitèle. Technical report, Renault, Trappes, 1996.

[CHP97] Fabrice Chauvet, Névine Hafez, and Jean-Marie Proth. Gestion d'un système de véhicules électriques en libre-service. In *Modélisation et simulation des systèmes de production et de logistique*, pages 325–332, Paris, Mai 1997. MOSIM'97, Hermes.

[CHPS97] Fabrice Chauvet, Névine Hafez, Jean-Marie Proth, and Nathalie Sauer. Management of a pool of self-service cars. *Journal of Intelligent Manufacturing*, pages 459–466, 1997.

[Dan66] George Dantzig. *Applications et prolongements de la programmation linéaire*. Dunod, Paris, 1966.

[Dav91] Lawrence Davis, editor. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.

[DDS92] Martin Desrochers, Jacques Desrosiers, and Marius Solomon. A new optimisation algorithm for the vehicle routing problem with time windows. *Operations Research*, 40:342–254, 1992.

[DFR98] Moshe Dror, Dominique Fortin, and Catherine Roucairol. Redistribution of self-service electric cars: A case of pickup and delivery. Technical Report 3543, INRIA, Metz, 1998.

[DLSS88] M Desrochers, J. Lenstra, M.W.P. Savelsbergh, and F. Soumis. *Vehicle Routing : Methods and Studies*, volume 16, chapter Vehicle Routing with Time Windows : Optimization, and Approximation, pages 65–85. Elsevier Science Publishers, Amsterdam, 1988.

[ea97] Emile Aarts et al. *Local Search in Combinatorial Optimization*, chapter Simulated Annealing, pages 91–120. John Wiley and Sons, Chichester, 1997.

[Ele96] Electricité dans les Transports Urbains. *Conduite en train pour véhicules électriques en libre-service*, Paris, avril 1996.

[Fou96] Fourth IEEE Mediterranean Symposium on New Directions in Control and Automationg. *Platooning technique for empty vehicles distribution in the PRAXITELE project*, Krete, June 1996.

[GLP94] Michel Gendrau, Gilbert Laporte, and Jean-Yves Potvin. Local search algorithms for the vehicle routing problem. Technical report, Centre de recherche sur les transports, Montréal, 1994.

[GLP97] Michel Gendreau, Gilbert Laporte, and Jean-Yves Potvin. *Local Search in Combinatorial Optimization*, chapter Vehicle Routing: Modern Heuristics, pages 311–336. John Wiley and Sons, Chichester, 1997.

[Haf97] Névine Hafez. Analyse fonctionnelle de l'optimisation de la recharge - rapport interne praxitèle. Technical report, INIRA, Rocquencourt, 1997.

[HDD90] Mohamed Haouari, Pierre Dejax, and Martin Desrochers. Les problèmes de tournées avec contraintes de fenêtres de temps, l'état de l'art. *RAIRO, Recherche Opérationnelle*, 24(3):217–244, 1990.

[HKS88] M. Haimovich, A. Rinnooy Kan, and L. Stougie. *Vehicle Routing : Methods and Studies*, volume 16, chapter Analysis of Heuristics for Vehicle Routing Problems, pages 47–61. Elsevier Science Publishers, Amsterdam, 1988.

[HTdW97] Alain Hertz, Eric Taillard, and Dominique de Werra. *Local Search in Combinatorial Optimization*, chapter Tabu Search, pages 121–136. John Wiley and Sons, Chichester, 1997.

[ISA96] ISATA Congress. *Praxitele : A New Public Transport with Self Service Electric Cars*, Florence, June 1996.

[Jod94] Jean-François Jodouin. *Les réseaux de neurones : principes et définitions.* Hermes, Paris, 1994.

[KS97] Gerard Kindervater and Martin Savelsbergh. *Local Search in Combinatorial Optimization*, chapter Vehicle Routing: Handling Edge Exchanges, pages 337–360. John Wiley and Sons, Chichester, 1997.

[La87] Eugene L. Lawler and al. *The Travelling Salesman Problem : a Guided Tour of Combinatorial Optimisation.* Series In Discrete Mathematics And Optimisation. J. Wiley and sons, 1987.

[Loo64] Paul Loomba. *Linear Programming : an Introductory Analysis.* Mc Graw-Hill Book Company, New York, 1964.

[Mau93] Thierry Mautor. *Contribution à la résolution des problèmes d'implantation algorithmes séquentiels et parallèles pour l'affectation quadratique.* PhD thesis, Université Pierre et Marie Curie, Paris, 1993.

[Mur76] Katta Gopalakrishna Murty. *Linear and Combinatorial Programming.* J. Wiley and sons, 1976.

[Nag86] Barinda Nag. *Vehicle Routing in the Presence of Site/Vehicle Dependency Constraints.* PhD thesis, University of Maryland, Maryland, 1986.

[PG96] Jean-Yves Potvin and François Guertin. *Meta-heuristics: theory and applications*, chapter The Clustered Traveling Salesman Problem: A Genetic Approach, pages 619–631. Kluwer Academic Publishers, Boston, 1996.

[PH90] Jean-Marie Proth and H. P. Hillion. *Mathematical Tools in Production Management, Competitive Methods in Operations Research and Data Analysis.* Plenum Press, New York, 1990.

[PS97] Carsten Peterson and Bo Soderberg. *Local Search in Combinatorial Optimization*, chapter Artificial Neural Networks, pages 173–214. John Wiley and Sons, Chichester, 1997.

[Psa88] Harilaos Psaraftis. *Vehicle Routing : Methods and Studies*, volume 16, chapter Dynamic Vehicle Routing Problems, pages 223–249. Elsevier Science Publishers, Amsterdam, 1988.

[Ros91] Sheldon Ross. *A course in simulation*. Maxwell Macmillan International Editions, New York, 1991.

[Rou87] Catherine Roucairol. *Du séquentiel au Parallèle : la recherche arborescente et son application à la programmation quadratique en variable 0.1*. PhD thesis, Université Pierre et Marie Curie, Paris, 1987.

[RR94] César Rego and Catherine Roucairol. Le problème de tournées de véhicules : étyde et résolution approchée. Technical Report 2197, INRIA, Rocquencourt, 1994.

[Sak84a] Michel Sakarovitch. *Graphes et programmation linéaire*. Optimisation Combinatoire, Méthodes mathématiques et algorithmiques, graphes et programmation linéaire. Hermann, Editeurs des sciences et des arts, Paris, 1984.

[Sak84b] Michel Sakarovitch. *Programmation discrète*. Optimisation Combinatoire, Méthodes mathématiques et algorithmiques, graphes et programmation linéaire. Hermann, Editeurs des sciences et des arts, Paris, 1984.

[Thi78] Robert Thierauf. *An Introductory Approach to Operations Research*. John Wiley and Sons, Santa Barbara, 1978.

[VDBKS93] L.J.J. Van Der Bruggen, J.K. Kenstra, and P.C. Schuur. Variable depth search for the single vehicle pickup and delivery problem with time windows. *Transportation Science*, 27:298–311, 1993.