Titre: Title:	Application de la méthode IPS au problème de localisation d'entrepôt sans capacité
Auteur: Author:	Bertrand Velut
Date:	2010
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Velut, B. (2010). Application de la méthode IPS au problème de localisation d'entrepôt sans capacité [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/411/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/411/
Directeurs de recherche: Advisors:	Guy Desaulniers, & François Soumis
Programme: Program:	Mathématiques appliquées

UNIVERSITÉ DE MONTRÉAL

APPLICATION DE LA MÉTHODE IPS AU PROBLÈME DE LOCALISATION D'ENTREPÔTS SANS CAPACITÉ

BERTRAND VELUT
DÉPARTEMENT DE MATHÉMATIQUES ET GÉNIE INDUSTRIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION DU DIPLÔME DE
MAÎTRISE ÈS SCIENCES APPLIQUÉES
(MATHÉMATIQUES ET GÉNIE INDUSTRIEL)
AOÛT 2010

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

APPLICATION DE LA MÉTHODE IPS AU PROBLÈME DE LOCALISATION D'ENTREPÔTS SANS CAPACITÉ

présenté par : M. VELUT Bertrand, B.Ing.
en vue de l'obtention du diplôme de : Maîtrise ès Sciences Appliquées
a été dûment accepté par le jury constitué de :

M. AUDET Charles, Ph.D., président.
M. DESAULNIERS Guy, Ph.D., membre et directeur de recherche.
M. SOUMIS François, Ph.D., membre et co-directeur de recherche.
M. HERTZ Alain, D.Sc., membre.

À ma famille et mes amis.

Remerciements

Mes remerciements vont tout d'abord à M. Guy Desaulniers, mon directeur de recherche pour son aide, sa patience et sa gentillesse tout au long de cette maîtrise. Ensuite, je souhaite remercier M. François Soumis, mon codirecteur, pour son accueil au GERAD et ses explications très enrichissantes. Leurs idées et la pertinence de leur propos m'ont beaucoup aidé dans mes recherches. Je tiens également à les remercier pour l'aide fiancière qu'ils m'ont apportée durant toute la durée du projet.

Je souhaite ensuite remercier M. Quentin Lequy et M. Benoît Rochefort pour leur aide en informatique. L'aboutissement de cette maîtrise aurait été plus difficile sans eux.

Je voudrais également remercier M. François Lessard pour sa disponibilité et pour avoir assuré toutes les modifications d'IPS (*Improved Primal Simplex*) nécessaires à l'aboutissement de ce travail. Ses réponses à mes questions m'ont également été d'une grande aide.

Je voudrais aussi remercier M$^{\text{lle}}$ Aida Reguigui qui m'a encouragé tout au long de ma maîtrise. Son amitié et sa joie de vivre ont fini par déteindre sur moi.

J'aimerais enfin exprimer ma gratitude à mes amis, au personnel du GERAD et du département de mathématiques et de génie industriel pour leur sympathie et leur soutien.

Résumé

Le problème de localisation d'entrepôts sans capacité (LESC) est un problème qui a été très étudié dans le domaine de la recherche opérationnelle depuis une cinquantaine d'années pour ses caractéristiques et ses applications dans le monde industriel. Il consiste à installer un certain nombre d'entrepôts sur un ensemble de sites et à affecter un et un seul entrepôt à chaque client d'un ensemble de clients. Sur chaque site, il coûte un certain montant pour y installer un entrepôt et pour chaque client, il coûte un autre montant pour l'affecter à un site sur lequel un entrepôt est installé. L'objectif est de minimiser le coût total. Ce problème NP-difficile est très compliqué à résoudre de manière exacte surtout pour des instances de grande taille. Nous proposons dans ce mémoire une méthode permettant de le résoudre à l'optimalité.

La méthode proposée est basée sur la combinaison d'une métaheuristique de type recherche tabou et d'un algorithme de résolution de programmes linéaires nommé IPS (*Improved Primal Simplex*) qui implémente une version améliorée du très connu algorithme du simplexe primal. Le tout fonctionne en faisant créer par la métaheuristique une solution proche de la solution optimale. Celle-ci sert de point de départ pour l'algorithme IPS qui termine la résolution jusqu'à l'optimalité. Cet algorithme IPS corrige une faiblesse de l'algorithme du simplexe primal en profitant de la dégénérescence quand celle-ci apparaît lors de la résolution d'un problème. Le problème LESC fait partie des problèmes à dégénérescence forte.

Dans ce mémoire, nous développons d'abord une métaheuristique de recherche tabou qui produit des résultats qui sont, dans certains cas, supérieurs à ceux obtenus dans la littérature. Par la suite, nous montrons que la méthode proposée est plus efficace pour résoudre la relaxation linéaire du problème LESC que le traditionnel algorithme du simplexe primal sur les instances dont la matrice des coûts d'affectation a une densité comprise entre 0,1 et 0,6. Nous montrons également que cette efficacité augmente avec la taille des instances. Nos résultats mettent en évidence des temps de calcul réduits d'un facteur pouvant aller jusqu'à 3,5 pour certaines relaxations linéaires par rapport à l'algorithme du simplexe primal.

Enfin, pour la résolution en nombres entiers, nous testons l'intégration de cette méthode dans un algorithme d'évaluation et séparation utilisant deux stratégies dif-

férentes : la stratégie profondeur d'abord et la stratégie meilleur d'abord. Nos tests sur cette intégration dans l'algorithme d'évaluation et séparation font ressortir des faiblesses sur l'implémentation, ce qui empêche la méthode proposée de conserver sa supériorité sur un algorithme d'évaluation et séparation identique utilisant l'algorithme du simplexe primal.

Nous terminons ce mémoire en donnant des pistes d'amélioration pour la méthode proposée, ainsi que pour son application à la résolution du problème LESC.

Abstract

In the last five decades, the uncapacitated facility location problem (UFLP) has been widely studied in operations research because of its features and its many applications in industrial contexts. Given a set of customers to service and a set of potential sites on which a warehouse can be built, the UFLP consists of selecting on which sites to install a warehouse and assigning one and only one warehouse to each customer. A fixed cost for building a warehouse is associated with each site and customer/warehouse assignment costs are also considered. The aim of the problem is to minimize the total cost. This NP-hard problem is very hard to solve to optimality, especially when the instance size is large. In this master's thesis, we propose a method to solve it to optimality.

The proposed method is based on a combination of a tabu search heuristic and a linear programming solution algorithm called Improved Primal Simplex (IPS). It starts by computing a solution close to optimality using tabu search. This solution is then used to initialize IPS that solves the problem to optimality. The IPS algorithm overcomes a weakness of the simplex primal algorithm by taking advantage of degeneracy when it appears during the solution process. Typically, the UFLP yields high degeneracy.

In this thesis, we first develop an efficient tabu search method that produces better results in some cases than the algorithms from the literature. Second, we show that the proposed method solves the linear relaxation of UFLP more efficiently than the classic primal simplex algorithm for instances where the assignment cost matrix has a density between 0.1 and 0.6. This efficiency increases with the instance size. We report a reduction of the computational time of up to a factor of 3.5 when the IPS algorithm is compared with the primal simplex algorithm.

Then, for the integer solution process, we test an integration of the IPS method into a branch-and-bound algorithm using two search strategies: depth-first and best-first. Our tests highlight a weakness of this approach that forbids our method to keep its superiority over the same branch-and-bound algorithm based on the primal simplex algorithm.

We conclude this thesis by briefly discussing future research directions for the

proposed method and also for its application to solve the UFLP.

Table des matières

Liste des tableaux

Liste des algorithmes

Liste des figures

Chapitre 1

INTRODUCTION

Depuis une cinquantaine d'années, de nombreuses avancées dans le domaine de la recherche opérationnelle ont mené à des progrès significatifs en optimisation. Ces avancées consistent pour une grande part en la résolution de problèmes à combinatoire particulièrement importante, pas toujours supportable pour les ordinateurs actuels. Les problèmes de localisation font partie de cette catégorie de problèmes à combinatoire lourde. Nous allons nous intéresser dans ce mémoire à l'un de ces problèmes qui a très rapidement suscité l'intérêt de nombreux scientifiques pour ses applications industrielles et sa difficulté : le problème de Localisation d'Entrepôts Sans Capacité (LESC). En anglais, ce problème est appelé *uncapacitated facility location problem* ou *simple plant location problem*.

Ce problème de localisation d'entrepôts est un problème d'optimisation à variables entières qui peut être résolu, entre autres, par un algorithme d'évaluation et de séparation (ou *branch-and-bound* en anglais) qui résout des relaxations linéaires à chaque nœud d'un arbre de branchement. Ces relaxations linéaires sont résolues par des algorithmes de programmation linéaire tels que l'algorithme du simplexe primal. Cependant, la structure du problème de localisation d'entrepôts sans capacité est telle qu'elle induit une dégénérescence importante qui favorise l'apparition de pivots de simplexe pour lesquels la valeur de la fonction objectif ne décroît pas. Ces pivots dits dégénérés sont des opérations qui ne font pas progresser l'algorithme dans sa recherche de l'optimalité et sont donc source d'inefficacité.

Récemment, une méthode nommée"méthode primale du simplexe améliorée", plus connue sous son acronyme anglais IPS (*Improved Primal Simplex*), qui permet de contourner ce problème a été développée (l'acronyme anglais IPS sera utilisé pour désigner cette méthode dans la suite du mémoire).

Dans le cadre de ce mémoire, notre objectif est d'explorer la méthode IPS en l'appliquant au problème LESC et en comparant ses performances à l'algorithme du simplexe primal.

1.1 Éléments de la problématique

Le problème LESC introduit par Balinski (1965) représente un des problèmes de base d'une grande famille de problèmes de localisation. Il est d'ailleurs devenu très connu dans le domaine de l'optimisation et a aidé à contruire d'autres modèles de localisation plus détaillés d'après ReVelle et Eiselt (2005) et Melo *et al.* (2009). Le fait que de nombreux problèmes industriels actuels, notamment dans le domaine de la logistique, en dérivent directement lui confère une importance particulière. On peut penser que chaque progrès qui peut être fait pour sa résolution se répercutera logiquement sur ses problèmes fils.

En quelques mots, ce problème consiste à installer un certain nombre d'entrepôts sur un ensemble de sites dans le but de servir un ensemble de clients. Sur chaque site, il coûte un certain montant pour y installer un entrepôt, et pour chaque client, il coûte un autre montant pour l'affecter à un site sur lequel un entrepôt est installé. Le but est d'installer des entrepôts sur quelques uns des sites de l'ensemble des sites (au plus un entrepôt par site) et d'affecter chaque client à un et un seul entrepôt, tout cela en minimisant le coût total (coût d'installation des entrepôts sur les sites choisis et coût d'affectation des clients à leur entrepôt respectif). Le nombre d'entrepôts à installer et leur site d'installation ne sont pas déterminés à l'avance : ils dépendent de ce qui est le plus profitable pour une instance de problème donnée.

Il est évident que les termes "entrepôt" et "client" sont arbitraires : il peut s'agir d'objet d'autres types, du moment que le problème garde la même structure. Par exemple, selon le secteur d'activité, les entrepôts peuvent aussi devenir des usines ou des hôpitaux et les clients, des régions ou des patients d'hôpitaux. Les "coûts d'affectation" peuvent aussi représenter des grandeurs différentes comme des distances par exemple.

Bien que ce problème semble simple à expliquer, sa résolution reste difficile. Un certain nombre de travaux ont été faits permettant de le résoudre. Cependant, encore aujourd'hui avec les algorithmes actuels, il reste très difficile de le résoudre de manière exacte (c'est-à-dire de trouver la solution qui engendre le coût minimum), notamment lorsque les instances sont de grande taille.

D'autres problèmes proches du problème LESC sont également connus dans la littérature comme le problème de localisation d'entrepôts avec capacité qui fait intervenir des demandes pour les clients et des capacités pour les entrepôts. Cela mo-

difie certaines contraintes et en fait apparaître d'autres comme par exemple celles qui font qu'un ensemble de clients peut être affecté à un même entrepôt que si la somme de leurs demandes est inférieure ou égale à la capacité de l'entrepôt considéré. Ces contraintes supplémentaires compliquent cependant grandement la résolution. Le problème du p-médian est également connu et très proche du problème LESC : il en diffère par son absence de coûts d'installation et sa majoration du nombre d'entrepôts qu'il est possible d'installer. Ce problème revient en fait à trouver les p sites sur lesquels doivent être installés les entrepôts pour minimiser les coûts d'affectation des clients aux entrepôts. Enfin, le problème du k-centre diffère du problème LESC car il faut trouver la répartition de k entrepôts qui minimise le coût d'affectation maximum payé entre un client et un entrepôt (chaque client étant relié à l'entrepôt qui minimise son coût d'affectation). Dans le cadre de ce mémoire, nous ne considérerons que le problème LESC.

Nous tiendrons compte des cas où l'ensemble des clients n'est pas connexe à l'ensemble des sites, c'est-à dire que des clients peuvent ne pas avoir le droit d'être affectés à certains sites même si un entrepôt y est installé. Ces interdictions seront plus ou moins fréquentes selon l'instance de problème considérée.

Comme nous l'avons vu précédemment, le problème LESC peut être résolu, entre autres, par un algorithme d'évaluation et séparation. La relaxation linéaire à chaque nœud de l'arbre de branchement peut être résolue de différentes façons : l'algorithme du simplexe primal peut être utilisé ou l'algorithme simplexe dual ou encore un algorithme de points intérieurs. Il est bien connu qu'en présence de dégénérescence, les deux derniers algorithmes sont plus performants que le premier. Or, l'introduction récente de la méthode IPS par Elhallaoui *et al.* (2007) qui améliore l'algorithme du simplexe primal donne une alternative intéressante qui pourrait supplanter les algorithmes du simplexe dual et des points intérieurs. Nous voulons explorer cet aspect en plus de comparer IPS à l'algorithme du simplexe primal qui est l'objectif principal de ce mémoire. Une des caractérisques d'IPS est qu'en plus de contourner les pertes d'efficacité liées aux pivots dégénérés, il est censé avoir une performance accrue quand on lui donne en entrée une solution initiale du problème de bonne qualité. L'idée est donc de tenter de résoudre le problème LESC par un algorithme d'évaluation et séparation utilisant l'algorithme IPS pour résoudre les relaxations linéaires de l'arbre de branchement et de produire une solution initiale de bonne qualité en amont de cette résolution afin d'exploiter les forces d'IPS. Pour obtenir cette solution initiale,

nous utiliserons une métaheuristique de type recherche tabou ; cela devrait ainsi aider IPS à terminer plus rapidement la résolution jusqu'à l'optimalité.

Pour IPS, il a déjà été vu dans le passé que la résolution de la relaxation linéaire fonctionnait bien lorsque certaines conditions sur la dégénerescence et sur la densité de la matrice des contraintes étaient vérifiées. Cependant, rien n'a encore été testé sur la résolution en nombres entiers. Nous avons aussi pour objectif dans ce mémoire d'observer le comportement d'IPS pour la résolution en nombres entiers.

Il est à noter que les mécanismes intervenants dans l'algorithme d'évaluation et séparation sont déjà intégrés dans le solveur Cplex et ont été intégrés récemment dans IPS. Nous les utiliserons donc tels quels. Aucune modification de notre part n'a été apportée à IPS : nous nous sommes placés d'un point de vue utilisateur tout au long de cette recherche.

1.2 Objectifs de recherche

Les objectifs de cette recherche sont multiples. En effet, le premier est tout d'abord de résoudre le problème LESC d'une manière différente de ce qui a été fait jusqu'à présent en utilisant la méthode IPS, et d'évaluer les performances de la méthode proposée. Étant donné l'importance du problème, il est possible que si la résolution fonctionne bien, des espoirs naîssent sur la résolution de problèmes de même type que l'on retrouve régulièrement dans le domaine de la logistique.

Le deuxième objectif découle du premier : nous voulons exploiter au maximum la méthode IPS en lui fournissant une solution initiale de qualité. Pour cela, une métaheuristique de recherche tabou est développée pour le problème LESC dans laquelle nous avons pris soin d'apporter différentes stratégies d'accélération afin d'éviter d'absorber une trop grande partie du temps de calcul par rapport à IPS.

Enfin, le dernier objectif est d'analyser dans ce contexte les performances d'IPS qui est encore au stade expérimental, notamment en ce qui concerne la résolution en nombres entiers qui n'a encore jamais été testée. Pour cela, l'algorithme du simplexe primal et IPS sont comparés. Le solveur Cplex qui est actuellement la référence en termes de solveur de programmation linéaire en nombres entiers et auquel on impose d'utiliser l'algorithme du simplexe primal est utilisé pour les tests. Cela permettra d'avoir un regard plus pertinent sur IPS et ses performances qui sont encore mal connues. Cette comparaison est d'abord faite au niveau de la résolution de la re-

laxation linéaire du problème LESC, où IPS est comparé à l'algorithme du simplexe primal mais aussi, à titre d'information, à ceux du simplexe dual et barrière. La comparaison est ensuite faite au niveau de la résolution en nombres entiers, où seuls IPS et l'algorithme du simplexe primal tous les deux encastrés dans un algorithme d'évaluation et séparation sont comparés.

1.3 Plan du mémoire

Dans ce premier chapitre, nous avons survolé ce qui est abordé dans ce mémoire. Dans le deuxième chapitre, nous expliquons dans un premier temps le problème LESC en détail et présentons son modèle de base. Dans un deuxième temps, nous faisons une revue de la littérature existante sur le problème LESC depuis sa formulation dans les années 1960. Cela permet de dessiner le contexte du mémoire.

Au troisième chapitre, nous exposons la première étape de notre méthode de résolution. Nous commençons d'abord par expliquer la métaheuristique qui permet d'obtenir une solution initiale de bonne qualité (l'algorithme glouton qui crée une solution préliminaire et la recherche tabou appliquée en aval qui l'améliore) pour ensuite exposer et analyser les résultats obtenus.

Dans le quatrième chapitre, nous détaillons la deuxième et dernière étape de la méthode que nous proposons. Nous donnons d'abord un aperçu de l'algorithme afin d'en comprendre le fonctionnement pour ensuite exposer les tests numériques effectués. Ces tests présentent des résultats sur la relaxation linéaire ainsi que d'autres sur l'algorithme d'évaluation et séparation utilisant IPS pour la résolution de la relaxation en nombres entiers. Une analyse de ces résultats est faite en même temps, où il est d'ailleurs mis en évidence que pour des problèmes de densité comprises entre 0,1 et 0,6, la méthode tabou-IPS est préférable à la méthode du simplexe primal. Ce chapitre fait office d'analyse de l'ensemble du mémoire.

Nous terminons par une conclusion qui résume l'ensemble du travail effectué et qui retient les principales avancées qui ont été mises en évidence. Nous ouvrons ensuite des pistes de recherche futures qui découlent logiquement de notre travail.

Chapitre 2

DEFINITION DU PROBLÈME ET REVUE DE LITTÉRATURE

Avant de présenter le modèle classique du problème LESC et une revue de littérature sur ce problème, nous commencons par préciser certaines terminologies utilisées dans ce mémoire.

Nous considérons par abus de langage qu'une instance de taille $n \times m$ est un problème ou une instance ayant un ensemble de site de cardinalité n et un ensemble de clients de cardinalité m. L'expression "affecter un client à l'entrepôt i" est aussi utilisée pour dire "affecter un client au site i sur lequel un entrepôt est installé", et l'expression "ouvrir ou fermer le site i" signifie la même chose que "installer ou désinstaller un entrepôt sur le site i". Enfin, "affecter un client à l'entrepôt le plus proche" veut dire "affecter un client au site sur lequel un entrepôt est installé et qui possède le coût d'affectation minimum".

2.1 Modèle

Soit n le nombre de sites considérés dans le problème ;
m le nombre de clients ;
E l'ensemble des n sites ;
J l'ensemble des m clients ;
f_i le coût d'installation d'un entrepôt sur le site i ;
c_{ij} le coût d'affectation du client j à l'entrepôt i ;

Pour modéliser ce problème, deux types de variables sont utilisés :

x_{ij} : variable non-négative (indirectement binaire) qui est égale à 1 si le client $j \in J$ est affecté à l'entrepôt $i \in E$ et à 0 si ce n'est pas le cas.

y_i : variable binaire qui est égale à 1 si un entrepôt est installé sur le site $i \in E$ et à 0 si ce n'est pas le cas.

Le problème LESC se modélise couramment sous la forme du programme linéaire en nombres entiers suivant :

$$\min_{x,y} \quad \sum_{j\in J}\sum_{i\in E} c_{ij}x_{ij} + \sum_{i\in E} f_i y_i \tag{2.1}$$

$$\text{sujet à} \quad \sum_{i\in E} x_{ij} = 1 \qquad \forall\ j \in J, \tag{2.2}$$

$$x_{ij} \leq y_i \qquad \forall\ j \in J,\ \forall\ i \in E, \tag{2.3}$$

$$y_i \in \{0,1\} \qquad \forall\ i \in E, \tag{2.4}$$

$$x_{ij} \geq 0 \qquad \forall\ j \in J,\ \forall\ i \in E. \tag{2.5}$$

La fonction objectif (2.1) consiste à minimiser les coûts totaux. Elle est composée de deux termes : un premier terme en x_{ij} qui comptabilise les coûts d'affectation des clients à leur entrepôt respectif et un deuxième terme en y_i qui comptabilise les coûts d'installation des entrepôts sur les différents sites choisis.
Les contraintes (2.2) imposent à chaque client d'être affecté à un et un seul site.
Les contraintes (2.3) obligent à ce que chaque client j ne puisse être servi par le site i uniquement si un entrepôt y est installé (c'est-à-dire si $y_i = 1$).
Les contraintes (2.4) sont les contraintes d'intégrité sur les variables y_i. Elles forcent à ce que les y_i soient binaires.
Enfin, les contraintes (2.5) sont les contraintes de non-négativité sur les variables x_{ij}.

Remarque 1 : Concernant l'intégrité implicite des x_{ij}, si $y_i \in \{0,1\}$, alors une solution optimale s'obtient en affectant chaque client au site i le moins cher parmi les sites sur lesquels un entrepôt est installé. Par conséquent $x_{ij} \in \{0,1\}$ implicitement.

Remarque 2 : Dans certaines variantes du problème, certains clients peuvent être restreints à certains sites. On supposera que $c_{ij} = +\infty$ pour les affectations interdites, bien que dans une implantation dans un solveur, les variables correspondantes ne sont pas créées.

Ainsi, nous pouvons définir la "matrice d'affectation" comme étant la matrice des coûts d'affectation (c_{ij}), et sa densité δ comme étant la proportion de coûts d'affectation c_{ij} non-infinis par rapport au nombre de c_{ij} total. On a donc :

$$\delta \quad = \quad \frac{r}{n\ m} \quad \text{où } r \text{ est le nombre de } c_{ij} \text{ non-infini dans la matrice d'affectation.}$$

Cette formulation, appelée “formulation forte”, est la plus couramment utilisée dans la littérature.

Un peu après l'apparition de cette formulation qui est la première formulation du problème et qui a été introduite par Balinski (1965), les scientifiques Efroymson et Ray (1966) ont essayé de modifier les contraintes (2.3) afin d'accélérer l'algorithme d'évaluation et séparation pour la résolution exacte. Cette formulation a été appelée la “formulation faible” du problème. Elle possède un nombre de contraintes bien inférieur à celui de la formulation forte ce qui est censé accélérer l'évaluation de chaque noeud de l'arbre de branchement et donc d'en accélérer le parcours. Dans cette formulation faible, les contraintes (2.3) sont remplacées par les contraintes :

$$0 \leq \sum_{j \in J_i} x_{ij} \leq n_i y_i, \ \forall i \in E \tag{2.6}$$

où J_i représente l'ensemble des clients qui peuvent être affecté au site i,
et n_i est le nombre d'éléments dans J_i.

Cependant, le principal inconvénient de cette formulation est que le nombre de clients affectés au site i est souvent petit par rapport à n_i, ce qui fait que des sites ont tendance à absorber seulement une petite partie de la charge fixe dans la solution du programme linéaire. Ce nombre n_i joue finalement le rôle d'un grand M. Il en résulte un saut d'intégrité plus grand que pour la formulation forte ce qui engendre ainsi un plus grand arbre de branchement qui oblige à résoudre davantage de relaxations linéaires. Cela fait perdre du temps. Enfin, le dual de la formulation forte a une structure très simple que n'a pas le dual de la formulation faible et qui est grandement exploitée dans les algorithmes de résolution exacte les plus efficaces . Cette formulation faible n'a finalement pas été réutilisée depuis 1966 dans la littérature. Dans la suite du mémoire, nous considérons donc la formulation forte.

2.2 Caractéristiques du problème

Comme mentionné précédemment, la résolution de ce problème consiste à trouver la combinaison de sites à ouvrir et de clients à affecter à ces sites qui engendre le coût total minimum. Une solution peut donc être modélisée par un vecteur de taille $n(m+1)$, qui aurait pour ses n premières composantes les variables y_i d'installation

et pour ses $n \times m$ composantes suivantes les variables x_{ij} d'affectation.

Le nombre de solutions possibles pour notre problème est donc $2^{n(m+1)}$. Cependant, comme les entrepôts n'ont pas de capacité maximale, pour une combinaison de sites ouverts donnée, chaque client est connecté à l'entrepôt le plus proche. Par conséquent, les variables x_{ij} modélisant les affectations des clients aux sites découlent directement des sites choisis pour l'installation des entrepôts. Le nombre de solutions potentiellement intéressantes peut donc finalement être ramené à 2^n.

D'autre part, Cornuejols *et al.* (1990) ont démontré que le problème LESC est NP-difficile ce qui explique entre autres pourquoi il n'existe pas aujourd'hui d'algorithme glouton permettant de le résoudre.

Lorsque le modèle (2.1)-(2.5) est résolu par une méthode d'évaluation et séparation employant l'algorithme du simplexe primal, il y a de fortes chances d'effectuer de nombreux pivots dégénérés en résolvant les relaxations linéaires. En effet, chaque variable y_i prenant la valeur 0 dans une solution de base entraîne une variable de base nulle par contrainte (2.3) associée à i. Lorsque les coûts d'installation sont relativement élevés par rapport aux coûts d'affectation, les solutions de base rencontrées en cours de résolution contiennent beaucoup de variables y_i égales à 0 entraînant beaucoup de dégénérescence. L'algorithme IPS pourrait alors s'avérer beaucoup plus efficace que l'algorithme du simplexe primal.

D'un autre côté, la structure de coût d'une instance peut favoriser l'apparition de valeurs fractionnaires dans la solution de la relaxation linéaire ce qui engendre moins de dégénérescence. Pour illustrer cela, considérons une instance du problème LESC avec 3 sites et 3 clients dont les coûts associés sont représentés dans le tableau 2.1 et dont le graphe associé est représenté dans la figure 2.1 (les arcs de coût infini sont omis).

Site (i)	*Coûts d'installation* (f_i)	*Matrice d'affectation* (c_{ij})		
Clients :		1	2	3
1	80	30	40	∞
2	100	∞	40	30
3	60	30	∞	50

TABLEAU 2.1 Coûts associés à l'exemple

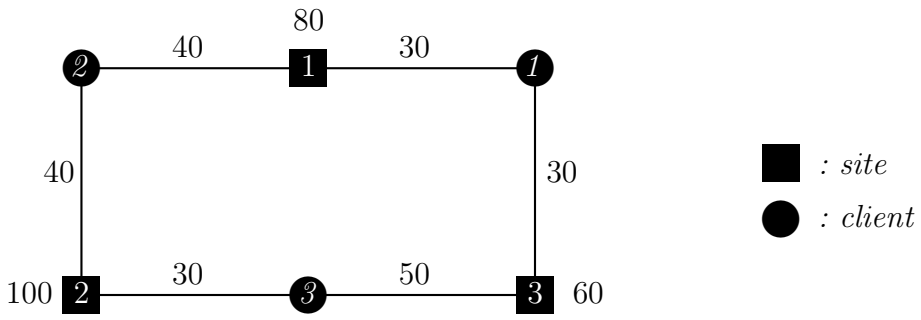


FIGURE 2.1 Graphe associés à l'exemple

Pour l'exemple, la solution optimale de la relaxation linéaire est :

$y_1 = 0,5$	$x_{11} = 0,5$	$x_{12} = 0,5$	$x_{13} = 0$
$y_2 = 0,5$	$x_{21} = 0$	$x_{22} = 0,5$	$x_{23} = 0,5$
$y_3 = 0,5$	$x_{31} = 0,5$	$x_{32} = 0$	$x_{33} = 0,5$

Sa valeur est de 230 dont 110 et 120 pour les coûts d'affectation et les coûts d'installation respectivement.

Toute réduction des coûts d'installation n'est possible qu'en augmentant la valeur d'une variable x_{ij} ayant un coût infini, ce qui impliquerait une solution de coût total plus élevée que 230. En effet, la seule manière de respecter les contraintes 2.2 ($\forall\ j \in J,\ \sum_{i \in E} x_{ij} = 1$) sans utiliser les coûts infinis est de respecter les inégalités suivantes : $y_1 + y_2 \geq 1$, $y_1 + y_3 \geq 1$ et $y_2 + y_3 \geq 1$. Donc, pour diminuer les coûts d'installation sans utiliser les coûts infinis, il faut trouver de meilleures valeurs pour les variables y_i que les valeurs ($y_1 = 0,5\ ;\ y_2 = 0,5\ ;\ y_3 = 0,5$) tout en respectant les inégalités précédentes, ce qui n'est pas possible avec ces coûts d'installation. Par conséquent, le seul moyen restant pour diminuer le coût total est d'accepter de dégrader les coûts d'installation pour espérer obtenir une économie supérieure dans les coûts d'affectation. Cependant, une telle situation n'existe pas : en effet, le seul moyen d'avoir une économie dans les coûts d'affectation est d'augmenter x_{23} tout en diminuant x_{33} (pour les autres clients ce transfert ne donne rien puisque leurs coûts d'affectation, hors coûts d'affectation infinis, sont égaux). Si l'on fait cela, en augmentant x_{23} de $\lambda > 0$ et en diminuant x_{33} de λ par exemple, on obtient une économie de $(50 - 30)\lambda = 20\lambda$; cependant, pour continuer à respecter les contraintes 2.2 pour le client 3, il faut augmenter y_2 de λ,

ce qui implique une augmentation de 100λ. On obtient donc augmentation du coût total de $(100 - 20)\lambda = 80\lambda$.

Pour la résolution en nombres entiers, la solution entière optimale est de coût 260, ce qui est supérieur à la solution précédente. Il est à noter que, pour cet exemple, il existe trois solutions optimales entières de coût 260 :

$$\begin{array}{llll} y_1 = 0 & x_{11} = 0 & x_{12} = 0 & x_{13} = 0 \\ y_2 = 1 & x_{21} = 0 & x_{22} = 1 & x_{23} = 1 \\ y_3 = 1 & x_{31} = 1 & x_{32} = 0 & x_{33} = 0 \ , \end{array}$$

$$\begin{array}{llll} y_1 = 1 & x_{11} = 1 & x_{12} = 1 & x_{13} = 0 \\ y_2 = 0 & x_{21} = 0 & x_{22} = 0 & x_{23} = 0 \\ y_3 = 1 & x_{31} = 0 & x_{32} = 0 & x_{33} = 1 \ , \end{array}$$

$$\begin{array}{llll} y_1 = 1 & x_{11} = 0 & x_{12} = 1 & x_{13} = 0 \\ y_2 = 0 & x_{21} = 0 & x_{22} = 0 & x_{23} = 0 \\ y_3 = 1 & x_{31} = 1 & x_{32} = 0 & x_{33} = 1 \ . \end{array}$$

Cet exemple exhibe donc un saut d'intégrité de près de 12%. La solution de la relaxation linéaire est toutefois non dégénérée si on exclut les variables associées aux indices i et j tels que $c_{ij} = \infty$. Cet exemple montre bien que lors de la résolution du problème LESC, on peut s'attendre à des sauts d'intégrité élevés, à de la dégénérescence ou à ces deux difficultés simultanément, ce qui rend la résolution d'autant plus difficile.

2.3 Revue de littérature

2.3.1 Historique

Le problème LESC est un problème assez ancien puisqu'il a été formulé dans les années 1960 par Balinski (1965). D'autres formulations ont ensuite été proposées dont celle de Efroymson et Ray (1966) qui a été vue dans la section 2.1. Ce problème possède certaines propriétés qui ont permis à des scientifiques d'élaborer des algorithmes relativement efficaces pour le résoudre.

L'ensemble des algorithmes appliqués à ce problème peut être divisé en deux catégories : les algorithmes exacts qui garantissent l'optimalité et les heuristiques

qui sont généralement plus rapides mais qui n'offrent aucune garantie d'optimalité. Les premières méthodes efficaces connues sont des méthodes exactes avec la méthode DualLoc proposée par Erlenkotter (1978) qui est encore d'actualité, la méthode PD-Loc de Korkel (1989) et BBDual de Janacek et Kovacikova (1997) pour ne citer qu'elles.

Les heuristiques quant à elles se sont fortement développées après les années 1990 avec l'avènement de métaheuristiques efficaces : la méthode de recuit simulé de Alves et Almeida (1992), les algorithmes génétiques par Kratica *et al.* (2001) et la recherche Tabou par Sultan et Fawzan (1999), Ghosh (2003), Michel et Hentenryck (2004), et Sun (2006). Des combinaisons de plusieurs métaheuristiques ont également été utilisées comme celle de Resende et Werneck (2006) et celle de Harm et Hentenryck (2005). Ces heuristiques ont l'avantage d'aboutir à des résultats de très bonne qualité bien plus rapidement que les algorithmes exacts, en particulier quand les instances ne sont pas euclidiennes d'après Galvao (2004), c'est-à-dire que les coûts d'affectation sont générés aléatoirement. Plus de détails sur ces méthodes de résolution seront donnés dans la suite de la revue de littérature.

Aujourd'hui, en plus des problèmes classiques du p-médian, du k-centre et de localisation d'entrepôts avec capacité, d'autres problèmes de localisation ressemblant au problème LESC sont à l'étude. C'est le cas des problèmes mentionnés par ReVelle et Eiselt (2005) : le "cent-dians" qui consiste à concilier à la fois les problèmes du k-centre et du p-médian en minimisant en même temps le coût d'affectation maximum et la somme des coûts d'affectation des clients aux entrepôts, ainsi que le "medi-centers" qui vise à minimiser la moyenne des coûts d'affectation sous contrainte que chaque coût d'affection n'excède pas une limite donnée. Le problème de localisation d'usines non désirables dont le but est de maximiser la somme des coûts d'affectation tout en maximisant le coût d'affectation maximum en fait également partie. Souvent, les coûts d'affectation sont assimilés à des distances dans ces problèmes. Le problème de localisation d'entrepôts sans capacité dynamique formulé par Warszawski (1973), Roodman et Schwarz (1975), et Van Roy et Erlenkotter (1982) fait aussi partie de cette famille. Il consiste à installer des entrepôts en vue de servir des clients comme dans le problème LESC mais cette fois-ci en faisant varier les coûts selon des périodes. Pour une partie de ces problèmes, les algorithmes d'optimisation utilisés sont proches et reprennent souvent les mêmes idées.

2.3.2 Applications industrielles

Le problème LESC est un problème qui possède de nombreuses applications dans le monde industriel et en particulier dans le domaine de la logistique où les problèmes de localisation font partie intégrante des nouvelles problématiques dans le contexte actuel de mondialisation. Ce problème est souvent incrusté dans des problèmes de localisation qui orientent les décisions pour les implantations d'infrastructures, comme par exemple l'installation d'entrepôts à des points stratégiques, ou encore l'installation de maternités pour améliorer l'efficacité de couverture des infrastructures médicales d'une municipalité comme le mentionne Galvao (2004). Il peut aussi s'avérer intéressant pour l'implantation d'écoles dans une région en développement, ou améliorer l'efficacité de chaînes d'approvisionnement d'entreprises. Ce sont ces applications, aussi nombreuses et diversifiées soient-elles, qui font toute la richesse de ce problème.

Ces enjeux se ressentent aussi bien dans la communauté scientifique que dans l'industrie si l'on en croît l'étude menée par Melo *et al.* (2009) où il est mentionné que le nombre de publications par année dans le domaine croît depuis une dizaine d'années. L'étude fait d'ailleurs état d'une recherche particulièrement active dans le domaine du management de la chaîne d'approvisionnement et des problèmes de localisation d'usines ou d'entrepôts, mentionnant qu'environ 120 articles ont été publiés entre 1998 et 2007 sur le sujet dont 56 entre 2004 et 2007. Pour la majorité d'entre eux, ils s'intéressent à des problèmes de localisation-allocation dans le domaine des stocks, de la capacité, de la production, du routage et de l'approvisionnement. Ces articles sont rattachés à des domaines aussi variés que l'alimentation, l'automobile, la chimie, l'informatique et les industries forestière et militaire. Pour une partie d'entre eux, ces problèmes ont un lien direct avec le problème LESC, ce qui explique encore une fois sa notoriété et l'importance de son étude.

Il est ensuite important de comprendre que ce problème LESC fait partie des grands classiques de la recherche opérationnelle dans le domaine des problèmes de localisation. ReVelle et Eiselt (2005) ont étudié les domaines de l'analyse de localisation et ont mis en avant les différentes classes de problèmes actuellement étudiées. Outre les problèmes tels que le p-médian ou le k-centre, qui sont étroitement liés au problème LESC, un certain nombre de variantes provenant pour la plupart d'associations de problèmes fondamentaux de localisation sont en passe d'être étudiés. C'est le cas par exemple des problèmes vus dans la fin de la section précédente.

Bien qu'il ait été reproché au problème LESC de ne pas être suffisamment réaliste

par Melo *et al.* (2009), il reste une problématique de base qui a inspiré beaucoup de variantes représentant des problèmes réels du monde industriel. Il est donc intéressant de se pencher dessus car si des progrès sont faits sur sa résolution il est possible qu'ils se répercutent sur l'ensemble des problèmes qui en découlent.

Nous allons maintenant détailler les principales méthodes de résolution qui ont été mises en œuvre depuis les cinquantes dernières années. La connaissance de ces méthodes n'est pas nécessaire pour suivre le reste du mémoire. Il est cependant intéressant d'en avoir un aperçu pour comprendre les principales évolutions des méthodes de résolution qui ont été inventées pour résoudre le problème LESC.

2.3.3 Méthodes de résolution exactes

Algorithme Dualoc

L'algorithme Dualoc de Erlenkotter (1978) est sans doute l'un des plus connus et l'un des plus efficaces pour résoudre le problème LESC de manière exacte. Il utilise la formulation forte du problème LESC afin d'exploiter certaines propriétés du dual du problème. L'algorithme se base sur deux idées :

- éviter la dégénérescence du primal en exploitant la simplicité et les solutions multiples du dual du problème,
- utiliser un algorithme d'évaluation et séparation pour rendre la solution entière si elle ne l'est pas.

La méthode s'articule ainsi de la manière suivante :

- dans un premier temps, une procédure d'ascension duale est exécutée, suivie d'une procédure d'ajustement afin d'obtenir une solution duale. Cette procédure fournit un candidat pour une solution entière primale.
- dans un deuxième temps, si ces deux procédures ne trouvent pas directement une solution optimale entière, un algorithme d'évaluation séparation est exécuté.

La procédure d'ascension duale part d'une solution duale arbitraire. À partir de cette solution duale, elle cycle sur les clients j les uns après les autres en augmentant séquentiellement chaque variable duale correspondante v_j aussi longtemps que la solution sous-jascente respecte les contraintes du problème dual. Si une contrainte bloque l'augmentation de la variable v_j, c'est que v_j est augmentée au niveau maximum autorisé par la contrainte. Quand toutes les variables duales v_j sont bloquées, la procédure se termine. Cette procédure fournit alors une solution duale et une solution primale

candidates. Si toutes les conditions d'écart complémentaires sont satisfaites alors la solution candidate est optimale, sinon l'algorithme entame la procédure d'ajustement dual. Il s'agit d'une procédure d'ascension duale similaire à celle qui a été introduite par Bilde et Krarup (1977).

La procédure d'ajustement duale, quant à elle, diminue certaines variables v_j afin de créer une marge sur des sites i qui pourra être utilisée pour augmenter d'autres variables v_j. La procédure est répétée aussi longtemps que l'objectif dual s'améliore.

Si la procédure d'ascension duale et la procédure d'ajustement dual n'ont pas fourni de solution optimale entière, l'algorithme d'évaluation et séparation qui utilise les bornes des procédures précédentes débute jusqu'à en obtenir une. Cet algorithme d'évaluation et séparation a volontairement été conçu par Erlenkotter le plus simplement possible car il avait remarqué que ces deux procédures donnaient des bornes telles qu'il n'était pas nécessaire de l'optimiser davantage pour les instances qu'il résolvait. Le branchement se fait ainsi directement sur les sites : pour un site i violant une contrainte d'écart complémentaire, l'algorithme fixe la valeur du coût fixe f_i à $+\infty$ pour simuler le fait que le site est fermé puis relance les deux procédures d'ascension et d'ajustement duale. Ensuite, pour résoudre l'autre branche, il fixe la valeur de f_i à 0 tout en réduisant les variables v_i pour garder la réalisibilité dans le dual et simuler le fait que le site est ouvert avant de relancer encore une fois les deux procédures. Enfin, comme à chaque nœud de branchement les deux procédures sont utilisées, une solution primale est obtenue ce qui permet d'obtenir une solution entière optimale à la fin de l'algorithme.

Algorithme BBdual

Comme l'algorithme Dualoc de Erlenkotter, l'algorithme BBdual introduit par Janacek et Kovacikova (1997) est basé sur l'idée de procédures d'ascension et d'ajustement duals encastrées dans un algorithme d'évaluation et séparation. Une des différences entre les deux algorithmes réside dans l'algorithme d'évaluation et séparation qui, dans le cas de l'algorithme BBdual, fait le branchement sur les variables y_i. En effet, au lieu de modifier des valeurs de f_i à chaque nœud de branchement, l'algorithme fixe les valeurs des y_i à 0 ou 1 et résout les deux sous-problèmes qui en découlent. La stratégie de branchement est aussi différente puisqu'au lieu de choisir au hasard les sites sur lesquels brancher, une stratégie profondeur d'abord est utilisée. Enfin, les bornes dictant si un sous-problème doit être résolu ou non sont obtenues

avec une procédure d'ascension et d'ajustement duale tout comme dans l'algorithme Dualoc.

Janacek et Buzna (2008) ont amélioré cette méthode en modifiant la procédure d'ascension duale de manière à exploiter plus efficacement le potentiel d'augmentation de la borne inférieure du dual. Cette amélioration consiste à donner priorité aux clients les plus prometteurs pour l'augmentation de cette borne. Elle a été testée sur deux familles d'instances : la première qui donne des résultats impressionnants puisque l'amélioration permet de diviser le temps de calcul d'un facteur 19 à 127, et la deuxième qui donne des résultats plus aléatoires avec un facteur d'amélioration du temps de calcul allant de 0,2 (< 1 donc dégradation du temps de calcul) à 2,5.

Algorithme PDLoc

L'algorithme PDLoc de Korkel (1989) est basé encore une fois sur les mêmes principes que l'algorithme Dualoc de Erlenkotter : un algorithme d'évaluation et séparation combiné à des procédures d'ascension et d'ajustement duales sont utilisés, mais cette fois-ci avec un certain nombre d'améliorations significatives. Outre les différences dans l'algorithme d'évaluation et de séparation qui utilise une stratégie de minimisation de la borne inférieure dans l'arbre de branchement, un certain nombre de procédures additionnelles ont été apportées dans les procédures d'ascension et d'ajustement duales. Janacek et Buzna (2008) expliquent bien ces différences que nous allons retranscrire.

La première variation se retrouve dans le choix des clients lors de l'exécution des procédures : leur ordre est volontairement changé pour que de nouveaux sites soient ouverts à chaque fois, ce qui permet d'explorer un spectre plus large de solutions primales. Cela fait décroître plus rapidement la borne inférieure et donc les procédures utilisées peuvent se terminer plus rapidement.

La deuxième variation qui permet également de gagner en efficacité s'exécute uniquement dans le cas où les coûts fixes f_i sont très grands par rapport aux coûts d'affectation c_{ij}. Cette nouvelle procédure commence par calculer la première solution duale par une procédure d'ascension multiple au lieu d'une procédure d'ascension duale. En fait, les variables duales v_j sont initialisées au début à une grande valeur pour être ensuite ajustées pour épouser les contraintes. L'exécution de la traditionnelle procédure d'ascension duale vient ensuite terminer le processus.

La troisième variation consiste à appliquer une simple heuristique d'échange après

avoir obtenu la première solution primale. En effet, quand une grosse différence entre la borne supérieure et la borne inférieure apparaît, elle est réduite par une procédure d'ajustement duale modifiée qui comporte deux phases : la première appelée "ajustement primal-dual multiple" qui réduit fortement les variables duales v_i puis les incrémente graduellement par l'intermédiaire d'une procédure d'ascension duale, et la deuxième appelée "ajustement primal-dual" qui réduit les v_j en boucle, variable par variable, et dont la solution duale résultante est utilisée par la procédure d'ascension duale pour terminer. Le nombre de répétitions de cette grande procédure d'ajustement dual dépend de la taille du problème.

Enfin, la dernière variation se situe dans l'algorithme d'évaluation et séparation : l'algorithme autorise de fixer l'état des sites (ouvert $y_i = 1$ ou fermé $y_i = 0$) de manière permanente et ainsi de réduire la taille du problème à résoudre. Pour fixer une variable, des conditions spéciales doivent être satisfaites. Cependant, comme l'évaluation de ces conditions est très coûteuse en terme de temps de calcul, surtout lorsqu'on est profond dans l'arbre de branchement, il est préférable de fixer des variables au niveau de la racine de l'arbre de branchement. Si le branchement voulu est profond dans l'arbre, les variables sont fixées uniquement s'il y a la possiblilité d'en fixer plusieurs simultanément.

Cette méthode est l'algorithme exact qui produit les meilleurs résultats selon ReVelle et Eiselt (2005). La procédure d'ascension duale a d'ailleurs également été améliorée par Janacek et Buzna (2008) comme ils l'avaient fait pour l'algorithme BBdual en y introduisant une sélection dans le choix des clients par rapport à leur potentiel d'amélioration de la borne inférieure du dual. Les deux familles d'instances sur lesquelles cette amélioration a été testée donnent une division du temps de calcul par un facteur allant de 1,3 à 2,1 pour la première famille, et de 1,4 à 2,2 pour la deuxième famille.

Algorithme de pincement et séparation

Goldengorin *et al.* (2004) ont tenté d'améliorer l'algorithme d'évaluation et séparation utilisé pour la résolution du problème LESC en le remplaçant par un algorithme de "pincement et séparation" (*branch and peg* en anglais). Pour faire cela, ils utilisent la formulation de Beresnev du problème LESC et implémentent une fonction pseudo-booléenne nommée fonction de Beresnev qui permet de voir l'effet global qu'aura l'ouverture de n'importe quel site dans une configuration donnée. Cela permet

plusieurs améliorations dont la possibilité d'élaborer des règles efficaces concernant l'installation des entrepôts sur les sites. Une de ces règles est par exemple utilisée en pré-traitement à chaque nœud de branchement pour fixer des variables dans les sous-problèmes correspondants afin de déterminer avant de brancher si tel ou tel site doit être ou non ouvert. Cela réduit donc d'emblée la taille des sous-problèmes, ce qui accélère les calculs. L'autre amélioration de la méthode est que grâce aux résultats donnés par la fonction de Beresnev, une fonction de branchement efficace permettant de mieux déterminer sur quelles variables faire le branchement peut être implémentée. Ainsi, comme le choix des variables est fondamental dans l'efficacité du branchement, encore un peu de temps de calcul est gagné à cet endroit. L'algorithme est testé successivement avec trois différentes règles de branchement et il a été remarqué que certaines d'entre elles sont plus efficaces que d'autres selon la densité de la matrice d'affectation.

Au niveau expérimental, cet algorithme surpasse systématiquement un algorithme d'évaluation et séparation "classique" en particulier lorsque la matrice d'affectation est dense. Il lui faut en moyenne dans ce cas moins de 10% du temps d'exécution de l'algorithme d'évaluation et séparation classique pour résoudre le problème.

Relaxation lagrangienne et semi-lagrangienne

La relaxation lagrangienne a été adaptée par Guignard (2003) avec moins de succès que les algorithmes basés sur les procédures d'ascension duale à cause de ses performances inférieures. Cependant, Beltran-Royo *et al.* (2007) ont tenté de remettre cette méthode sur le devant de la scène en introduisant un nouvel algorithme s'en inspirant : la relaxation semi-lagrangienne (RSL). Cette RSL consiste à dualiser les containtes (2.2) du problème vues dans la section précédente (c'est-à-dire insérer dans la fonction objectif une expression de ces contraintes dégradant sa valeur si ces contraintes sont violées), mais au lieu de les enlever définitivement des contraintes du problème, elles continuent à rester dans le sous-problème mais sous la forme d'une inégalité faible. Cela rend le sous-problème un peu plus difficile à résoudre que l'approche standard, mais cela réduit très fortement le saut de dualité (qui est identique au saut d'intégrité dans ce cas). Ensuite, ce sous-problème, aussi appelé problème semi-Lagrangien dual est résolu. Deux méthodes sont essayées pour résoudre ce problème semi-Lagrangien dual : une première qui est une variante de la méthode d'ascension duale de Korkel (1989), et une deuxième faisant appel à la méthode des plans coupants nommée

méthode analytique approximative centrale des plans coupants. La méthode a été testée sur deux types d'instances et semble performante pour le premier type car elle bat l'heuristique hybride avec départs multiples de Resende et Werneck (2006) que nous verrons dans la sous-section suivante, en trouvant la solution optimale en des temps en moyenne seulement 64% supérieurs. Pour le deuxième type d'instance, la méthode prend trop de temps à résoudre donc lorsqu'elle est arrêtée, elle ne fournit presque que des résultats inférieurs à ceux de l'heuristique hybride avec départs multiples. La comparaison n'est faite qu'avec cette heuristique hybride avec départs multiples, ce qui empêche de tirer plus de conclusions sur les performances de cette méthode.

2.3.4 Heuristiques

Recherche Tabou

La recherche Tabou a été adaptée au problème LESC pour la première fois par Sultan et Fawzan (1999). L'algorithme a été implémenté de la manière suivante : un premier algorithme glouton nommé "heuristique du bénéfice net" permet d'obtenir une solution préliminaire qu'il donne à la procédure de recherche tabou qui affine la solution. L'algorithme glouton est composé de deux phases : une phase initiale qui consiste à ouvrir pour chaque client le site ayant le coût d'affectation le plus faible, puis une phase de raffinement qui calcule pour chaque site ouvert i le coût engendré par sa fermeture (*pertes liées à la réaffectation des clients anciennement servis par le site i moins f_i*) et si ce coût est négatif, le site i est fermé. Une solution préliminaire de qualité acceptable est ainsi obtenue. L'algorithme procède ensuite aux itérations de recherche tabou. Cette recherche tabou possède les spécificités suivantes :

- l'ensemble des solutions est l'ensemble des combinaisons possibles des ouvertures de sites (c'est-à-dire l'ensemble des vecteurs $s = [y_1, y_2, ..., y_n]$ possibles avec $y_i = 0$ ou 1, $\forall\ i \in \{1, ..., n\}$),
- la fonction d'évaluation de la solution est celle qui calcule le coût total (*coûts d'affectation + coûts d'implantation*),
- le voisinage est l'ensemble des solutions obtenues par inversion de l'état des sites d'une solution donnée (c'est-à-dire pour une solution donnée $s = [y_1, y_2, ..., y_n]$, le voisinage sera $N(s) = \{[\bar{y}_1, y_2, ..., y_n], [y_1, \bar{y}_2, ..., y_n], ..., [y_1, y_2, ..., \bar{y}_n]\}$ où $\bar{y}_i = 1 - y_i$). Cependant, tout le voisinage n'est pas considéré : un système de tirage

aléatoire est mis en place permettant de choisir un élément du voisinage. Ce système de tirage aléatoire est appelé $5n$ fois par itération (chaque appel ne débouche pas nécessairement sur l'obtention d'un élément du voisinage).
- le critère d'arrêt est un nombre fixé d'itérations sans amélioration de la meilleure solution.
- pour chaque élément du voisinage considéré, toutes les affectations clients-sites sont cherchées à nouveau, et pour leur évaluation, la valeur de la solution résultante est recalculée entièrement.

Sur les instances testées l'algorithme est rapide selon Sultan et Fawzan et atteint à chaque fois l'optimum, mais il n'est pas testé sur de grosses instances où la méthode pourrait perdre son efficacité. En effet, les plus grandes instances testées sont des instances de 57 sites et 57 clients. Cette implémentation de la recherche tabou semble quand même relativement lourde à cause de l'absence d'optimisation pour l'évaluation des éléments de voisinage considérés et du système de tirage aléatoire appelé $5n$ fois par itération alors qu'il semble particulièrement coûteux en terme de nombre d'opérations. Ce système de tirage aléatoire n'a d'ailleurs pas été repris dans les heuristiques qui ont suivi.

La recherche tabou a été reprise plus tard par Ghosh (2003), Michel et Hentenryck (2004) puis par Sun (2006). La version de Ghosh utilise le même voisinage (sans le système de tirage aléatoire), et ajoute une mémoire sur les fréquences qui décourage l'application de mouvemement effectués trop souvent afin de favoriser l'exploration de l'ensemble de l'espace des solutions. Au niveau des résultats, les performances sont bonnes par rapport aux algorithmes exacts, et légèrement meilleures qu'une autre heuristique qu'il a introduite en même temps : la recherche locale complète avec mémoire qui sera vue dans la sous section suivante.

La version de Michel et Hentenryck se différencie de celle de Sultan et Fawzan par l'adoption d'un voisinage légèrement différent : tant qu'il existe une solution dans le voisinage qui améliore la solution courante, le voisinage est le même (tout le voisinage est considéré comme pour Ghosh), par contre, si ce n'est pas le cas (c'est-à-dire que l'algorithme est dans un minimum local), cette recherche tabou choisit au hasard un site ouvert et le ferme. Elle se différencie également par l'utilisation d'un calcul incrémental qui permet de faire les remises à jour et les évaluations de voisinage bien plus rapidement. Le critère d'arrêt est le suivant : la recherche tabou s'arrête quand elle fait 500 itérations sans améliorer la meilleure solution. Enfin, la dernière différence

se trouve dans la taille de la liste taboue qui est variable dans le cas de Michel et Hentenryck.

En ce qui concerne la recherche tabou de Sun, elle se différencie en beaucoup de points de la première. La première différence est que deux listes taboues sont implémentées au lieu d'une : une pour les fermetures de sites et une pour les ouvertures. Ces deux listes ont des tailles différentes et variables et évoluent dans deux intervalles différents. La deuxième différence est au niveau du processus de mémoire : au lieu d'avoir simplement un processus de mémoire (liste taboue), Sun en a introduit trois : un à court terme, un à moyen terme et un à long terme. D'ailleurs, le critère d'arrêt dépend directement des informations produites par le processus de mémoire à long terme. La troisième différence est au niveau de la solution initiale : elle utilise une heuristique gloutonne déjà utilisée par Ghosh (2003) pour l'obtenir. Enfin, la dernière différence réside dans les remises à jour d'une itération à l'autre et dans les évaluations des mouvements : un calcul incrémental similaire à celui de Michel et Hentenryck est utilisé permettant d'accélérer fortement le déroulement de l'algorithme. Toutes ces améliorations font de cette recherche tabou l'une des heuristiques les plus performantes actuellement si ce n'est pas la plus performante pour la résolution du problème LESC. En effet, les tests réalisés comparent cette recherche tabou aux heuristiques de Ghosh (2003) et à l'hybride avec départs multiples de Resende et Werneck (2006) et montrent qu'elle les surpasse toutes sur presque toutes les familles d'instances testées.

Recherche locale complète avec mémoire

Cette méthode a été développée par Ghosh (2003) pour résoudre le problème LESC. Elle commence par définir deux ensembles nommés $Vivant$ et $Mort$. Ensuite, une solution initiale est créée et mise dans l'ensemble $Vivant$ avant de commencer la phase itérative. Pendant cette phase itérative, au début de chaque itération, une solution de l'ensemble $Vivant$ est choisie afin d'en explorer le voisinage (le même qui a été vu pour la recherche tabou) et les solutions de plus petite valeur sont sélectionnées. La solution préalablement choisie est ensuite mise dans l'ensemble $Mort$ et les solutions sélectionnées sont introduites dans l'ensemble $Vivant$ si elles ne sont pas déjà dans les ensembles $Vivant$ ou $Mort$. L'exécution de ces itérations dessine ainsi un arbre de voisinage qui est exploré au fur et à mesure. Ces itérations sont répétées jusqu'à ce que l'ensemble $Vivant$ soit vide ou qu'un certain nombre de nœuds

de cet arbre aient été explorés. Si l'ensemble $Vivant \neq \emptyset$ quand le critère d'arrêt est atteint, une post optimisation qui consiste à exécuter une recherche locale sur les meilleures solutions trouvées est mise en route. Cette méthode, qui est souvent utilisée comme méthode de comparaison pour d'autres métaheuristiques, fait partie des heuristiques les plus efficaces, même si les tests montrent que la recherche tabou implémentée par Ghosh (2003) la surpasse légèrement.

Hybride avec départs multiples et recherche à voisinages variables avec départs multiples

La méthode hybride avec départs multiples a été développée par Resende et Werneck (2006). Elle fonctionne en 2 phases :

- La première est un algorithme avec départs multiples et intensification : à chaque itération, une solution est construite au hasard et un algorithme de recherche locale lui est appliqué. Les solutions ainsi obtenues sont combinées avec plusieurs autres solutions d'un ensemble de solutions d'élite selon un processus appelé "*path-relinking*". Il en résulte une autre solution que l'algorithme réinsérera sous certaines conditions dans l'ensemble des solutions d'élite. Ces conditions dépendent de la valeur de la solution et d'autres facteurs.
- La deuxième phase est une post-optimisation qui combine les solutions de l'ensemble des solutions d'élite avec une autre solution du processus en espérant que cela donne une meilleure solution au final.

Cette méthode, appelée hybride à cause de sa structure qui mélange plusieurs métaheuristiques, fait également partie des méthodes les plus efficaces. En effet, elle a été comparée à la recherche tabou de Michel et Hentenryck (2004), et les résultats montrent que ces deux métaheuristiques sont à peu près équivalentes en terme de performance, mais que sur les instances modélisants des cas extrêmes du problème LESC, un certain avantage est donné à l'hybride avec départs multiples.

La recherche à voisinages variables avec départs multiples de Harm et Hentenryck (2005) est une autre métaheuristique qui ressemble à la précédente. Ce n'est autre qu'une recherche tabou dans laquelle on fait varier le voisinage, et qui est réitérée plusieurs fois selon des paramètres fixés. Harm et Van Hentenryck montrent d'ailleurs par leurs résultats que leur métaheuristique bat l'hybride avec départs multiples sur les instances testées.

Algorithme génétique

L'algorithme génétique a été appliqué au problème LESC par Kratica *et al.* (2001). Cet algorithme démarre d'une population de solutions intiales et exécute pendant un certain nombre d'itérations une procédure que nous allons détailler. Cette procédure commence par l'évaluation des solutions de la population, puis elle les classe de la meilleure à la moins bonne pour ensuite en sélectionner une partie pour la suite de la procédure (la priorité est donnée aux solutions de meilleure qualité). Ces solutions sélectionnées sont ensuite croisées selon un croisement uniforme, c'est-à-dire que deux de ces solutions dites parents vont transmettre une partie d'elles même à deux enfants de manière à ce que le premier enfant soit l'opposé du deuxième et que chaque élément constitutif de la solution (ou gène, c'est-à-dire la valeur d'une variable y_i) soit transmis selon une probabilité p_1 pour qu'il vienne du père. Si le gène ne vient pas du père, c'est celui de la mère qui est transmis. Enfin, une mutation simple est appliquée sur ces solutions selon une autre probabilité p_2 avant de recommencer une nouvelle itération. À chaque itération, le tiers de la population est remplacée. Des tests ont été effectués sur cet algorithme ainsi que sur l'algorithme Dualoc pour la comparaison. Ils montrent que pour une partie des instances testées, l'algorithme Dualoc est non seulement nettement plus rapide (temps 80 à 300 fois inférieurs) mais en plus, il donne une solution optimale que l'algorithme génétique n'est pas toujours capable de fournir (erreur comprise entre 0% et 1%). Par contre, pour l'autre partie des instances, la tendance s'inverse, en particulier lorsque la taille des problèmes augmente : l'algorithme Dualoc met trop de temps à résoudre et est donc arrêté avant la fin de la résolution. La meilleure solution après arrêt est entre 9,5% et 15,5% moins bonne que la borne donnée par l'algorithme génétique. Cette métaheuristique telle qu'elle est implémentée semble moins efficace que d'autres métaheuristiques comme la recherche tabou si l'on en croit Michel et Hentenryck (2004) ou Hoefer (2003), mais elle reste cependant une méthode importante, souvent mentionnée dans la littérature.

2.3.5 Autres méthodes

Seules les méthodes les plus connues et/ou les plus efficaces ont été présentées, mais beaucoup d'autres ont également été utilisées pour résoudre le problème LESC. Parmi ces méthodes, on peut citer les recherches locales sur le primal du problème de Kuehn et Hamburger (1963), Feldman *et al.* (1966),Teitz et Bart (1968), Manne (1964), les

algorithmes d'évaluation et de séparation de Efroymson et Ray (1966) et Khumawala (1972), la méthode Galvao et Raggi (1989), la méthode de projection de Conn et Cornuejols (1990), la recherche à voisinages variables primal-dual de Hansen *et al.* (2007), l'approche à correction de données de Goldengorin *et al.* (2003), et l'algorithme d'optimisation par essaims particulaires discrets de Guner et Sevkli (2008).

Chapitre 3

OBTENTION DE LA SOLUTION INITIALE

Comme vu précédemment, IPS augmente en efficacité s'il tire profit d'une bonne solution initiale. Par conséquent, il est préférable d'élaborer un algorithme capable d'en générer une rapidement, et qui peut l'affiner à un niveau de qualité désiré. Nous allons donc dans ce chapitre décrire la recherche tabou que nous avons implémentée pour obtenir cette solution initiale ainsi que les différentes stratégies d'accélération qui ont été mises en œuvre.

Afin d'optimiser le temps de calcul de l'ensemble de la résolution pour IPS avec solution initiale, il faut trouver un équilibre entre le temps de calcul nécessaire à la recherche de la solution initiale et le temps d'exécution de l'algorithme IPS. Pour cela, un algorithme glouton permettant d'obtenir une solution préliminaire avec un temps de calcul très faible mais dont la qualité de solution peut être aléatoire est mis en place, suivi de la recherche tabou dont il est possible de choisir le nombre d'itérations effectuées pour améliorer cette solution. Le choix du nombre d'itérations permettra de moduler la durée du calcul.

3.1 Heuristiques et mise en oeuvre

3.1.1 Algorithme glouton

Comme on vient de le voir, l'algorithme glouton est la première phase de résolution. Il permet d'aboutir à une première solution initiale. Le pseudo-code de cet algorithme est donné dans l'algorithme 1 et peut être décrit de la manière suivante :

1 - **Algorithme glouton**

INITIALISATION

1 : $I = \{1, 2, ..., n\}$ et $J = \{1, 2, ..., m\}$

2 : **Pour** chaque $i \in I$, **poser** $y_i = 0$ et $\textit{coût\_total}[i] = f_i + \sum_{j \in J} c_{ij}$.

OUVERTURE DU PREMIER SITE

3 : Soit $k \in \arg\min_{i \in I} \textit{coût\_total}[i]$.

4 : Poser $y_k = 1$ et $z := \textit{coût\_total}[k]$

5 : et $\textit{écart}[i] = \begin{cases} f_i - \sum_{j \in J} c_{kj} - \tilde{c}_{ij} & \text{si } i \in I \setminus \{k\} \\ \infty & \text{si } i = k \end{cases}$

où $\tilde{c}_{ij} = \min(c_{ij}, c_{kj}),\ \forall\ i \in I$ et $\forall\ j \in J$.

OUVERTURE DES SITES SUBSÉQUENTS

6 : **Tant que** $\min_{i \in I} \textit{écart}[i] < 0$, **faire**

7 : Soit $l \in \arg\min_{i \in I} \textit{écart}[i]$.

8 : Poser $y_l = 1$ et $z := z + \textit{écart}[l]$

9 : et $\textit{écart}[i] = \begin{cases} f_i - \sum_{j \in J} \tilde{c}_{lj} - \tilde{c}_{ij} & \text{si } i \in I \text{ et } y_i = 0 \\ \infty & \text{si } i = l \end{cases}$

où $\tilde{c}_{ij} = \min(\tilde{c}_{ij}, \tilde{c}_{lj}),\ \forall\ i \in I$ et $\forall\ j \in J$.

10 : **Fin tant que**

SOLUTION HEURISTIQUE

11 : **Retourner** y.

- Les lignes 1 et 2 représentent l'étape d'initialisation. On impose à tous les sites d'être préalablement fermés (ligne 2).
- Les étapes 3 et 4 concernent l'ouverture du premier site : le coût total engendré par l'ouverture de chaque site est évalué puis celui engendrant le coût total minimum est ouvert. Cela permet d'initialiser la fonction objectif z à cette valeur (ligne 4).
- À la ligne 5 on crée la "matrice des écarts unitaires" $(\tilde{c}_{ij})$ dont chaque $\tilde{c}_{ij}$ représente le coût d'affectation du client j si le site i est ouvert. Évidemment, ces $\tilde{c}_{ij}$ sont relatifs à la solution courante. Puis les écarts sont calculés. Un écart i représente le coût supplémentaire qu'il faudrait payer si l'on ouvrait le site i.

Le ∞ est présent pour que le premier site ouvert ne puisse pas être réouvert, ce qui n'aurait aucun de sens.

– Ensuite, quand on rentre dans la boucle **tant que** (lignes 6 à 10), le site engendrant l'écart le plus faible est choisi, puis ouvert si cet écart est négatif (puisque cela implique que l'on diminue le coût total). La fonction objectif z est remise à jour et on fixe $\acute{e}cart[k] = \infty$, là encore, pour interdire l'ouverture d'un site déjà ouvert (ligne 9). Ce processus de mise à jour de la matrice des écarts unitaires et des écarts (lignes 9) ainsi que le choix du site à ouvrir (ligne 7) est réitéré jusqu'à ce que tous les écarts soient positifs ou nuls puisque cela implique que n'importe laquelle des ouvertures autorisées ne ferait pas diminuer la valeur de la fonction objectif z.
– Ainsi, lorsque tous les $\acute{e}cart[i],\ i = 1, .., n$ sont positifs l'algorithme s'arrête.

Nous allons maintenant appliquer l'algorithme 1 sur l'instance d'exemple donnée par le tableau 3.1.

Sites (i)	*Coûts d'installation* (f_i)	*Matrice d'affectation* (c_{ij})					
Clients :		1	2	3	4	5	6
1	2	0	1	3	4	5	7
2	3	1	0	2	3	4	6
3	4	3	2	0	1	3	4
4	5	4	3	1	0	2	3
5	3	5	4	3	2	0	2
6	2	7	6	4	3	2	0

TABLEAU 3.1 Instance d'exemple 6×6

Commençons par calculer pour chaque site le coût total (étape 2) :

Sites	*Installation* (f_i)	*Mat. d'affectation* (c_{ij})	*Coût_total*
1	2	0 1 3 4 5 7	$\rightarrow 2 + \sum_j c_{1,j} = 22$
2	3	1 0 2 3 4 6	$\rightarrow 3 + \sum_j c_{2,j} = 19$
3	**4**	**3 2 0 1 3 4**	$\rightarrow 4 + \sum_j c_{3,j} = \mathbf{17}$
4	5	4 3 1 0 2 3	$\rightarrow 5 + \sum_j c_{4,j} = 18$
5	3	5 4 3 2 0 2	$\rightarrow 3 + \sum_j c_{5,j} = 19$
6	2	7 6 4 3 2 0	$\rightarrow 2 + \sum_j c_{6,j} = 24$

Le site 3 possède le coût total le plus petit ($k = 3$) par conséquent 3 est choisi. On a donc $y_3 = 1$ et une valeur de $z = 17$. Ensuite, pour les étapes suivantes, il faut créer la matrice des écarts unitaires ($\tilde{c}_{ij}$). Comme le site 3 a été choisi on a :

Mat. d'affectation (c_{ij})

0	1	3	4	5	7
1	0	2	3	4	6
3	2	0	1	3	4
4	3	1	0	2	3
5	4	3	2	0	2
7	6	4	3	2	0

$\Rightarrow$

Mat. écarts unitaires ($\tilde{c}_{ij}$)

0	1	0	1	3	4
1	0	0	1	3	4
3	**2**	**0**	**1**	**3**	**4**
3	2	0	0	2	3
3	2	0	1	0	2
3	2	0	1	2	0

Il faut maintenant calculer les écarts à partir de la matrice des écarts unitaires (étape 5) :

Sites	*Install.* (f_i)	*Écarts unitaires* ($\tilde{c}_{ij}$)	*Écarts*
1	2	0 1 0 1 3 4	$\rightarrow 2 - \sum_j \tilde{c}_{3j} - \tilde{c}_{1j} = -2$
2	3	1 0 0 1 3 4	$\rightarrow 3 - \sum_j \tilde{c}_{3j} - \tilde{c}_{2j} = -1$
3	4	3 2 0 1 3 4	$\rightarrow \infty$
4	5	3 2 0 0 2 3	$\rightarrow 5 - \sum_j \tilde{c}_{3j} - \tilde{c}_{4j} = 2$
5	3	3 2 0 1 0 2	$\rightarrow 3 - \sum_j \tilde{c}_{3j} - \tilde{c}_{5j} = -2$
6	**2**	**3 2 0 1 2 0**	$\rightarrow 2 - \sum_j \tilde{c}_{3j} - \tilde{c}_{6j} = \mathbf{-3}$

Le site 6 possède l'écart le plus petit donc $k = 6$. De plus, $\acute{e}cart[6] < 0$ par conséquent 6 est choisi. On a donc $y_6 = 1$, $\acute{e}cart[6] = \infty$ et la valeur de la solution devient $z = 17 - 3 = 14$. Ensuite, la matrice des écarts unitaires est remise à jour :

Mat. écarts unitaires ($\tilde{c}_{ij}$)

0	1	0	1	3	4
1	0	0	1	3	4
3	**2**	**0**	**1**	**3**	**4**
3	2	0	0	2	3
3	2	0	1	0	2
3	2	0	1	2	0

$\Rightarrow$

Mat. écarts unitaires ($\tilde{c}_{ij}$)

0	1	0	1	2	0
1	0	0	1	2	0
3	**2**	**0**	**1**	**3**	**4**
3	2	0	0	2	0
3	2	0	1	0	0
3	**2**	**0**	**1**	**2**	**0**

Maintenant, les écarts peuvent être remis à jour. Ensuite, on recommence ces même opérations jusqu'à ce que tous les *écart*$[k]$ soient positifs ou nuls (étapes 6 à 10).

Sites	*Install.* (f_i)	*Écarts unitaires* $(\tilde{c}_{ij})$	*Écarts*
1	**2**	**0 1 0 1 2 0**	$\rightarrow 2 - \sum_j \tilde{c}_{6j} - \tilde{c}_{1j} = \mathbf{-2}$
2	3	1 0 0 1 2 0	$\rightarrow 3 - \sum_j \tilde{c}_{6j} - \tilde{c}_{2j} = -1$
3	4	3 2 0 1 3 4	$\rightarrow \infty$
4	5	3 2 0 0 2 0	$\rightarrow 5 - \sum_j \tilde{c}_{6j} - \tilde{c}_{4j} = 4$
5	3	3 2 0 1 0 0	$\rightarrow 3 - \sum_j \tilde{c}_{6j} - \tilde{c}_{5j} = +1$
6	2	3 2 0 1 2 0	$\rightarrow \infty$

Le site 1 a l'écart le plus petit donc $k = 1$ et *écart*$[1] < 0$ par conséquent 1 est choisi. On a donc $y_1 = 1$, *écart*$[1] = \infty$ et la valeur de la solution devient $z = 14 - 2 = 12$.

Mat. écarts unitaires $(\tilde{c}_{ij})$

0	1	0	1	2	0
1	0	0	1	2	0
3	**2**	**0**	**1**	**3**	**4**
3	2	0	0	2	0
3	2	0	1	0	0
3	**2**	**0**	**1**	**2**	**0**

$\Rightarrow$

Mat. écarts unitaires $(\tilde{c}_{ij})$

0	**1**	**0**	**1**	**2**	**0**
0	0	0	1	2	0
3	**2**	**0**	**1**	**3**	**4**
0	1	0	0	2	0
0	1	0	1	0	0
3	**2**	**0**	**1**	**2**	**0**

Sites	*Install.* (f_i)	*Écarts unitaires* $(\tilde{c}_{ij})$	*Écarts*
1	2	0 1 0 1 2 0	$\rightarrow \infty$
2	3	0 0 0 1 2 0	$\rightarrow 3 - \sum_j \tilde{c}_{1j} - \tilde{c}_{2j} = 2$
3	4	3 2 0 1 3 4	$\rightarrow \infty$
4	5	0 1 0 0 2 0	$\rightarrow 5 - \sum_j \tilde{c}_{1j} - \tilde{c}_{4j} = 4$
5	3	0 1 0 1 0 0	$\rightarrow 3 - \sum_j \tilde{c}_{1j} - \tilde{c}_{5j} = 1$
6	2	3 2 0 1 2 0	$\rightarrow \infty$

Le site 5 a l'écart le plus petit donc $k = 5$ mais *écart*$[5] = 1 \geq 0$ (donc tous les écarts sont positifs) par conséquent on ne peut plus ouvrir d'entrepôts sans dégrader la solution. L'algorithme s'arrête. La solution pour cet exemple est donc $y_1 = 1, y_2 = 0, y_3 = 1, y_4 = 0, y_5 = 0, y_6 = 0$ et sa valeur est $z = 12$.

3.1.2 Recherche tabou

Mise en œuvre

La recherche tabou a été inventée par Glover (1986). C'est une méthode itérative qui explore à chaque itération les solutions d'un voisinage et en prend la meilleure tout en respectant le critère tabou et le critère d'aspiration. Cette nouvelle solution engendre ainsi un nouveau voisinage qui permet d'obtenir une nouvelle solution, et ainsi de suite. Chaque déplacement d'une solution à une autre est appelé mouvement. Lorsqu'un mouvement est effectué, il est mémorisé dans une liste taboue. Chaque mouvement de la liste taboue n'a pas le droit d'être défait tant qu'il est dans cette liste. Le nombre d'éléments et le temps que reste un mouvement dans cette liste sont des paramètres à fixer. Cette mémoire à court terme permet de pouvoir faire des descentes locales dans l'espace de solution sans pour autant rester coincé dans des minima locaux. Nous avons choisi la métaheuristique de recherche tabou car cette méthode fait partie actuellement des heuristiques les plus efficaces pour ce problème et sa mise en place est relativement simple.

Dans le cadre de cette recherche tabou, nous avons défini nos éléments de structure de la manière suivante :

- l'ensemble des solutions S a été choisi comme étant l'ensemble des combinaisons possibles des ouvertures de sites (c'est-à-dire l'ensemble des vecteurs $s = [y_1, y_2, ..., y_n]$ possibles avec $y_i = 0$ ou $1,\ \forall\ i \in \{1, ..., n\}$).
- la fonction d'évaluation z varie au cours de la recherche, mais pour l'instant, supposons qu'elle est égale à celle qui calcule le coût total de la solution : $z(s) = \sum_{i \in E} \sum_{j \in J} c_{ij} x_{ij} + \sum_{i \in E} f_i y_i$.
- un mouvement est l'inversion de l'état d'un site i :
  $[y_1, y_2, ..., y_i, ..., y_n] \rightarrow [y_1, y_2, ..., \bar{y}_i, ..., y_n]$.
  Cela veut dire que si le site i est ouvert ($y_i = 1$), le mouvement sera la fermeture du site i ($y_i = 0$), et inversement, si le site i est fermé ($y_i = 0$), le mouvement sera l'ouverture du site i ($y_i = 1$).
- le voisinage est l'ensemble des solutions obtenues par application de tous les mouvements possibles à partir d'une solution donnée, c'est-à-dire, pour une solution $s = [y_1, y_2, ..., y_n]$, le voisinage s'écrira :
  $N(s) = \{[\bar{y}_1, y_2, ..., y_n], [y_1, \bar{y}_2, ..., y_n], ..., [y_1, y_2, ..., \bar{y}_n]\}$.
  On remarquera que chaque voisinage contient exactement n éléments.

– le critère d'arrêt est un nombre fixe d'itérations *NbrIt* qui est donné en paramètre.

On remarque que cette structure ressemble à la structure de base utilisée dans les recherches tabou vues dans la revue de littérature : nous avons choisi de ne faire les mouvements que sur les sites, et en faisant l'inversion de l'état d'un seul site à la fois. En effet, puisque pour une répartition d'entrepôts donnée la meilleure combinaison des variables x_{ij} est d'affecter chaque client à l'entrepôt le plus proche, il est facile de trouver les meilleures valeurs des variables x_{ij}. Faire des mouvements sur les affectations serait donc une perte de temps. En fait, notre recherche tabou se différencie des autres principalement par la structure permettant de faire varier la fonction d'évaluation z que nous avons ajoutée, par les stratégies d'accélération que nous avons mis en place et que certaines n'ont pas, par la manière dont nous faisons varier la taille de la liste taboue, et enfin par le critère d'arrêt à nombre d'itérations fixe que nous avons choisi.
Le pseudo-code de la recherche tabou classique peut s'expliciter selon l'algorithme 2.

---

2 - **Recherche tabou**

1 : Calculer la solution gloutonne $s \in S$, poser $T := \emptyset$ et $s^* := s$;
2 : **Tant que** aucun critère d'arrêt n'est satisfait **faire**
3 : Déterminer une solution s' qui minimise $z(s')$ dans $N^T(s)$;
4 : **Si** $z(s') < z(s^*)$ **alors** poser $s^* := s'$;
5 : Poser $s := s'$ et mettre à jour T ;
6 : **Fin tant que**
7 : **Retourner** s^* ;

Il fait appel aux notations suivantes :

T la liste taboue ;
$N^T(s) = \{s' \in N(s)$ tel que $s' \notin T$ ou $z(s') < z(s^*)\}$;
s la solution courante ;
s' une solution du voisinage $N^T(s)$;
s^* la meilleure solution trouvée.

L'ensemble $N^T(s)$ est en fait l'ensemble des solutions qui ne sont pas taboues ainsi que celles qui le sont mais dont le statut tabou est levé en raison du critère d'aspiration (c'est-à-dire que le mouvement qui est tabou engendre une solution s' meilleure que tout ce qui a été trouvé jusque là).

D'autre part, nous avons considéré que tant que le nombre d'itérations fait par un mouvement dans la liste taboue est inférieur ou égal à la taille de la liste taboue, le mouvement reste tabou. C'est seulement quand ce nombre devient strictement supérieur à la taille de la liste taboue que le critère tabou du mouvement est levé. La taille de la liste taboue et le nombre d'itérations fait par un mouvement dans cette liste représente donc la même chose ici.

Enfin, dans notre cas, nous faisons varier la taille de la liste taboue dans un intervalle $[a, b]$ au cours de la recherche. Nous en reparlerons dans la suite de la section.

Complexité et calcul incrémental

L'étape la plus longue de cette recherche tabou est l'étape de la ligne 3 dans l'algorithme 2 à cause de l'évaluation du voisinage $N^T(s)$. En effet, pour chaque mouvement de $N^T(s)$, il faut évaluer la valeur de la solution résultante pour savoir laquelle est la meilleure. Une méthode simple pourrait être de recalculer entièrement pour chacun des mouvements la valeur de la fonction objectif de cette solution résultante en additionnant tous les coûts d'affectation et les coûts d'installation. Cependant, cela prendrait beaucoup de temps puisque pour les n mouvements du voisinage, il serait nécessaire de rechercher quel site est affecté à quel client, additionner les coûts d'affectation en conséquence et additionner les coûts d'installation ensuite. Cela aboutirait ainsi à une complexité en $O(n(mn + n))$ c'est-à-dire $O(n^2m)$ pour évaluer $N^T(s)$. Le terme mn est présent car pour chaque client m, il est nécessaire de parcourir les n sites pour savoir à quel site le client est affecté (le client est affecté au site ouvert qui possède le coût d'affectation le plus faible, mais il est nécessaire de parcourir l'ensemble des sites pour être sûr que le site choisi est bien celui qui possède le coût minimum et qu'il est ouvert). Le terme $+n$ est présent pour additionner les coûts d'installation. Enfin, le coefficient n devant le $(mn + n)$ est présent car il y a n éléments dans le voisinage. La complexité pour l'ensemble de cette métaheuristique serait donc en $O(NbrIt\ n^2m)$, où $NbrIt$ est le nombre d'itérations effectuées.

Ainsi, afin de réduire le temps de calcul, un certain nombre d'optimisations ont été faites dans la manière d'évaluer les éléments du voisinage. La première amélioration consiste à ne pas recalculer l'ensemble de la fonction objectif à chaque mouvement, mais plutôt calculer simplement l'écart qu'il engendre. Pour cela, il suffit de soustraire les anciens coûts d'affectation des clients réaffectés (terme $\boxed{1}$ dans l'égalité (3.1)), d'additionner les nouveaux coûts d'affectation de ces mêmes clients (terme $\boxed{2}$ dans l'égalité (3.1)) et d'ajouter ou enlever le coût d'installation du site concerné par le mouvement, suivant que le mouvement est une ouverture ou une fermeture de site (terme $\boxed{3}$ dans l'égalité (3.1)). On obtient ainsi l'écart qu'il suffit d'additionner à la valeur de la solution courante s pour avoir la valeur de la nouvelle solution s' après application du mouvement. Si cet écart est négatif, la nouvelle valeur de la solution sera inférieure à l'ancienne, et s'il est positif la valeur de la nouvelle solution sera supérieure. On ne s'intéresse ainsi qu'à ce qui change. Ce calcul est appelé "calcul incrémental". Il peut être résumé avec l'égalité 3.1.

Pour un mouvement sur le site p,

$$z(s') = z(s) - \underbrace{\sum_{j \in F_{s,p}} c_{e_{s,j}j}}_{\boxed{1}} + \underbrace{\sum_{j \in F_{s,p}} c_{e_{s',j}j}}_{\boxed{2}} + \underbrace{\alpha_p f_p}_{\boxed{3}} \tag{3.1}$$

où s est la solution courante (c'est-à-dire avant application du mouvement) ;
s' est la solution résultante (c'est-à-dire après application du mouvement) ;
$F_{s,p}$ est l'ensemble des clients réaffectés dans la solution s si le mouvement sur le site p est fait ;
$e_{s,j}$ est l'entrepôt (ou site) auquel le client j est affecté dans la solution s ;
$\alpha_p = \begin{cases} 1 & \text{si le mouvement est l'ouverture du site } p \\ -1 & \text{si le mouvement est la fermeture du site } p. \end{cases}$

On appelera Δz l'écart entre $z(s')$ et $z(s)$:

$$\Delta z = -\sum_{j \in F_{s,p}} c_{e_{s,j}j} + \sum_{j \in F_{s,p}} c_{e_{s',j}j} + \alpha_p f_p \ .$$

Ainsi, si le mouvement sur le site p est une ouverture, les clients réaffectés à cause du mouvement sont ceux qui vont voir leur affectation changer pour ce site p dans s'.

L'égalité (3.1) devient donc :

$$z(s') = z(s) - \sum_{j \in F_{s,p}} c_{e_{s,j}j} + \sum_{j \in F_{s,p}} c_{pj} + f_p \ .$$

Cela est illustré dans la figure 3.1 où le mouvement est l'ouverture du site 4 et où $z(s) = 355$ avant application du mouvement. On obtient dans le cas de cet exemple : $z(s') = 355 - (40 + 40) + (30 + 30) + 95 = 430$, ce qui est bien le même résultat que si on avait recalculé la valeur de la fonction objectif $z(s')$ entièrement.

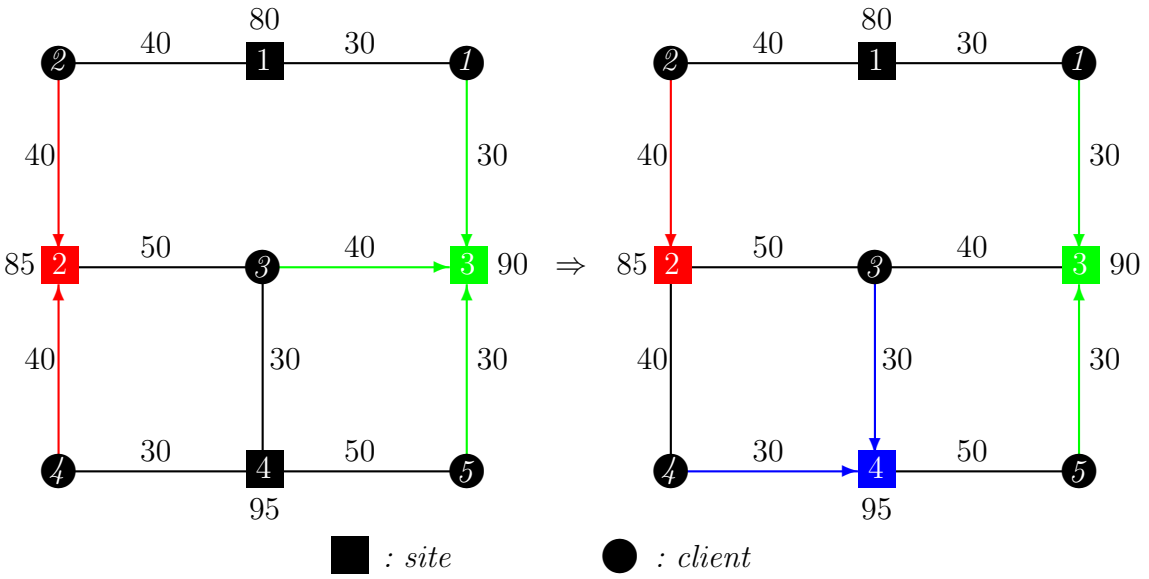


FIGURE 3.1 Graphes exemple d'un mouvement d'ouverture

Dans le cas où le mouvement sur le site p est une fermeture, les clients réaffectés sont ceux qui vont être désaffectés du site p dans s. L'égalité (3.1) devient ainsi :

$$z(s') = z(s) - \sum_{j \in F_{s,p}} c_{pj} + \sum_{j \in F_{s,p}} c_{e_{s',j}j} - f_p \ .$$

Dans la figure 3.2, le mouvement est la fermeture du site 2, la valeur de la fonction objectif avant application du mouvement est $z(s) = 440$, et on obtient après application du mouvement $z(s') = 440 - (40 + 30 + 30) + (50 + 50 + 40) - 85 = 395$, ce qui correspond bien au même résultat que si on avait recalculé la valeur de la fonction objectif $z(s')$ entièrement.

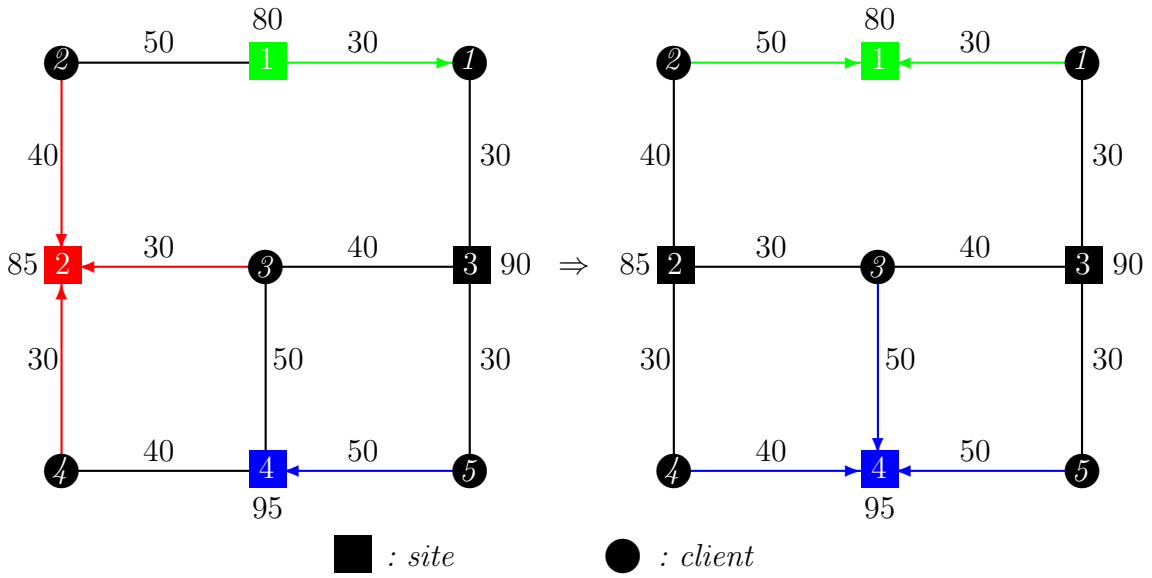


FIGURE 3.2 Graphes exemple d'un mouvement de fermeture

On notera cependant que même si ce calcul incrémental semble intéressant, il demande de déterminer l'ensemble $F_{s,p}$ des clients réaffectés à cause du mouvement ainsi que les entrepôts $e_{s,j}$ et $e_{s',j}$ auxquels les clients de l'ensemble $F_{s,p}$ sont et vont être affectés. Cela nécessite de parcourir l'ensemble des sites E pour chaque client de l'ensemble des clients J. En effet, dans le cas d'une ouverture de site, chaque client est affecté à un entrepôt mais il faut bien parcourir les coûts c_{ij} pour savoir quels clients seront réaffectés à l'entrepôt p dans la solution s' et parcourir les variables x_{ij} pour savoir quel client était affecté à quel entrepôt dans la solution s afin de soustraire les coûts d'affectation c_{ij} correspondants. De la même façon, dans le cas de la fermeture, il faut bien également parcourir les variables x_{ij} pour savoir quels clients étaient affectés à l'entrepôt p dans s ainsi que les coûts d'affectation c_{ij} pour savoir à quel entrepôt ils seront réaffectés dans s'.

Pour réduire ce parcours qui peut être fastidieux, nous avons mis en place un certain nombre d'outils.

Tableau de clients

Le premier outil est un tableau nommé "*tableau_clients*" qui garde en mémoire pour chaque client, son site d'affectation dans la solution s :

Client	1	2	3	...	m
Site d'affectation	$e_{s,1}$	$e_{s,2}$	$e_{s,3}$	...	$e_{s,m}$

Ce *tableau_clients* permet de connaître directement deux choses :

- dans le cas d'un mouvement d'ouverture, il permet de connaître directement les sites d'affectation $e_{s,j}$ sans avoir à parcourir les variables x_{ij}. Il suffit donc, pour calculer la somme $\boxed{1}$ de l'égalité (3.1), d'additionner les coûts d'affectation $c_{e_{s,j}j}$ en prenant pour chaque client j de l'ensemble $F_{s,p}$ le site d'affectation correspondant $e_{s,j}$ dans le *tableau_clients*.
- Dans le cas d'un mouvement de fermeture, il permet de connaître directement l'ensemble $F_{s,p}$ des clients réaffectés à cause du mouvement : il suffit de parcourir le *tableau_clients* en retenant les clients qui ont pour site d'affectation le site qui ferme p.

C'est également avec ce *tableau_clients* que l'on met en mémoire les affectations d'une solution : chaque couple (*client*, *site d'affectation*) représente une variables x_{ij} égale à 1. Toutes les autres variables x_{ij} sont égales à 0 (figure 3.3). Cela évite d'avoir à mettre en mémoire les $n \times m$ valeurs des variables x_{ij}.

Client	...	j	...
Site d'affectation	...	$e_{s,j}$	...

$$\Leftrightarrow \quad x_{ij} = \begin{cases} 1 & \text{si } i = e_{s,j} \\ 0 & \text{si } i \in E \setminus \{e_{s,j}\} \end{cases}$$

FIGURE 3.3 Équivalence entre le *tableau_clients* et les valeurs des variables

Enfin, ce tableau nécessite d'être remis à jour à chaque itération de recherche tabou une fois le meilleur mouvement du voisinage $N^T(s)$ choisi. Cette remise à jour permet d'enregistrer les affectations de la nouvelle solution s'. Elle prend un temps limité en $O(m)$ à s'exécuter (le temps d'inscrire le nouveau numéro des nouveaux sites d'affectation pour les clients qui ont été réaffectés à cause du mouvement).

Tableau des ouvertures

Le deuxième outil mis en place pour améliorer la recherche tabou est un tableau de listes appelé "*tableau_ouverture*". Ce *tableau_ouverture* permet d'avoir directement l'ensemble $F_{s,p}$ pour les mouvements d'ouverture de site p sans avoir à comparer le coût d'affectation c_{pj} avec le coût d'affectation $c_{e_{s,j}j}$ pour chaque client j (si son coût d'affectation actuel $c_{e_{s,j}j}$ est strictement supérieur au coût d'affectation c_{pj} entre ce client et le site qui ouvre alors le client j appartient à l'ensemble $F_{s,p}$). Le tableau contient dans chacune de ses cases une liste de numéros qui désigne les clients qui seront réaffectés si le site correspondant à la case était ouvert.

Il a donc la forme indiquée au tableau 3.2 pour une solution s donnée,

où $j_{p,s,u}$ est le numéro du $u^{ème}$ client réaffecté à cause de l'ouverture du site p dans la solution s ;

v_p est le nombre de clients réaffectés à cause de l'ouverture du site p.

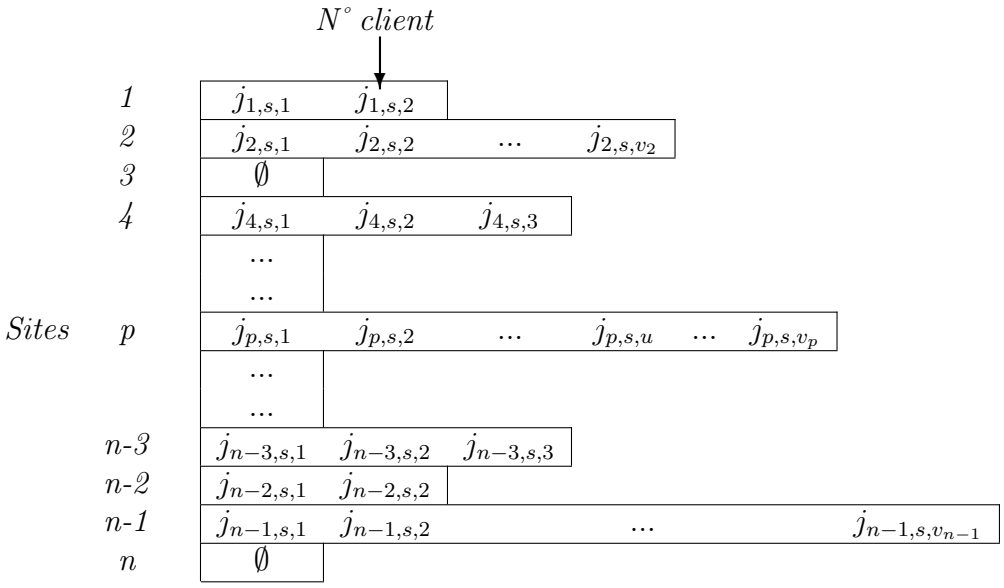


TABLEAU 3.2 Forme d'un *tableau_ouverture*

Si le mouvement sur le site p est une ouverture, la liste p du tableau représente les éléments de l'ensemble $F_{s,p}$. Par contre, si le mouvement sur le site p est une fermeture, la liste p du tableau est vide (représenté par ∅ dans le tableau 3.2) mais

cela ne veut pas dire que $F_{s,p}$ l'est. Le traitement d'un mouvement d'ouverture et de fermeture est fait différemment : ce tableau n'est utilisé que pour les mouvements d'ouverture. On a donc, pour une solution s et pour tout site p tel que le mouvement sur p est une ouverture :

$$F_{s,p} = \{j_{p,s,1}\ ,\ j_{p,s,2}\ ,\ \ldots\ ,\ j_{p,s,v_p}\}.$$

L'évaluation des mouvements d'ouverture aboutit ainsi à une complexité en $O(n \times m)$.

La constuction de ce tableau, qui sera expliquée dans la suite, est faite à chaque itération.

Tableau des fermetures

Le troisième outil mis en place pour améliorer la recherche tabou ressemble beaucoup à l'outil précédent sauf qu'il est destiné à améliorer l'efficacité de l'évaluation des mouvements de fermeture. Cet outil est un tableau de listes appelé "*tableau_fermeture*" qui contient dans chacune de ses cases une liste d'objets qui représente la liste des clients qui seront réaffectés si le site correspondant à la case était fermé. Chacun de ces objets possède deux attributs :

- le numéro du client correspondant (N° *client*),
- le numéro du site qui remplacera l'ancien site d'affectation si celui-ci est fermé (N° *site remplacement*).

Ce *tableau_fermeture* a donc la forme indiquée dans le tableau 3.3 pour une solution s donnée, et :

$j_{p,s,u}$	est le numéro du $u^{ème}$ client réaffecté à cause de la fermeture du site p dans la solution s ;
$e_{s',j_{p,s,u}}$	est le numéro du site auquel le client $j_{p,s,u}$ sera réaffecté si le site p est fermé ;
v_p	est le nombre de clients réaffectés à cause de la fermeture du site p.

Si le mouvement sur le site p est une fermeture, la liste p du tableau spécifie tous les clients qui seront réaffectés : les premiers arguments des couples constituent l'ensemble $F_{s,p}$ et les deuxièmes arguments indiquent leur site de remplacement $e_{s',j}$ respectif. Par contre, si le mouvement sur le site p est une ouverture, la liste p du

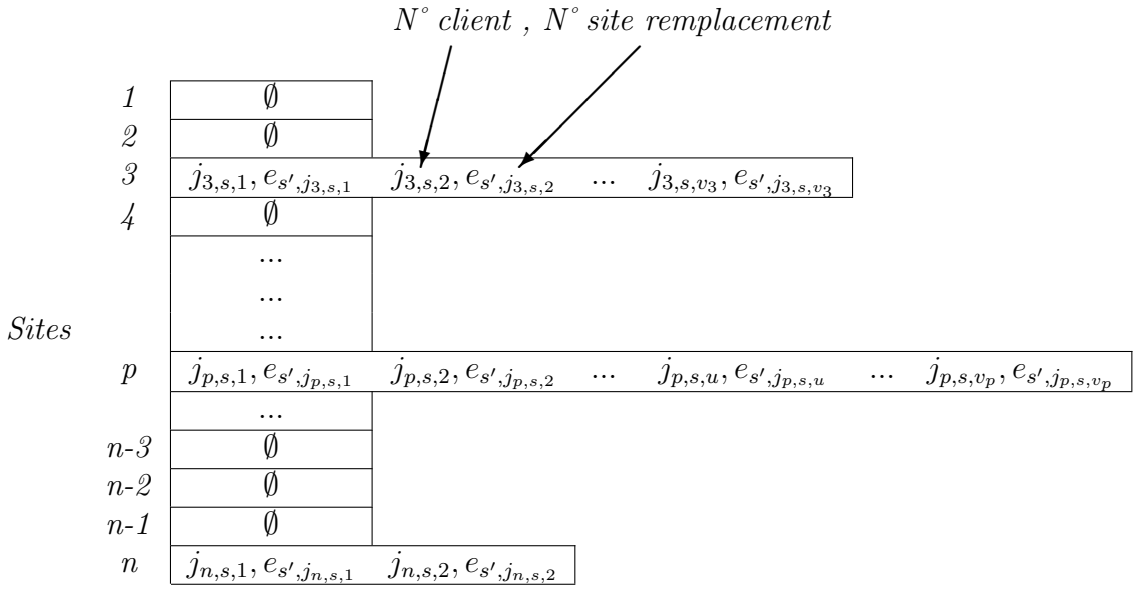


TABLEAU 3.3 Forme d'un *tableau_fermeture*

tableau est vide (∅ dans le tableau ci-dessus) mais cela ne veut pas dire que $F_{s,p}$ l'est : il faut en fait aller voir dans le *tableau_ouverture* pour l'obtenir. On a donc pour une solution s et pour tout site p tel que le mouvement sur p est une fermeture :

$$F_{s,p} = \{j_{p,s,1}\ ,\ j_{p,s,2}\ ,\ \ldots\ ,\ j_{p,s,v_p}\}$$

et leurs sites remplaçants $e_{s',j_{p,s,1}}\ ,\ e_{s',j_{p,s,2}}\ ,\ \ldots\ ,\ e_{s',j_{p,s,v_p}}$ correspondants.

Ce *tableau_fermeture* permet ainsi de connaître directement les ensembles $F_{s,p}$ sans avoir à parcourir les m cases du *tableau_clients* pour chaque mouvement de fermeture, et aussi de ne pas avoir à parcourir les n coûts d'affectation c_{ij} de la colonne j de la matrice d'affectation pour chaque client affecté $j \in F_{s,p}$ pour connaître les sites de remplacement.

Ce tableau permet ainsi de réduire la complexité des calculs. En effet, parcourir les m cases du *tableau_clients* à chaque mouvement de fermeture pour voir quels clients sont réaffectés, et regarder pour chacun d'eux les n coûts d'affectation c_{ij} qui leur sont rattachés implique une complexité en $O(n^2m)$. Alors qu'avec ce *tableau_fermeture* on a simplement à parcourir les éléments des n listes, ce qui donne une complexité en $O(nm)$.

À chaque itération, ce tableau est reconstruit en un temps en $O(nm)$. En effet, il faut parcourir le tableau des clients afin de reporter le numéro de chaque client

dans la liste correspondant au site auquel il est affecté (m opérations), et pour chacun de ces clients, parcourir les n éléments de la colonne de la matrice d'affectation lui correspondant pour trouver les sites de remplacement (site ouvert ayant le coût d'affectation c_{ij} le plus faible). Nous verrons dans la sous-section suivante comment cette construction peut être améliorée grâce à une matrice d'affectation ordonnée.

Matrice d'affectation ordonnée

Le quatrième outil mis en place pour améliorer la rapidité de la recherche tabou est une matrice d'objet appelé "*matrice_affectation_ordonnée*". Chacun des objets qu'elle stocke possède deux attributs :

- un coût d'affectation ;
- le numéro du site auquel ce coût d'affectation correspond.

a. Construction et intégration des objets

matrice d'affectation

sites \ *clients*	*1*	*2*	*3*	*4*	*5*	*6*
1	0	1	3	4	5	7
2	1	0	2	3	4	6
3	3	2	0	1	3	4
4	4	3	1	0	2	3
5	5	4	3	2	0	2
6	7	6	4	3	2	0

N° site / *coût d'affec.* ⇒

clients	*1*	*2*	*3*	*4*	*5*	*6*
	0,1	1,1	3,1	4,1	5,1	7,1
	1,2	0,2	2,2	3,2	4,2	6,2
	3,3	2,3	0,3	1,3	3,3	4,3
	4,4	3,4	1,4	0,4	2,4	3,4
	5,5	4,5	3,5	2,5	0,5	2,5
	7,6	6,6	4,6	3,6	2,6	0,6

b. Tri de chaque colonne par ordre croissant des coûts d'affectation

clients	*1*	*2*	*3*	*4*	*5*	*6*
	0,1	1,1	3,1	4,1	5,1	7,1
	1,2	0,2	2,2	3,2	4,2	6,2
	3,3	2,3	0,3	1,3	3,3	4,3
	4,4	3,4	1,4	0,4	2,4	3,4
	5,5	4,5	3,5	2,5	0,5	2,5
	7,6	6,6	4,6	3,6	2,6	0,6

⇒

matrice_affectation_ordonnée

clients	*1*	*2*	*3*	*4*	*5*	*6*
	0,1	0,2	0,3	0,4	0,5	0,6
	1,2	1,1	1,4	1,3	2,4	2,5
	3,3	2,3	2,2	2,5	2,6	3,4
	4,4	3,4	3,1	3,2	3,3	4,3
	5,5	4,5	3,5	3,6	4,2	6,2
	7,6	6,6	4,6	4,1	5,1	7,1

FIGURE 3.4 Exemple de construction d'une *matrice_affectation_ordonnée*

Cette matrice est construite à partir de la *matrice d'affectation*. On commence par construire et intégrer les objets à la matrice comme l'illustre la figure 3.4.a pour ensuite trier chaque colonne par ordre croissant des coûts d'affectation (figure 3.4.b). Ce tri est fait une seule fois dans toute la recherche tabou, dans la phase d'initialisation.

L'avantage de cette matrice réside dans le fait que les coûts d'affectation étant triés par ordre croissant pour chaque client, pour remettre à jour le *tableau_ouverture*, il suffit de parcourir chaque début de colonne jusqu'au premier site ouvert, et de reporter l'ensemble de ces sites dans le *tableau_ouverture*. En effet, ces sites présents avant le premier site ouvert dans la colonne ont tous un coût d'affectation c_{ij} inférieur à celui du premier site ouvert (ce "premier site ouvert" est logiquement celui qui sert actuellement le client auquel correspond la colonne puisque le tri fait que c'est le site ouvert qui possède le coût d'affectation minimum pour le client correspondant à la colonne). Par conséquent, si l'un d'entre eux ouvre, il est évident que le client correspondant à la colonne changera de site d'affectation pour celui-ci. Le client correspondant à la colonne fera donc partie des clients réaffectés si un de ces sites venait à ouvrir. Par conséquent, grâce à cette *matrice_affectation_ordonnée*, on ne parcourt plus chaque colonne entièrement : simplement les débuts de chaque colonne ont besoin d'être parcourus pour construire le *tableau_ouverture* (sites en gras dans la figure 3.5).

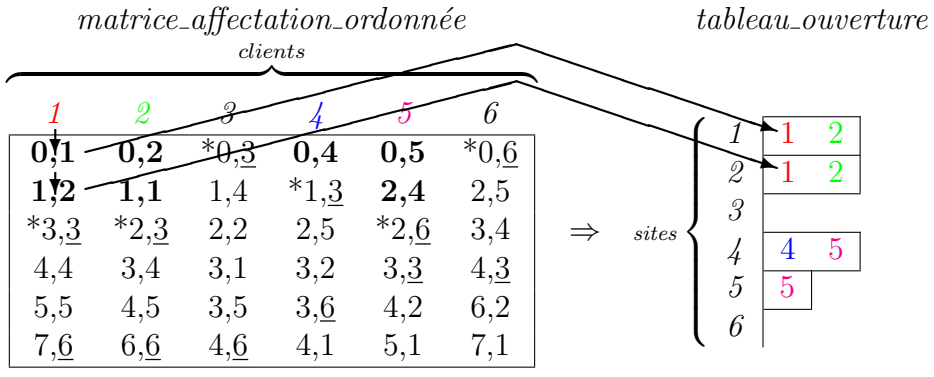


FIGURE 3.5 Exemple de remise à jour d'un *tableau_ouverture*

Cela est illustré dans la figure 3.5 où seuls les sites 3 et 6 sont ouverts dans la solution courante s, et où $*$ désigne le site affecté au client correspondant à la colonne dans la solution courante s ; _ spécifie que le site ouvert.

La phase d'évaluation des mouvements d'ouverture est donc fortement écourtée puisque l'on évite de parcourir pour chaque client l'ensemble des sites au complet.

D'autre part, en ce qui concerne la construction du *tableau_fermeture*, on avait vu que le *tableau_clients* et la *matrice d'affectation* étaient nécessaires pour obtenir les attributs N° *client* et N° *site remplacement* des éléments. Maintenant, grâce cette *matrice_affectation_ordonnée*, au lieu de parcourir entièrement les colonnes correspondant à chaque client réaffecté dans la *matrice d'affectation* à cause de la fermeture d'un site, il suffit de considérer les colonnes de la *matrice_affectation_ordonnée* correspondant à ces clients réaffectés (on trouve ces clients à l'aide du *tableau_clients*), de partir de la case correspondant à ce site qui ferme, et de parcourir les cases suivantes les unes après les autres jusqu'à trouver un site ouvert. Le prochain site ouvert sera le nouveau site affecté au client en question (puisque chaque colonne est triée par coûts d'affectation c_{ij} croissants, ce site sera le site ouvert qui aura le plus petit coût d'affectation c_{ij} pour le client correspondant à la colonne). Cela permet ainsi de ne parcourir là encore qu'une petite partie de l'ensemble des sites. Cette construction du *tableau_fermeture* est illustrée dans la figure 3.6 qui reprend l'exemple précédent où les sites 3 et 6 étaient ouverts. On a donc toujours $*$ qui désigne le site affecté au client correspondant à la colonne dans la solution courante s ; _ qui spécifie que le site est ouvert.

Dans cette figure 3.6, les flèches et les couleurs concernent simplement les opérations faites pour la construction de la $6^{\text{ème}}$ liste du *tableau_fermeture*.

On peut remarquer que le fait d'utiliser la *matrice_affectation_ordonnée* pour constuire le *tableau_fermeture* et le *tableau_ouverture* fait évoluer les temps de calcul différemment en fonction de la taille des instances, mais également en fonction des caractéristiques des coûts d'installation par rapport aux coûts d'affectation. En effet, plus le nombre de sites ouverts est petit dans une solution, plus il est nécessaire d'aller loin dans les colonnes de la *matrice_affectation_ordonnée* pour trouver les sites qui servent les clients et ceux qui doivent les remplacer s'ils ferment puisqu'il y a moins de sites ouverts dans chaque colonne. Par conséquent, plus le nombre de sites

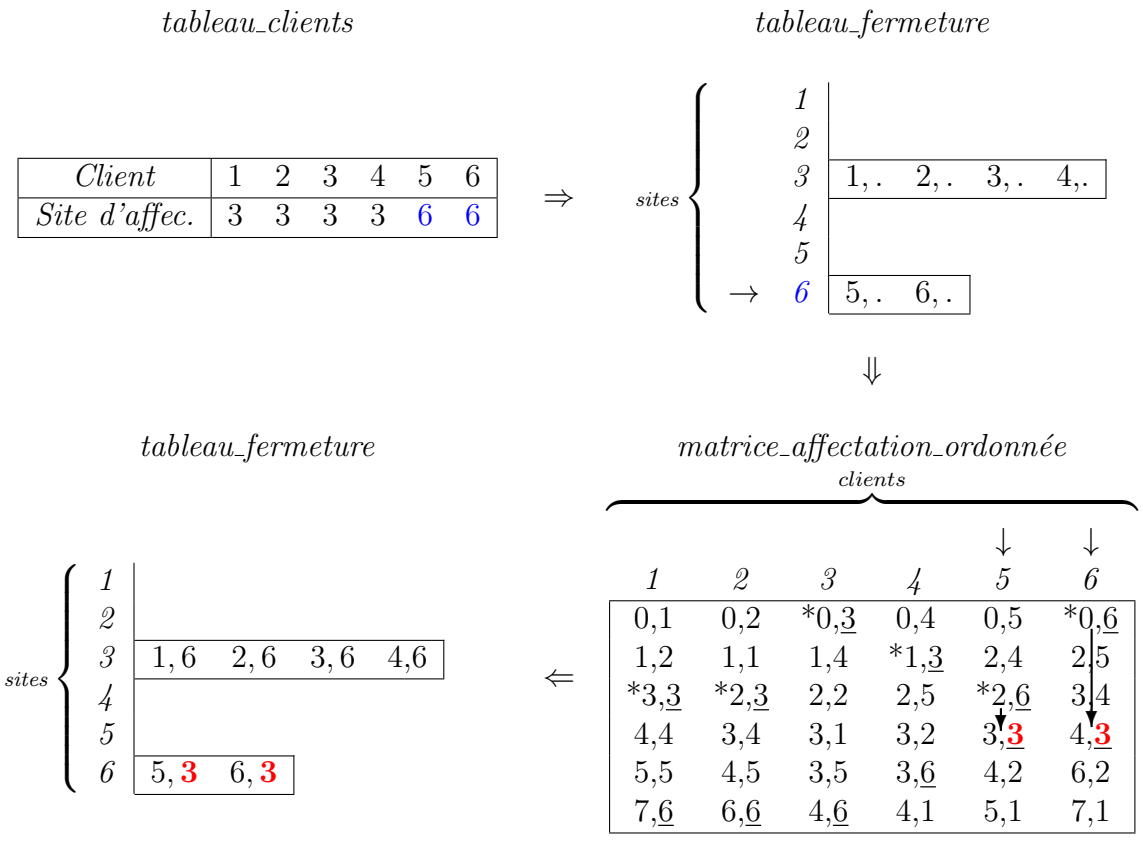


FIGURE 3.6 Exemple de remise à jour d'un *tableau_fermeture*

ouverts est petit, plus les constructions et remises à jour des outils vues ci-dessus voient leur temps augmenter, ce qui engendre ainsi une augmentation du temps de calcul total. Cette augmentation peut être très importante dans certains cas comme par exemple lorsque les coûts d'installation sont très grands comparativement aux coûts d'affectation.

Algorithme d'évaluation du voisinage

Avec le calcul incrémental et l'ensemble des outils vus précédemment, la procédure d'évaluation du voisinage d'une solution s peut-être modélisée selon le pseudo-code de l'algorithme 3.

3 - **Algorithme d'évaluation d'un voisinage**

INITIALISATION

1 : $min := +\infty$ et solution s.

ÉVALUATION DES MOUVEMENTS

2 : **Pour** $p = 1$ à n, **poser** $\Delta z = \begin{cases} f_p + \sum\limits_{j \in F_{s,p}} c_{pj} - c_{e_{s,j}j} & \text{si } y_p = 0 \\ -f_p + \sum\limits_{j \in F_{s,p}} c_{e_{s',j}j} - c_{pj} & \text{si } y_p = 1 \end{cases}$

3 : et $z(s') := z(s) + \Delta z$;

4 : **Si** ($z(s') < min$ et $s' \notin T$) ou $z(s') < z(s^*)$ **alors** $p_{min} := p$ et $min := z(s')$

5 : **Fin pour**

MEILLEUR MOUVEMENT

6 : **Retourner** p_{min}

Les notations de cet algorithme sont les suivantes :

T	est la liste taboue ;
s	est la solution dont on veut évaluer le voisinage ;
s'	est la solution obtenue après application sur la solution s du mouvement p ;
min	est une variable pour retenir la valeur du meilleur mouvement trouvé depuis le début de l'évaluation du voisinage de la solution s
p_{min}	est le meilleur mouvement trouvé depuis le début de la procédure d'évaluation du voisinage (c'est aussi le numéro du site correspondant à ce meilleur mouvement) ;
s^*	est la meilleure solution trouvée depuis le début de la recherche tabou ;
$F_{s,p}$	est l'ensemble des clients réaffectés dans la solution s si le mouvement sur le site p est fait ;
$e_{s,j}$	est l'entrepôt (ou site) auquel le client j est affecté dans la solution s.

La ligne 2 calcule l'écart Δz des mouvements. Si $y_p = 0$, c'est-à-dire que le mouvement p est une ouverture, l'ensemble $F_{s,p}$ est la liste contenue dans la $p^{ème}$ case du *tableau_ouverture* qui contient la liste des clients j réaffectés si le mouvement p est effectué. Les sites $e_{s,j}$ correspondants sont obtenus directement avec le *tableau_clients*.

Si $y_i = 1$, c'est-à-dire que le mouvement p est une fermeture, l'ensemble $F_{s,p}$ est la liste obtenue en prenant l'attribut *N°client* de chaque objet contenu dans la liste située dans la $p^{ème}$ case du *tableau_fermeture*. Les sites $e_{s',j}$ sont obtenus en prenant l'attribut *N° site remplacement* correspondant de ces mêmes objets. La ligne 3, quand à elle, permet de retrouver la valeur de la fonction objectif z pour la solution s' engendrée par le mouvement p sur la solution s. Enfin, la ligne 4 sert à retenir le mouvement p si celui-ci est meilleur et qu'il respecte le critère tabou ou s'il est sujet au critère d'aspiration. Le meilleur mouvement p_{min} est ensuite retourné à la ligne 6.

Pour terminer une itération de recherche tabou, après avoir évalué le voisinage et obtenu le meilleur mouvement p_{min} selon l'algorithme 3, la mise à jour des solutions s et s^* ainsi que celle des outils sont initiées. À la fin, une nouvelle itération peut être commencée et ainsi de suite jusqu'à ce que le nombre d'itérations *NbrIt* soit atteint. Cela termine ainsi la recherche tabou.

On peut remarquer que lorsque plusieurs "meilleurs mouvements" sont égaux lors de l'exécution de l'algorithme 3, le premier trouvé est choisi. En effet, nous avons essayé de mettre en mémoire l'ensemble des meilleurs mouvements trouvés lors de l'évaluation d'un voisinage et de faire le choix aléatoirement, mais cela augmentait de manière non négligeable les temps de calcul tout en ayant très peu d'impact sur la solution finale. Cette recherche tabou est donc totalement déterministe, ce qui n'est pas dérangeant pour ce que l'on veut en faire.

Complexité révisitée

Grâce à ce calcul incrémental, la complexité globale de la recherche tabou devient réduite. En effet, l'algorithme d'évaluation du voisinage a une complexité de l'ordre de $O(nm)$, sachant qu'en pratique, avec le système de tableaux de liste, on n'est censé traiter nettement moins que m clients et n sites puisque l'on ne traite que ce qui est modifié. Pour estimer la complexité globale de cette méthode, il faut tenir compte de la mise à jour de tous les outils vus précédemment et de leur parcours. On a donc :

- le *tableau_clients* qui est remis à jour en $O(m)$ à chaque itération de recherche tabou,
- le *tableau_ouverture* qui est remis à jour en $O(nm)$ à chaque itération de recherche tabou,
- le *tableau_fermeture* qui est également remis à jour en $O(nm)$ à chaque itération de recherche tabou,

- la *matrice_affectation_ordonnée* qui est construite puis triée une seule fois au début de la recherche tabou avec un tri qui se fait en $O(n\ log(n))$,
- le parcours du *tableau_ouverture* et du *tableau_fermeture* qui se fait en $O(nm)$ et qui est fait à chaque chaque itération de recherche tabou.

Ainsi, la complexité globale de la recherche tabou proposée avec le calcul incrémental est en $O(NbrIt\, nm)$ ce qui réduit d'un facteur n la complexité en $O(NbrIt\, n^2 m)$ d'une recherche tabou évaluant les éléments du voisinage selon la méthode vue à la page 32.

Taille de la liste tabou

Pour des raisons d'efficacité, nous avons pris soin d'adopter une taille de liste taboue variable, qui évolue pendant la recherche. L'évolution se fait dans un intervalle $[a, b]$ de la manière suivante : la liste commence avec une taille initiale a, et si la recherche tabou n'a pas amélioré la meilleure solution trouvée s^* pendant un nombre K d'itérations, sa taille est augmentée de 1. Quand celle-ci atteint la taille b, elle est remise à sa taille initiale a. À chaque amélioration de la meilleure solution s^*, la taille de la liste est remise à sa taille initiale a afin d'explorer plus profondément la région de l'espace des solutions. Cela permet d'intensifier la recherche autour de la solution trouvée. En faisant cela, nous avons constaté un gain d'efficacité assez important.

Structure permettant de faire varier la fonction d'évaluation

Toujours pour gagner en performance, nous avons ajouté à cette recherche tabou une structure permettant de faire varier la fonction d'évaluation z. En effet, pour favoriser la diversification, selon la fonction d'évaluation active, les éléments du voisinage vont voir leur valeur changer. Nous avons mis en place trois fonctions d'évaluation différentes :

- la fonction z_1 qui est la fonction d'évaluation classique : la valeur des mouvements est l'écart Δz entre la valeur de la solution s et la valeur de la solution s' découlant du mouvement, additionné à la valeur de la solution s : $z_1(s') = z(s) + \Delta z$;

– le fonction d'évaluation z_2, un peu différente de la précédente, évalue les éléments du voisinage de la manière suivante :

$$z_2(s') = \begin{cases} z(s) + (\Delta z - f_i)/1000 + 100\ f_i & \text{si le mouvement sur le site } i \text{ est une ouverture} \\ z(s) + (\Delta z + f_i)/1000 - 100\ f_i & \text{si le mouvement sur le site } i \text{ est une fermeture}\,; \end{cases}$$

– la fonction d'évaluation z_3 quand à elle retranche le coût d'implantation associé au mouvement :

$$z_3(s') = \begin{cases} z(s) + \Delta z - f_i & \text{si le mouvement est l'ouverture du site } i \\ z(s) + \Delta z + f_i & \text{si le mouvement est la fermeture du site } i. \end{cases}$$

Les fonctions d'évaluation z_2 et z_3 sont utiles en particulier pour les instances possédant des coûts d'installation f_i très grands par rapport aux coûts d'affectation c_{ij}, ou quand la densité δ de la matrice d'affectation est faible. En effet, dans ces cas, la fonction z_2 permet de se concentrer sur les coûts d'installation plutôt que sur le coût total puisque l'écart sur les coûts d'affectation $\Delta z - f_i$ est divisé par 1000 et l'écart sur le coûts d'installation est multiplié par 100. La fonction d'évaluation z_3 permet au contraire de se concentrer sur les coûts d'affectation puisque l'effet du coût d'implantantion du mouvement est annulé.

Lors de l'exécution de la recherche tabou, 25 itérations utilisant z_2 suivies de 25 itérations utilisant z_3 sont effectuées toutes les 300 itérations utilisant z_1. Le nombre d'itérations utilisant z_2 et z_3 est volontairement faible devant celui utilisant z_1 car ces fonctions d'évaluation z_2 et z_3 modifient très rapidement la solution. Par exemple, la fonction z_2 fait généralement choisir des fermetures de sites, donc trop d'itérations de z_2 peut aboutir à fermer l'ensemble des sites de la solution. Ainsi, si on a une proportion trop importante d'itérations utilisant z_2 et z_3, on s'éloigne trop rapidement de la solution d'origine, et on perd les informations accumulées aux cours de la recherche, ce qui donne lieu à des solutions de moindre qualité.

4 - **Recherche tabou implémentée**

1 : Calculer une solution intiale $s \in S$ avec l'algorithme 1 ;
2 : Poser $T := \emptyset$, $s^* := s$, $t(T) := a$, $u := 0$, $v := 1$, $w := 0$, $x := 0$;
3 : **Tant que** $u < NbrIt$ **faire**
4 : Déterminer le meilleur mouvement p_{min} de $N^T(s)$ avec l'algorithme 3 en prenant $z = z_v$;
5 : Obtenir s' en effectuant le mouvement p_{min} ;
6 : **Si** $z_1(s') < z_1(s^*)$ **alors** poser $s^* := s'$, $t(T) := a$ et $w := 0$;
7 : Poser $s := s'$ et mettre à jour T ;
8 : **Si** $w = K$ **alors**
9 : **Si** $t(T) = b$ **alors** poser $t(T) := a$ et $w := 0$;
10 : **Sinon** poser $t(T) := t(T) + 1$ et $w := 0$;
11 : **Fin si**
12 : Poser $v = \begin{cases} 1 & \text{si } x \in [0, 300[\\ 2 & \text{si } x \in [300, 325[\\ 3 & \text{si } x \in [325, 350[\end{cases}$
13 : **Si** $x = 350$ **alors** poser $x := 0$;
14 : **Sinon** poser $x := x + 1$;
15 : Poser $u := u + 1$;
16 : **Fin tant que**
17 : **Retourner** s^*.

La recherche tabou telle que nous l'avons implémentée suit ainsi le nouvel algorithme 4, où

T est la liste taboue ;
$N^T(s) = \{s' \in N(s)$ tel que $s' \notin T$ ou $z(s') < z(s^*)\}$;
s est la solution courante ;
s' est une solution du voisinage $N^T(s)$;
s^* est la meilleure solution trouvée ;
$t : T \mapsto t(T)$ est la fonction qui associe à une liste taboue T sa taille $t(T)$.

Dans cet algorithme 4, la ligne 2 est l'étape d'initialisation, les lignes 4 à 7 effectuent les itérations de recherche tabou avec la fonction d'évaluation et la taille de la liste taboue spécifiée, les lignes 8 à 11 servent à remettre à jour la taille de la liste tabou $t(T)$, et enfin, les lignes 12 à 14 remettent à jour le numéro v de la fonction d'évaluation z_v qui sera utilisée dans l'itération suivante.

On pourra remarquer que la liste taboue n'est pas remise à sa taille initiale lors d'un changement de fonction d'évaluation : en effet, nous avons essayé les deux possibilités, et les tests nous ont montré que la conservation de la liste taboue lors du passage d'une fonction d'évaluation à l'autre permettait d'obtenir de meilleurs résultats.

Enfin, avant de passer à la section suivante, nous préciserons que le nombre d'itérations fait avec chaque fonction d'évaluation et les coefficients 1/1000 et 100 utilisés dans la fonction d'évaluation z_2 sont des paramètres que nous avons directement fixés pour éviter d'alourdir le mémoire avec des notations supplémentaires. En effet, les essais que nous avons effectués au préalable ont montré que ces valeurs fonctionnent bien pour l'ensemble des tests réalisés dans le cadre de ce mémoire.

3.2 Résultats

3.2.1 Présentation des instances

Afin de préciser si l'algorithme glouton et la recherche tabou sont suffisamment efficaces et rapides pour obtenir des résultats concluants, il est nécessaire de faire un certain nombre de tests. Ces tests permettent d'une part d'apprécier l'efficacité du programme, mais surtout d'analyser son comportement selon différents types d'instances du problème LESC. Tous les tests ont été réalisés sur un PC Dell Studio muni d'un processeur Intel(R) Core(TM) Quad CPU Q9550 à 2,83 GHz et de 3,9 Go de mémoire RAM, le tout fonctionnant sous le système d'exploitation Linux openSUZE 11.1 (x86_64) KDE 4.1.3 "realease 4.9". Le langage de programmation JAVA a été utilisé pour la programmation de la métaheuristique.

Les instances utilisées sont des instances trouvées dans une bibliothèque d'instances pour le problème LESC proposée par Hoefer (2004) de l'institut d'informatique Max Planck (Allemagne). Plusieurs familles d'instances sont disponibles et constituent la base de nos tests. Nous avons utilisé les familles suivantes :

- Euclidienne : l'ensemble des clients est le même que l'ensemble des sites ($n = m$). Dans un carré imaginaire de 7 000 unités de côté, n points sont placés aléatoirement. Chaque point est à la fois un client et un site ; les coûts d'affectation sont les distances euclidiennes entre chacun des points. La matrice d'affectation est donc symétrique, sa diagonale est nulle et l'inégalité triangulaire est toujours respectée. Les coûts d'installation sont égaux à 3000 pour

chaque site. Les instances sont de taille (100×100) et sont toutes résolues à l'optimalité.

– Galvão-Raggi : les ensembles des sites et des clients sont les mêmes ($n = m$). Les coûts d'installation sont choisis aléatoirement selon une loi normale de moyenne μ et d'écart-type σ. Les coûts d'affectation sont pris de manière à ce que le problème représente un graphe biparti (l'ensemble 1 est l'ensemble des sites et l'ensemble 2 est l'ensemble des clients) dont la densité d des arêtes est variable selon les instances. Cette densité d, différente de la densité δ que nous avons définie dans le chapitre précédent, est calculée de la manière suivante :

$$d = N(G(V, E))/N(K_n)$$

où $N(G(V, E))$ est le nombre d'arêtes dans le graphe $G(V, E)$ et $N(K_n)$ le nombre d'arêtes dans le graphe complet correspondant. Les coûts sur les arêtes sont choisis aléatoirement selon une loi uniforme sur un intervalle choisi ($[1, n]$ en général). La matrice d'affectation est donc une matrice dont les valeurs $c_{i,j}$ représentent en fait la valeur du plus court chemin du site i au client j. Lorsqu'il n'existe pas de chemin entre le site i et le client j, la valeur est égale à $+\infty$ (c'est-à-dire un nombre très grand par rapport aux autres coûts d'affectation et aux coûts d'installation). La matrice est non symétrique et sa diagonale est non nulle. Dans les instances, il existe des sites ayant un coût d'installation nul pour symboliser le fait que ces sites sont déjà ouverts. Cela permet de simuler des cas où par exemple on aurait un réseau d'usines déjà ouvertes et où l'on aimerait savoir quels sites devraient être affectés à quels nouveaux clients et quels nouveaux sites il serait intéressant d'installer. Les instances sont de taille moyenne à grande (entre 50×50 et 200×200) et sont toutes résolues à l'optimalité.

– Koerkel-Ghosh symétrique : $n = m$ mais les ensembles des sites et des clients sont différents. Les coûts d'installation et les coûts d'affectation sont choisis aléatoirement selon une loi uniforme sur un certain intervalle. La matrice d'affectation est symétrique mais sa diagonale est non nulle. Les instances données sont de grande taille : entre 250×250 et 750×750. Une seule instance de taille 250×250 est résolue à l'optimalité. Pour les autres instances, seules des solutions faisant office de borne sont données.

- Koerkel-Ghosh asymétrique : idem à la famille Koerkel-Ghosh symétrique sauf que la matrice d'affectation n'est pas symétrique. Comme précédemment, les instances données sont de grande taille : entre 250×250 et 750×750, et une seule instance de taille 250×250 est résolue à l'optimalité. Pour les autres instances, seules des bornes sont données.
- Uniforme : $n = m$ mais les ensembles des sites et des clients sont différents. Les coûts d'installation sont les mêmes pour tous les sites et les coûts d'affectation sont choisis aléatoirement selon une loi uniforme sur un certain intervalle. La matrice d'affectation n'est pas symétrique et possède une diagonale non nulle. Les instances sont de taille moyenne (100×100) et sont toutes résolues à l'optimalité.
- Grand saut de dualité : $n = m$ mais les ensembles des sites et des clients sont différents. Les coûts d'installation des sites sont les mêmes pour tous les sites. Les coûts d'affectation sont soit très grands (suffisamment grands pour représenter l'infini), soit très petit (compris entre 0 et 4). La matrice contient donc majoritairement des termes très grands et quelques termes très petits. Cela crée de grands "sauts de dualité", ce qui rend difficile le déroulement de l'algorithme d'évaluation et séparation d'après Kochetov et Ivanenko (2003). Ces instances de taille 100×100 sont répertoriées en trois familles : la classe A qui possède 10 petits coûts par client (c'est-à-dire 10 petits coûts par colonne de la matrice d'affectation), la classe B qui possède 10 petits coûts par site (c'est-à-dire 10 petits coûts par ligne de la matrice d'affectation) et la classe C qui possède 10 petits coûts par client et par site (c'est-à-dire 10 petits coûts par colonne et par ligne de la matrice d'affectation). La matrice d'affectation est non symétrique et sa diagonale non nulle. Les instances sont de taille moyenne (100×100) et sont toutes résolues à l'optimalité.

Pour chacune des instances de ces familles, une solution est disponible. Pour une partie d'entre elles, l'optimalité est garantie, et pour d'autres, elle ne l'est pas. Ces instances et leurs solutions nous permettront ainsi d'évaluer l'efficacité de notre métaheuristique.

On peut remarquer que, pour chacune des familles d'instances testées, on a $n = m$. Il semble que ce format soit devenu une tendance dans les bibliothèques d'instances. Dans la bibliothèque d'instances utilisée, seules quelques instances sont de format

$n \neq m$, cependant, leurs tailles trop petites les rendent peu intéressantes ($n \leq 50$ et $m \leq 100$). Nous les avons donc volontairement mises de côté.

3.2.2 Paramètres de la recherche tabou

Pour les tests de la sous-section suivante, les paramètres de la recherche tabou ont été choisis comme suit :

$a = 5$

$$b = \begin{cases} 15 & \text{pour les instances “Grand saut de dualité”} \\ 25 & \text{pour les autres instances} \end{cases}$$

$K = 600$

$NbrIt = 100000$.

Le changement du paramètre b pour les instances “Grand saut de dualité” permet d'intensifier la diversification. En effet, ces instances ont des caractéristiques telles que l'accroissement de la diversification produit une amélioration non négligeable dans les valeurs des solutions. Ces caractéristiques seront discutées plus en détail à la fin de la section.

En ce qui concerne le nombre d'itérations $NbrIt$ choisi égale à 100000, cela constitue un bon compromis entre qualité des solutions et temps de calcul. Il est évident que si ce nombre est augmenté, certaines instances peuvent voir leur solution s'améliorer.

3.2.3 Résultats

Nous donnons dans les tableaux 3.4 et 3.5 les résultats trouvés lors de nos tests. L'erreur ε est définie de la manière suivante :

$$\varepsilon = \frac{z^* - z_{UFLlib}}{z_{UFLlib}}$$

où z^* est la valeur de la solution trouvée ;
z_{UFLlib} est la valeur de la solution donnée par la librairie UFLlib.

$\bar{\varepsilon}$ est la moyenne des erreurs ε sur l'ensemble des instances de la famille considérée. Dans les tableaux 3.4 et 3.5, seule cette erreur moyenne est donnée.

Famille	*Nb inst.*	$n \times m$	$\bar{\varepsilon}_{Glouton}$	σ	*Tps (s)*
Euclidienne	30	100×100	4,455%	1,441%	$< 0,1$
Galvão-Raggi	10	50×50	1,266%	1,432%	$< 0,1$
Galvão-Raggi	10	70×70	1,148%	1,589%	$< 0,1$
Galvão-Raggi	10	100×100	1,080%	1,300%	$< 0,1$
Galvão-Raggi	10	150×150	0,164%	0,350%	$< 0,1$
Galvão-Raggi	10	200×200	0,984%	0,903%	$< 0,1$
Koerkel-Ghosh sym A	5	250×250	0,220%	0,061%	$< 0,1$
Koerkel-Ghosh sym B	5	250×250	0,528%	0,125%	$< 0,1$
Koerkel-Ghosh sym C	5	250×250	0,344%	0,255%	$< 0,1$
Koerkel-Ghosh sym A	5	500×500	0,207%	0,039%	$< 0,1$
Koerkel-Ghosh sym B	5	500×500	0,589%	0,270%	$< 0,1$
Koerkel-Ghosh sym C	5	500×500	0,407%	0,306%	$< 0,1$
Koerkel-Ghosh sym A	5	750×750	0,188%	0,027%	$< 0,1$
Koerkel-Ghosh sym B	5	750×750	0,513%	0,185%	$< 0,1$
Koerkel-Ghosh sym C	5	750×750	0,443%	0,303%	$< 0,1$
Koerkel-Ghosh asym A	5	250×250	0,209%	0,014%	$< 0,1$
Koerkel-Ghosh asym B	5	250×250	0,662%	0,325%	$< 0,1$
Koerkel-Ghosh asym C	5	250×250	0,491%	0,549%	$< 0,1$
Koerkel-Ghosh asym A	5	500×500	0,189%	0,053%	$< 0,1$
Koerkel-Ghosh asym B	5	500×500	0,521%	0,242%	$< 0,1$
Koerkel-Ghosh asym C	5	500×500	0,187%	0,164%	$< 0,1$
Koerkel-Ghosh asym A	5	750×750	0,171%	0,040%	$< 0,1$
Koerkel-Ghosh asym B	5	750×750	0,453%	0,083%	$< 0,1$
Koerkel-Ghosh asym C	5	750×750	0,363%	0,174%	$< 0,1$
Uniforme	30	100×100	6,052%	2,499%	$< 0,1$
G. S. Dualité A	30	100×100	9,335%	4,734%	$< 0,1$
G. S. Dualité B	30	100×100	18,887%	6,672%	$< 0,1$
G. S. Dualité C	30	100×100	12,820%	4,720%	$< 0,1$

TABLEAU 3.4 Résultats de l'algorithme glouton

Famille	*Nb inst.*	$n \times m$	$\bar{\varepsilon}_{Tabou}$	σ	*Tps (s)*
Euclidienne	30	100×100	0,371%	0,228%	3,9
Galvão-Raggi	10	50×50	0,000%	0,000%	1,2
Galvão-Raggi	10	70×70	0,000%	0,000%	1,6
Galvão-Raggi	10	100×100	0,000%	0,000%	2,2
Galvão-Raggi	10	150×150	0,015%	0,049%	3,1
Galvão-Raggi	10	200×200	0,020%	0,041%	4,8
Koerkel-Ghosh sym A	5	250×250	-0,007%	0,007%	11,0
Koerkel-Ghosh sym B	5	250×250	-0,003%	0,007%	26,0
Koerkel-Ghosh sym C	5	250×250	0,000%	0,000%	59,9
Koerkel-Ghosh sym A	5	500×500	-0,008%	0,005%	29,7
Koerkel-Ghosh sym B	5	500×500	-0,006%	0,013%	85,1
Koerkel-Ghosh sym C	5	500×500	0,000%	0,000%	288,6
Koerkel-Ghosh sym A	5	750×750	-0,010%	0,004%	54,2
Koerkel-Ghosh sym B	5	750×750	-0,007%	0,005%	204,4
Koerkel-Ghosh sym C	5	750×750	0,000%	0,000%	1039,8
Koerkel-Ghosh asym A	5	250×250	-0,007%	0,007%	10,8
Koerkel-Ghosh asym B	5	250×250	0,000%	0,000%	25,6
Koerkel-Ghosh asym C	5	250×250	0,000%	0,000%	60,2
Koerkel-Ghosh asym A	5	500×500	-0,002%	0,004%	30,0
Koerkel-Ghosh asym B	5	500×500	-0,001%	0,002%	82,9
Koerkel-Ghosh asym C	5	500×500	0,000%	0,000%	289,5
Koerkel-Ghosh asym A	5	750×750	-0,002%	0,004%	55,4
Koerkel-Ghosh asym B	5	750×750	-0,001%	0,001%	208,1
Koerkel-Ghosh asym C	5	750×750	-0,018%	0,029%	1058,3
Uniforme	30	100×100	0,000%	0,000%	4,1
G. S. Dualité A	30	100×100	3,575%	4,132%	3,7
G. S. Dualité B	30	100×100	6,661%	4,708%	3,5
G. S. Dualité C	30	100×100	4,535%	3,924%	3,7

TABLEAU 3.5 Résultats de la recherche tabou

On pourra remarquer que pour certaines familles d'instances, la valeur de l'erreur moyenne $\bar{\varepsilon}$ est négative. Ce cas arrive lorsque les solutions données par la librairie UFLlib ne sont pas optimales et que notre métaheuristique aboutit à de meilleures bornes. D'autre part, les tableaux 3.4 et 3.5 ont également une colonne σ qui représente l'écart-type sur l'ensemble des instances de la famille considérée, ainsi qu'une colonne *Tps* qui donne la moyenne des temps de calcul, et une colonne *Nb inst.* qui spécifie le nombre d'instances contenu dans la famille. Les résultats pour l'algorithme glouton sont présentés dans le tableau 3.4 alors que ceux pour la recherche tabou sont présentés dans le tableau 3.5.

L'algorithme glouton donne des résultats assez variables avec des erreurs moyennes allant de 0,1% à 18,9% par rapport aux valeurs des solutions de la librairie UFLlib. L'erreur maximale de 34,5% est obtenue pour une instance de la classe G. S. Dualité B. Les temps de calcul sont très rapides mais la moyenne des erreurs est relativement élevée avec une moyenne des moyennes $\bar{\varepsilon}_{Glouton}$ à 2,25%.

Quant à la recherche tabou, elle semble performante de manière générale, avec une moyenne des moyennes des erreurs $\bar{\varepsilon}_{Tabou}$ à 0,54%. Pour certaines familles d'instances, elle s'avère particulièrement efficace puisqu'elle arrive à donner de meilleures bornes que ce que donne la librairie UFLlib. C'est le cas pour la majeure partie des instances de type Koerkel-Ghosh symétrique et Koerkel-Ghosh asymétrique. Seules les instances de type Grand saut de dualité débouchent sur des erreurs assez importantes. Nous en reparlerons dans la sous-section suivante.

Pour les temps de calcul, ils sont plutôt faibles pour les instances de petite taille (100×100) et augmentent évidemment avec la taille des instances traitées. Nous pouvons observer, sur les instances Koerkel-Ghosh symétriques et asymétriques, l'effet dont nous avions parlé dans la section précédente, à savoir que quand les coûts d'installation augmentent par rapport aux coûts d'affectation, le temps de calcul augmente. C'est la raison pour laquelle les temps de calcul de la catégorie A sont toujours plus élevés que ceux de la catégorie B et ceux de la catégorie B plus élevés que ceux de la catégorie C bien que la taille des instances soit la même.

3.2.4 Cas particulier des instances Grand saut de dualité

Comme on peut le voir dans les tableaux de la sous-section précédente, les instances Grand saut de dualité ont des erreurs moyennes bien plus importantes que les

autres familles d'instances (moyennes des erreurs $\bar{\varepsilon}_{Tabou}$ variant entre 3,57% et 6,67% pour les instances Grand saut de dualité contre $-0,01\%$ et 0,38% pour les autres familles d'instances). Il faut cependant souligner un point important : les coûts d'installation sont tellement grands comparativement aux coûts d'affectation (ceux qui ne sont pas "infinis") et le nombre de sites d'affectation possibles pour chaque client est tellement faible (de l'ordre de 1/10 à cause de la densité $\delta = 10\%$) que le problème change légèrement de philosophie. Le but devient finalement de trouver la configuration permettant de servir tous les clients en ouvrant le moins de sites possibles.

Ainsi, en ce qui concerne l'erreur ε, comme les coûts d'installation sont tous égaux, chaque site ouvert en moins diminue la valeur de la solution trouvée d'un coefficient à peu près égal à $1/NbSiteOuv_{opt}$ où $NbSiteOuv_{opt}$ est le nombre de sites ouverts dans la solution optimale. Par exemple, si la solution optimale d'une instance a 14 sites ouverts comme c'est souvent le cas, chaque site supplémentaire dans une solution amène à $1/14 \approx 7\%$ de coûts d'installation en plus, c'est-à-dire une augmentation de l'ordre de 7% du coût total (puisque les coûts d'installation sont très faibles). On a donc la formule suivante, où z_{opt} représente la valeur de la solution optimale :

$$\begin{aligned} \varepsilon &\approx \frac{(z_{opt} + z_{opt}/NbSiteOuv_{opt}) - z_{opt}}{z_{opt}} \\ &\approx 1/NbSiteOuv_{opt} = 1/14 \approx 7\%. \end{aligned}$$

Par conséquent, comme les solutions avec le nombre optimal de sites ouverts et n'utilisant pas de coûts d'affectation "infinis" sont très peu nombreuses, si la métaheuristique n'en trouve pas, la solution se retrouve nécessairement avec une erreur de l'ordre de $1/NbSiteOuv_{opt}$ ou plus. Cela explique pourquoi la recherche tabou a des erreurs moyennes de quelques pourcents : soit elle trouve la solution optimale ou extrêmement proche (erreur $\varepsilon_{Tabou} \leq 0,1\%$), soit elle trouve une solution à un ou deux sites de plus et donc son erreur est propulsée à $1/NbSiteOuv_{opt}$ ou $2/NbSiteOuv_{opt}$ par rapport à la solution optimale.

D'ailleurs, ces caratéristiques donnent lieu à des espaces de solution très "saccadés" avec des "puits d'attraction" très profonds et très étroits, ce qui rend les choses difficiles pour une métaheuristique comme la recherche tabou qui a du mal à trouver les bons puits d'attraction et à en sortir quand ils ne sont pas prometteurs. C'est d'ailleurs pour cette raison que nous avons mis en place la structure permettant de faire varier la fonction d'évaluation z. En effet, elle permet de favoriser la

diversification de la solution grâce aux fonctions d'évaluation z_2 et z_3, en particulier lorsque la recherche tabou a du mal à remonter d'un minimum local. Ces deux fonctions d'évaluation supplémentaires nous ont permis de diviser l'erreur $\bar{\varepsilon}_{Tabou}$ pour les instances Grand saut de dualité d'un facteur 2 approximativement par rapport à la recherche tabou sans ces deux fonctions d'évaluation.

Chapitre 4

La méthode IPS

IPS est une méthode de résolution de programmation linéaire actuellement en développement au GERAD, un laboratoire universitaire commun à l'École Polytechnique de Montréal, HEC Montréal, l'Université du Québec à Montréal et l'Université McGill (Canada).

Cette méthode améliore l'algorithme du simplexe primal en réduisant l'impact de la dégénerescence grâce à l'utilisation du principe d'agrégation et désagrégation de contraintes qui permet de transformer la résolution du problème global en une succession de résolution de problèmes plus petits et donc plus faciles et plus rapides à résoudre. Comme le problème LESC engendre beaucoup de dégénerescence, on veut voir si IPS peut être efficace pour résoudre la relaxation linéaire de ce problème en comparaison à l'algorithme du simplexe primal. Si c'est le cas, IPS sera imbriqué dans un algorithme d'évaluation et séparation pour observer ce que donne la résolution en nombres entiers. Cela n'a encore jamais été fait.

Nous décrivons le fonctionnement d'IPS dans la première section afin de mieux comprendre les différents tests réalisés sur la relaxation linéaire dans la section suivante. Dans la troisième section, les résultats sur la résolution en nombres entiers sont exposés et discutés. Enfin, pour terminer, une analyse et des commentaires sont faits dans la troisième et dernière section.

4.1 Fonctionnement

4.1.1 Problème réduit

IPS a été introduit pour la première fois par Elhallaoui *et al.* (2007) puis repris et amélioré par Raymond *et al.* (2010). C'est une méthode de résolution de programmes linéaires fonctionnant sur le même modèle que l'algorithme du simplexe primal mais en y apportant un certain nombre d'améliorations en vue d'accroître son efficacité là

où l'algorithme du simplexe primal montre certaines faiblesses notamment lorsque le programme linéaire traité donne lieu à une dégénerescence importante.

Commençons par poser les notations de base. Soit PL le programme linéaire sous forme standard suivant :

$$(PL) \qquad z^{PL} = \min_x c^T x \tag{4.1}$$

$$\textit{sujet à} \quad Ax = b \tag{4.2}$$

$$x \geq 0 \tag{4.3}$$

où
- $x \in \mathbb{R}^n$ est le vecteur de variables ;
- $c \in \mathbb{R}^n$ est le vecteur des coefficients des variables dans la fonction objectif ;
- $b \in \mathbb{R}^m$ est le vecteur des membres de droite des contraintes ;
- $A \in \mathbb{R}^{m \times n}$ est la matrice des coefficients des variables du membre de gauche des contraintes ;
- a_{ij} est élément de A.

Définissons ensuite une base $B \in \mathbb{R}^{m \times m}$ formée de colonnes linéairement indépendantes de la matrices A. Elle est réalisable si $B^{-1}b \geq 0$ et fournit une solution de base réalisable $\bar{x} = \begin{pmatrix} B^{-1}b \\ 0 \end{pmatrix}$. Dans $\bar{x}$, on appelle les m premières variables, les variables de bases, et les $n-m$ dernières, les variables hors base. Notons également π le vecteur des variables duales associées aux contraintes (4.2). On a par définition $\pi = c_B B^{-1}$, où $c_B \in \mathbb{R}^m$ est le vecteur obtenu en ne prenant que les coefficients des variables de base dans c.

Supposons ensuite que la solution est dégénérée et comporte $p < m$ variables de base positives indicées de 1 à p, et $m - p$ variables de base nulles indicées de $p + 1$ à m. On a aussi $n - m$ variables hors base égales à 0.

Définissons maintenant les sous-matrices $Q_{P_B} \in \mathbb{R}^{p \times m}$ et $Q_{Z_B} \in \mathbb{R}^{(m-p) \times m}$ telles que $B^{-1} = \begin{pmatrix} Q_{P_B} \\ Q_{Z_B} \end{pmatrix}$. La sous-matrice Q_{P_B} est composée des p premières lignes de B^{-1} correspondant aux variables de base positives ($\bar{x}_j > 0$ pour $j = 1, ..., p$) et sert à définir les contraintes du problème réduit RP_B dont nous reparlerons dans la suite. La sous-matrice Q_{Z_B} quand à elle est composée des $m - p$ dernières lignes de B^{-1} correspondant aux variables de base nulles ($\bar{x}_j = 0$ pour $j = m - p, ..., m$) et sert à

choisir les variables qui prendront place dans ce problème réduit. Ces variables seront les variables dites compatibles avec Q_{Z_B}.

Par définition, il est considéré qu'une colonne A_j de la matrice A est dite compatible à une matrice M si $MA_j = 0$. Elle est dite incompatible si $MA_j \neq 0$. On dit ainsi par abus du langage que la variable correspondante x_j est aussi compatible avec la matrice M. Cette matrice M est appelée matrice de compatibilité.

Avec la décomposition de B^{-1} en Q_{P_B} et Q_{Z_B}, on peut remarquer que les variables de base positives demeurent compatibles avec Q_{Z_B} alors que les variables de base nulles sont incompatibles avec Q_{Z_B}. On peut expliquer cela de la manière suivante : si B_i^{-1} est une ligne de B^{-1} et B_j une colonne de B, comme on a $B^{-1}B = I_m$, le produit de $B_i^{-1}B_j$ est égale à 1 si et seulement si $i = j$, et 0 sinon. Par conséquent, comme les variables de base positives correspondent aux colonnes $j = 1, ..., p$ de la base B et que la matrice Q_{Z_B} est constituée des lignes $i = p + 1, ..., m$ de B^{-1}, on a jamais $i = j$. Donc le produit de Q_{Z_B} avec la colonne de B associée à une variable de base positive est toujours nul ce qui implique la compatibilité avec Q_{Z_B}. À l'inverse, les colonnes de B correspondant aux variables de base nulles sont les colonnes $j = p + 1, ..., m$, donc pour chacune de ces colonnes, il existe toujours une lignes de Q_{Z_B} qui respecte $i = j$. Par conséquent, leur produit avec Q_{Z_B} débouche toujours sur un vecteur non nul ce qui implique l'incompatibilité avec Q_{Z_B}.

D'autre part, on peut aussi remarquer qu'une colonne A_j de la matrice A est compatible avec Q_{Z_B} si et seulement si elle est linéairement dépendante avec les p premières colonnes A_l de la base B, où $l = 1, ..., p$ ($A_l = B_l$ pour reprendre les notations du paragraphe précédent). En effet :

$$
\begin{array}{lll}
 & A_j \text{ est dépendante des colonnes } A_1, ..., A_p & \\
\Rightarrow & \exists(\alpha_1, ..., \alpha_p) \in \mathbb{R}^p \text{ tels que} \quad A_j = \alpha_1 A_1 + ... + \alpha_p A_p & \\
\Rightarrow & Q_{Z_B} A_j = \alpha_1 Q_{Z_B} A_1 + ... + \alpha_p Q_{Z_B} A_p & \\
\Rightarrow & Q_{Z_B} A_j = 0 \quad \text{puisque } A_1, ..., A_p \text{ sont des} & \\
 & \qquad\qquad\qquad\qquad \text{colonnes compatibles avec } Q_{Z_B} & \\
\Rightarrow & A_j \text{ est compatible avec } Q_{Z_B} &
\end{array}
$$

Dans l'autre sens, on a :

A_j est compatible avec Q_{Z_B}

$$\Rightarrow \qquad Q_{Z_B} A_j = 0 \qquad \text{donc} \qquad B^{-1} A_j = \begin{pmatrix} Q_{P_B} \\ Q_{Z_B} \end{pmatrix} A_j = \begin{pmatrix} Q_{P_B} A_j \\ 0 \end{pmatrix}$$

or $\quad Q_{P_B}(A_1 \ ... \ A_p) = I_p \quad$ puisque $\quad B^{-1}B = I_m$

$$\Leftrightarrow \quad \begin{pmatrix} Q_{P_B} \\ Q_{Z_B} \end{pmatrix} (A_1 \ ... \ A_p \ ... \ A_m) = \begin{pmatrix} I_p & 0 \\ 0 & I_{m-p} \end{pmatrix}$$

$$\text{donc } B^{-1}(A_1 \ ... \ A_p) = \begin{pmatrix} Q_{P_B} \\ Q_{Z_B} \end{pmatrix} (A_1 \ ... \ A_p) = \begin{pmatrix} Q_{P_B}(A_1 \ ... \ A_p) \\ Q_{Z_B}(A_1 \ ... \ A_p) \end{pmatrix} = \begin{pmatrix} I_p \\ 0 \end{pmatrix}$$

puisque $\quad \forall \ l \in 1, ..., p, \ A_l$ est compatible avec Q_{Z_B}

Par conséquent, si $B^{-1}A_j \neq 0$, alors $\exists \ (\alpha_1, ... \ , \alpha_p) \in \mathbb{R}^p$ tel que :

$B^{-1}(\alpha_1 A_1 + ... + \alpha_p A_p) = B^{-1}A_j \quad \Rightarrow \quad BB^{-1}(\alpha_1 A_1 + ... + \alpha_p A_p) = BB^{-1}A_j$

$\Rightarrow \quad \alpha_1 A_1 + ... + \alpha_p A_p = A_j \quad$ donc $\quad A_j$ est dépendante des colonnes $A_1, ..., A_p$.

Afin d'aboutir au problème réduit, ajoutons encore les notations suivantes :

x_{C_B}	le vecteur des variables compatibles (avec Q_{Z_B}) ;
c_{C_B}	son vecteur coût associé ;
x_{I_B}	le vecteur des variables incompatibles (avec Q_{Z_B}) ;
c_{I_B}	son vecteur coût associé ;
A_{C_B}	la sous-matrice de A telle que ses colonnes sont compatibles avec Q_{Z_B} ;
A_{I_B}	la sous-matrice de A telle que ses colonnes sont incompatibles avec Q_{Z_B} ;
A_{C_B} et A_{I_B}	sont deux matrices telles que $A = (A_{C_B} \ A_{I_B})$.

On peut ainsi tranformer les contraintes (4.2) de la manière suivante :

$$\begin{aligned} AX = b \Leftrightarrow & (A_{C_B}\ A_{I_B}) \begin{pmatrix} x_{C_B} \\ x_{I_B} \end{pmatrix} = b \\ \Leftrightarrow & \begin{pmatrix} Q_{P_B} \\ Q_{Z_B} \end{pmatrix} (A_{C_B}\ A_{I_B}) \begin{pmatrix} x_{C_B} \\ x_{I_B} \end{pmatrix} = \begin{pmatrix} Q_{P_B} \\ Q_{Z_B} \end{pmatrix} b \\ \Leftrightarrow & \begin{cases} Q_{P_B} A_{C_B} x_{C_B} + Q_{P_B} A_{I_B} x_{I_B} = Q_{P_B} b \\ Q_{Z_B} A_{C_B} x_{C_B} + Q_{Z_B} A_{I_B} x_{I_B} = Q_{Z_B} b \end{cases} \end{aligned}$$

Comme les colonnes de A_{C_B} sont compatibles avec Q_{Z_B}, on a $Q_{Z_B} A_{C_B} = 0$. De plus comme $\bar{x}_j = 0$ pour $j = p+1, ..., m$ puisque les variables de la solution $\bar{x}$ ont été triées précédemment pour que seules les p premières variables (qui sont de base) soient non nulles, on a $\bar{x}_j = B_j^{-1} b = 0$ si $j = p+1, ..., m$. Or comme les lignes $j = p+1, ..., m$ de B^{-1} représentent Q_{Z_B}, on a $Q_{Z_B} b = 0$. PL devient donc équivalent à :

$$(PL) \qquad z^{PL} = \min_{x_{C_B}, x_{I_B}} c_{C_B}^T x_{C_B} + c_{I_B}^T x_{I_B} \tag{4.4}$$

$$\textit{sujet à} \quad Q_{P_B} A_{C_B} x_{C_B} + Q_{P_B} A_{I_B} x_{I_B} = Q_{P_B} b \tag{4.5}$$

$$Q_{Z_B} A_{C_B} x_{C_B} = 0 \tag{4.6}$$

$$x_{C_B}, x_{I_B} \geq 0\ . \tag{4.7}$$

On peut ainsi définir le problème réduit PR_B en reprenant le problème PL ci-dessus et en lui enlevant les membres correspondant aux variables incompatibles ainsi que les contraintes (4.6). Cela donne :

$$(PR_B) \qquad z_B^{PR} = \min_{x_{C_B}} c_{C_B}^T x_{C_B} \tag{4.8}$$

$$\textit{sujet à} \quad Q_{P_B} A_{C_B} x_{C_B} = Q_{P_B} b \tag{4.9}$$

$$x_{C_B} \geq 0\ . \tag{4.10}$$

En fait, A_{C_B} est construite en prenant les colonnes compatibles de B (c'est-à-dire les colonnes correspondant aux variables de base positives) ainsi que les autres colonnes de A qui sont compatibles avec Q_{Z_B} (c'est-à-dire les colonnes qui sont linéairement dépendantes des p premières colonnes de B). Quand aux colonnes correspondant aux variables de base nulles, elles sont incorporées à A_{I_B} puisqu'elles sont

incompatibles comme il a été vu précédemment. Ainsi, le problème réduit PR_B n'implique que des variables compatibles, c'est-à-dire uniquement les variables qui peuvent être manipulées dans PR_B sans que les autres contraintes du problème (c'est-à-dire les contraintes (4.6)) soient violées. En fait, tant que ces variables respectent les contraintes de PR_B, elles respecteront les contraintes du problème général PL.

Le cheminement précédent permettant d'aboutir à PR_B est schématisé dans les figures 4.1 et 4.2 où :

$\bar{b}$ est le vecteur $B^{-1}b = \begin{pmatrix} Q_{P_B}b \\ Q_{Z_B}b \end{pmatrix}$ qui représente les valeurs des variables de base dans la solution courante ;

$\bar{A}$ est la matrice $B^{-1}A$;

$+$ est un nombre positif ;

Les *lignes indépendantes* sont les p premières lignes de A. Les p premiers termes de chacune de ces lignes forment une base pour PR_B ;

Les *lignes dépendantes* sont les $m - p$ lignes suivantes de A. Les p premiers termes de chacune de ces lignes sont dépendants des p premiers termes des *lignes indépendantes*.

On pourra remarquer que PR_B ne contient que p contraintes lorsque la base B contient p variables de base positives (ou non dégénérées) et un peu plus de p variables (les variables compatibles de B ainsi que les autres variables compatibles de A). On peut donc en déduire que plus la solution $\bar{x}$ est dégénérée, plus la taille du problème réduit PR_B sera petite et donc plus il sera facile à résoudre.

On pourra aussi noter que si $\hat{x}_{C_B}$ est une solution réalisable pour PR_B, il est facile de la rendre réalisable pour PL : il suffit de compléter le vecteur $\hat{x}_{C_B}$ en lui fixant chaque autre variable à 0, c'est-à-dire en posant $\begin{pmatrix} x_{C_B} \\ x_{I_B} \end{pmatrix} = \begin{pmatrix} \hat{x}_{C_B} \\ 0 \end{pmatrix}$.

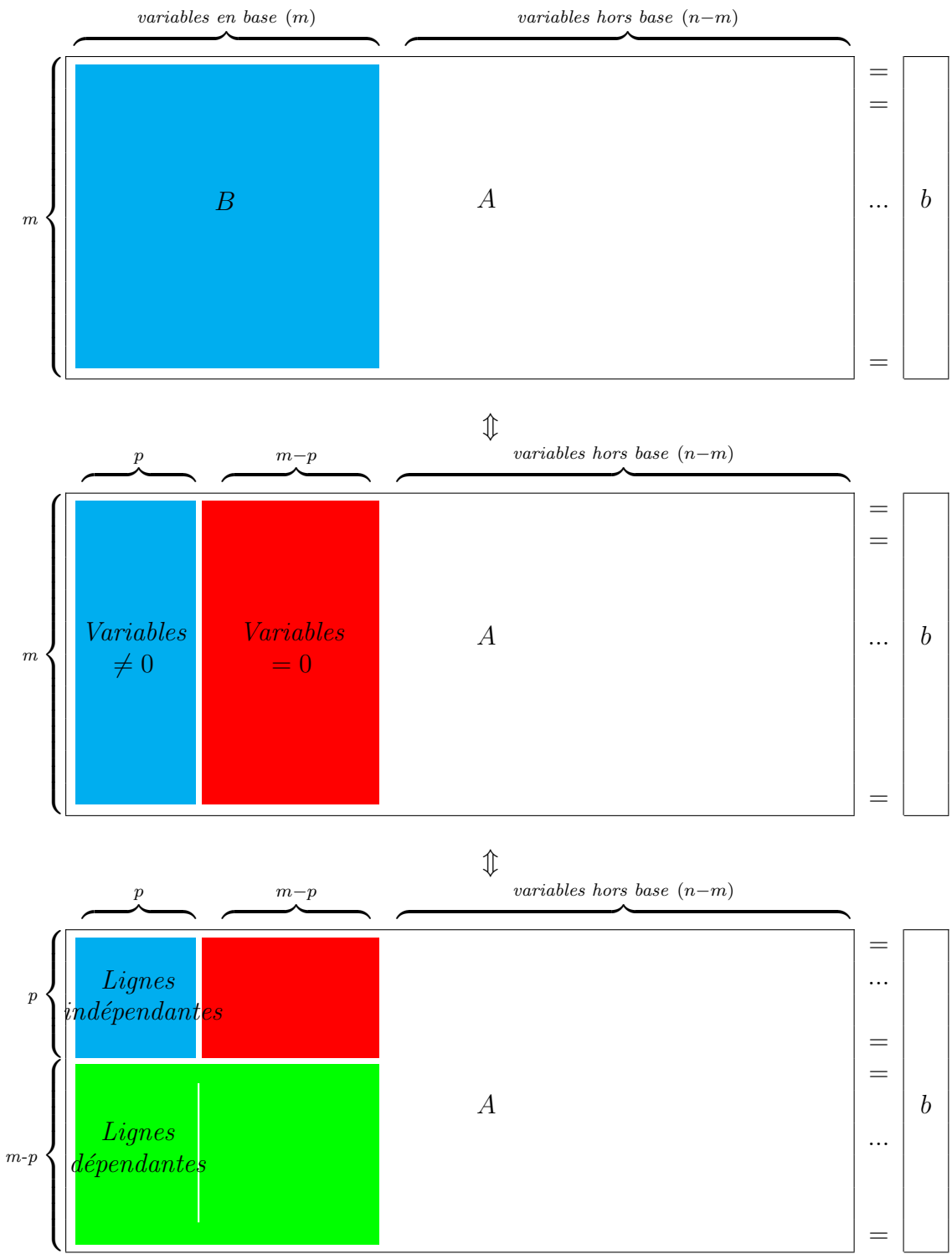


FIGURE 4.1 Schéma d'obtention du problème réduit PR_B (début)

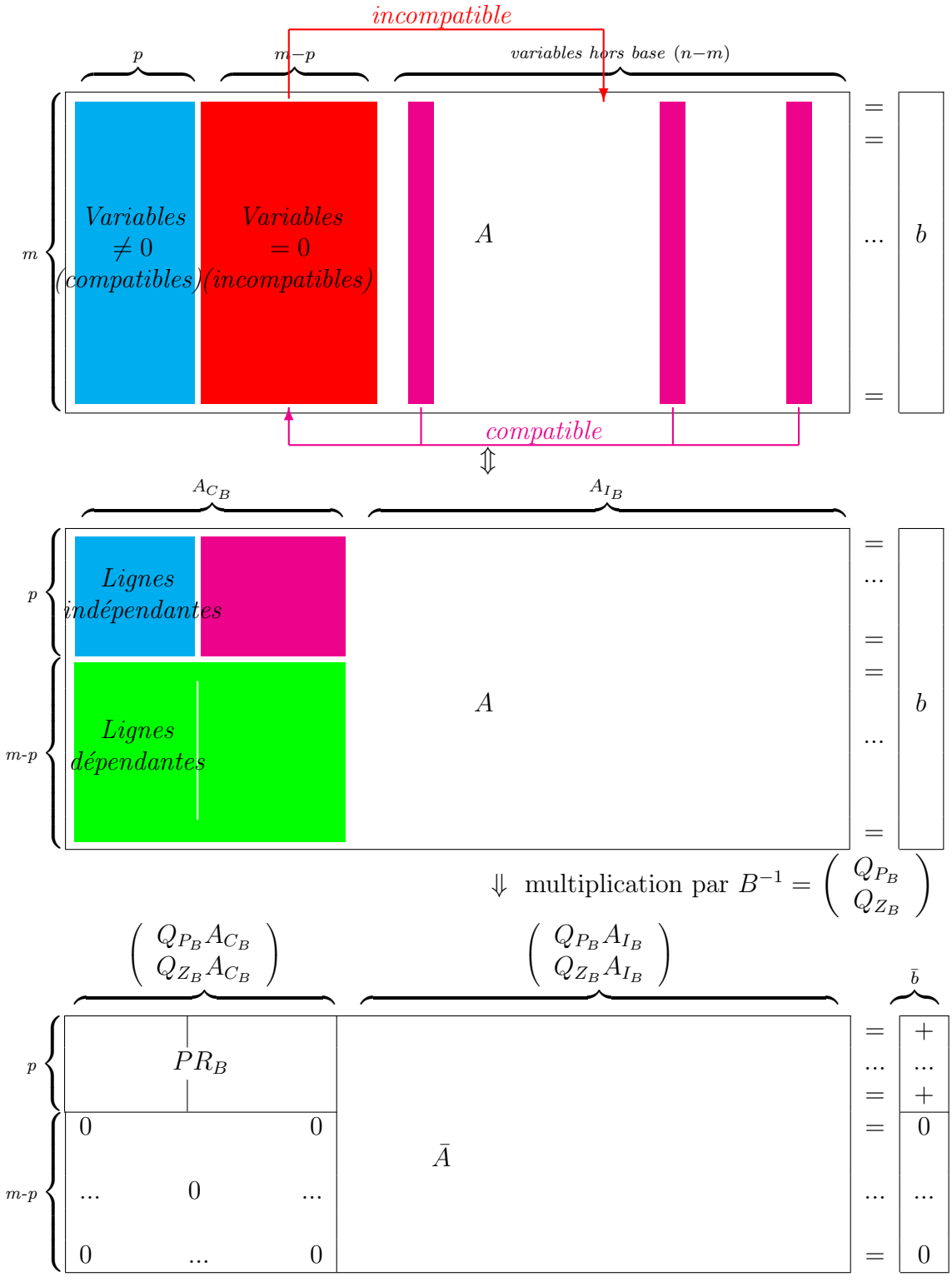


FIGURE 4.2 Schéma d'obtention du problème réduit PR_B (fin)

4.1.2 Conditions d'optimalité et problème complémentaire

Après avoir extrait le problème réduit PR_B, il est résolu à l'optimalité avec l'algorithme du simplexe primal qui fournit une solution optimale $x^*_{C_B}$. Cette solution est optimale pour PR_B mais rien ne garantit qu'elle l'est pour PL puisqu'un certain nombre de contraintes et de variables ont été "oubliées". En effet, pour PL, on a simplement que le vecteur $\begin{pmatrix} x^*_{C_B} \\ x_{I_B} \end{pmatrix} = \begin{pmatrix} x^*_{C_B} \\ 0 \end{pmatrix}$ est une solution de base réalisable. Ainsi, pour savoir si cette solution est optimale pour PL, il est nécessaire de résoudre un autre problème qui découle du théorème des écarts complémentaires : le problème complémentaire (PC_B). Nous allons expliquer sa construction.

Après avoir résolu PR_B à l'optimalité et avoir ainsi obtenu $x^*_{C_B}$, une nouvelle base $B^* \in \mathbb{R}^{p\times p}$ est obtenue contenant p^* variables positives ($p^* \leq p$). Comme précédemment, les colonnes correspondant à ces p^* variables de base positives sont placées en première position dans la matrice A. Posons maintenant les ensembles d'indices suivants :

- P_{B^*} l'ensemble des indices des p^* variables (c'est-à-dire celles qui sont positives dans $x^*_{C_B}$) ;
- Z_{B^*} l'ensemble des indices des variables de base nulles (c'est-à-dire celles qui sont nulles dans $x^*_{C_B}$).

Cela permet de définir, toujours comme dans la section précédente, la matrice $(B^*)^{-1} = \begin{pmatrix} Q_{P_{B^*}} \\ Q_{Z_{B^*}} \end{pmatrix}$, où la matrice $Q_{P_{B^*}} \in \mathbb{R}^{p^*\times p}$ est composée des p^* premières lignes de B^{-1} (c'est-à-dire celles correspondant aux nouvelles variables de base positives), et où la matrice $Q_{Z_{B^*}} \in \mathbb{R}^{(p-p^*)\times p}$ est composée des $p - p^*$ lignes suivantes (c'est-à-dire celles correspondant aux nouvelles variables de base nulles). On a également le fait que les colonnes A_l de A, qui dépendent linéairement des colonnes A_j tel que $j \in P_{B^*}$, sont les colonnes compatibles avec la matrice $\tilde{Q}_{B^*,B} = \begin{pmatrix} Q_{Z_{B^*}}Q_{P_B} \\ Q_{Z_B} \end{pmatrix} \in \mathbb{R}^{(m-p^*)\times m}$. Notons ensuite les ensembles d'indices suivants :

- C_{B^*} l'ensemble des indices des colonnes compatibles ;
- $I_{B^*} = \{1, ..., n\} \setminus C_{B^*}$ l'ensemble des indices des variables incompatibles avec $\tilde{Q}_{B^*,B}$.

On pourra remarquer que $C_{B^*} \subseteq C_B$ et $I_{B^*} \supseteq I_B$ et que les inclusions sont strictes si et seulement si $p^* < p$ (C_B et I_B représentent les mêmes ensembles d'indices que C_{B^*} et I_{B^*} mais appliqués à la matrice B introduite dans la sous-section précédente).

Avec le théorème des écarts complémentaires, on peut en déduire que la solution $\begin{pmatrix} x^*_{C_B} \\ x_{I_B} \end{pmatrix} = \begin{pmatrix} x^*_{C_B} \\ 0 \end{pmatrix}$ est une solution optimale de PL si et seulement si il existe une solution duale π de PL telle que :

$$\bar{c}_j := c_j - \pi^T A_j = 0 \qquad \forall\ j \in P_{B^*} \tag{4.11}$$

$$\bar{c}_j := c_j - \pi^T A_j \geq 0 \qquad \forall\ j \in C_{B^*} \setminus P_{B^*} \tag{4.12}$$

$$\bar{c}_j := c_j - \pi^T A_j \geq 0 \qquad \forall\ j \in I_{B^*} \tag{4.13}$$

où π est le vecteur des variables duales ;
$\bar{c}_j$ est le coût réduit de la variable x_j par rapport à π ;
c_j est le coefficient de la variable x_j dans la fonction objectif ;
A_j est la colonne associée à la variable x_j .

Ces conditions reviennent aux conditions traditionnelles utilisées dans l'algorithme du simplexe qui disent que l'optimalité est atteinte lorsque tous les coûts réduits $\bar{c}_j$ sont non négatifs.

On pourra remarquer que si les conditions (4.11) sont respectées alors les conditions (4.12) le sont aussi. Les conditions (4.12) sont donc redondantes. Cette redondance à été démontrée par Elhallaoui *et al.* (2007).

La résolution de PR_B ne permet donc peut-être pas d'obtenir une solution optimale pour PL mais elle donne les valeurs des variables duales π^*_i pour $i = 1, ..., p$. Il reste donc à savoir s'il existe des valeurs, dans les $m - p$ autres variables duales π_i (où $i = p + 1, ..., m$), qui respectent les conditions (4.11) et (4.13) ce qui permettrait de savoir si la solution $\begin{pmatrix} x^*_{C_B} \\ 0 \end{pmatrix}$ est optimale ou non. Pour vérifier cela, le problème complémentaire (PC_{B^*}) doit être résolu :

$$(PC_{B^*}) \qquad z^{PC}_{B^*} = \max_{y,\pi} y \tag{4.14}$$

$$\textit{sujet à} \quad c_j - \pi^T A_j = 0 \qquad \forall\ j \in P_{B^*} \tag{4.15}$$

$$c_j - \pi^T A_j \geq y \qquad \forall\ j \in I_{B^*}\ . \tag{4.16}$$

On pourra remarquer que ce problème est toujours réalisable mais n'est pas nécessairement borné. Sa résolution donne cependant des informations intéressantes :

- si $z^{PC}_{B^*} < 0$ alors les conditions (4.11), (4.12) et (4.13) ne peuvent être respectées et on peut en déduire qu'il n'existe pas de base réalisable contenant les colonnes A_j avec $j \in P_{B^*}$. D'où le fait que la solution $\begin{pmatrix} x^*_{C_B} \\ x_{I_B} \end{pmatrix} = \begin{pmatrix} x^*_{C_B} \\ 0 \end{pmatrix}$ n'est pas optimale pour PL et que $z^{PL} < z^{PR}_B$. Dans ce cas, la solution de PC_{B^*} fournit un coût réduit non-négatif pour toutes les variables x_j.
- si $z^{PC}_{B^*} \geq 0$ ou PC_{B^*} non borné, alors il existe une base réalisable contenant les colonnes A_j avec $j \in P_{B^*}$ et la solution $\begin{pmatrix} x^*_{C_B} \\ x_{I_B} \end{pmatrix} = \begin{pmatrix} x^*_{C_B} \\ 0 \end{pmatrix}$ est optimale pour PL.

Quand on regarde le problème PC_{B^*} de près, on s'aperçoit que sa taille augmente avec le niveau de dégénerescence de la solution $\bar{x}$. Il contient en fait $p^* + n - |C_{B^*}|$ contraintes et $m + 1$ variables ($|C_{B^*}|$ est le nombre de variables qui décroît quand la dégénérescence augmente).

En pratique, c'est au dual du problème complémentaire PC_{B^*} que l'on s'intéresse pour la résolution. Son expression est la suivante :

$$(DC_{B^*}) \qquad \min_{u,v} \; -c_P^T u + c_I^T v \tag{4.17}$$

$$\textit{sujet à} \qquad -A^P u + A^I v = 0 \tag{4.18}$$

$$e^T v = 1 \tag{4.19}$$

$$v \geq 0 \tag{4.20}$$

où u, v sont les vecteurs des variables duales des contraintes (4.15) et (4.16) ;
$e^T = (1, ..., 1)$;
$c_P = (c_j)_{j \in P_{B^*}}$;
$c_I = (c_j)_{j \in I_{B^*}}$;
$A^P = (a_{ij})_{1 \leq i \leq m,\ j \in P_{B^*}}$;
$A^I = (a_{ij})_{1 \leq i \leq m,\ j \in I_{B^*}}$.

Il est possible de simplifier ce problème dual DC_{B^*} pour faciliter sa résolution d'après Elhallaoui *et al.* (2007). Nous allons expliquer rapidement cette simplification. Comme les p^* premières colonnes de la matrice A^P sont linéairement indépendantes on

peut réécrire cette dernière de la manière suivante : $A^P = \begin{pmatrix} A_1^P \\ A_2^P \end{pmatrix}$, où $A_1^P \in \mathbb{R}^{p^* \times p^*}$ est inversible. De la même manière, on a $A^I = \begin{pmatrix} A_1^I \\ A_2^I \end{pmatrix}$. Cela permet, grâce aux contraintes (4.18) d'obtenir le système suivant :

$$\begin{cases} A_1^P u = A_1^I v \\ A_2^P u = A_2^I v \end{cases} \Leftrightarrow \begin{cases} u = (A_1^P)^{-1} A_1^I v \\ 0 = (A_2^P (A_1^P)^{-1} A_1^I - A_2^I) v \end{cases}$$

Cela permet ainsi d'obtenir le problème complémentaire dual simplifié (DS_{B^*}) suivant :

$$(DS_{B^*}) \qquad \min_v \ (c_I^T - c_P^T (A_1^P)^{-1} A_1^I) v \tag{4.21}$$

$$\textit{sujet à} \quad (A_2^P (A_1^P)^{-1} A_1^I - A_2^I) v = 0 \tag{4.22}$$

$$e^T v = 1 \tag{4.23}$$

$$v \geq 0 \tag{4.24}$$

Dans la méthode IPS que nous avons utilisée pour nos tests, c'est ce problème DS_{B^*} qui est résolu, et l'algorithme utilisé pour cette résolution est celui du simplexe dual car c'est ce qui est le plus efficace d'après Raymond *et al.* (2010).

Maintenant, posons v^* une solution de base optimale de DS_{B^*}. Grâce aux contraintes (4.18), cette solution permet d'identifier une combinaison convexe de colonnes incompatibles qui dépendent linéairement des colonnes correspondant aux variables de valeur positive (c'est-à-dire que les colonnes de la combinaison convexe sont les colonnes correspondant aux composantes $v_j > 0$ du vecteur v^*). Cette combinaison convexe peut s'écrire de la manière suivante :

$$w = \sum_{j \in S} v_j A_j \quad \text{où } S \text{ est l'ensemble des indices des variables } v_j \text{ de } v \text{ tel que } v_j > 0.$$

Par conséquent, cette combinaison convexe $A^I v$ est compatible avec la matrice $\tilde{Q}_{B^*,B}$. Ensuite, comme les valeurs optimales de PC_{B^*} et DS_{B^*} sont égales, cette combinaison convexe de variables minimise aussi le coût réduit maximum par rapport à toutes les variables contenant les variables positives x_j, $j \in P_{B^*}$. Il a d'ailleurs été démontré par Elhallaoui *et al.* (2007) que la solution $\begin{pmatrix} x^*_{C_B} \\ x_{I_B} \end{pmatrix} = \begin{pmatrix} x^*_{C_B} \\ 0 \end{pmatrix}$ est optimale si et

seulement si il n'y a pas de combinaison convexe de variables qui sont compatibles avec $\tilde{Q}_{B^*,B}$ et qui ont en plus un coût réduit maximum négatif par rapport à toutes les bases contenant les variables x_j, $j \in P_{B^*}$. Ils ont aussi démontré que, grâce à cela, la valeur de la fonction objectif décroît à chaque itération de l'algorithme IPS et qu'à chaque itération, au plus $m - p^* + 1$ variables sont nécessaires pour faire décroître la valeur de cette fonction objectif.

Des améliorations ont été apportées par Raymond *et al.* (2010) concernant les paramètres à adopter pour résoudre DS_{B^*}, le nombre de combinaisons convexes sélectionnées dans DS_{B^*} et lesquelles sélectionner. Ils ont également apporté des améliorations concernant le nombre maximum de pivots effectués de manière à ce que la résolution de PR_{B^*} fonctionne efficacement.

4.1.3 Description de l'algorithme

L'algorithme implémenté dans IPS est schématisé à la figure 4.3.

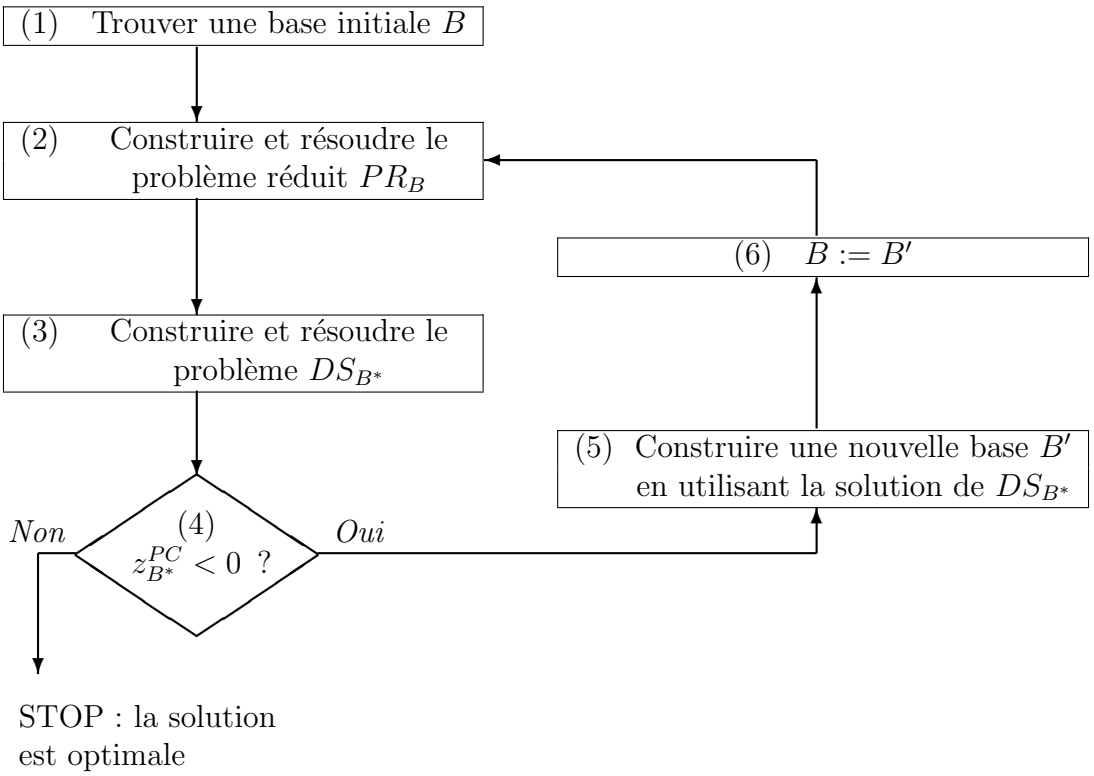


FIGURE 4.3 Algorithme IPS schématisé

- **Étape 1** : l'algorithme commence par trouver une base réalisable pour PL.
- **Étape 2** : avec la base réalisable B, le problème réduit PR_B est construit en modifiant le groupe des contraintes, en identifiant les variables incompatibles et en les enlevant. Ensuite, PR_B est résolu en utilisant l'algorithme du simplexe primal qui fournit une base B^* et une solution optimale $x^*_{C_B}$. La dégénérescence peut apparaître pendant la résolution de PR_B, cependant la petite taille du problème fait qu'elle a un impact relativement limité dans le processus.
- **Étape 3** : le problème complémentaire dual simplifié DS_{B^*} est construit à partir de la solution de PR_B et est résolu en utilisant l'algorithme du simplexe dual.
- **Étape 4** : le critère d'optimalité est testé.
- **Étape 5** : quand la solution trouvée $\begin{pmatrix} x^*_{C_B} \\ x_{I_B} \end{pmatrix} = \begin{pmatrix} x^*_{C_B} \\ 0 \end{pmatrix}$ n'est pas optimale, le problème réduit doit être modifié pour permettre de faire entrer des variables incompatibles dans la base. Cela est fait par la construction d'une nouvelle base B' qui transforme ces variables incompatibles en variables compatibles. Ces variables supplémentaires, qui peuvent donner lieu à une augmentation de la taille du problème réduit, permettront de faire décroître la valeur de la fonction objectif à la prochaine itération.

Ainsi, lorsqu'il n'y a pas de dégénerescence, IPS revient à un algorithme de simplexe primal classique qui fonctionne bien dans ce cas puisqu'il n'y a pas de pivots dégénérés, et quand la dégénérescence intervient, IPS divise le problème PL en deux problèmes PR_B et DS_{B^*} qui sont résolus alternativement. D'un côté, PR_B est plus petit que PL et moins sujet à la dégénérescence ce qui le rend plus facile à résoudre, et de l'autre côté, le problème DS_{B^*}, qui peut aussi être sujet à la dégénérescence, possède en général moins de contraintes que PL. Il est donc aussi censé être plus facile à résoudre. Cette méthode IPS, puisqu'elle garantit la décroissance de la fonction objectif à chaque résolution d'un problème réduit, permet d'éviter de longues séquences de pivots dégénérés comme cela peut se produire dans l'algorithme du simplexe primal.

D'autre part, IPS est censé obtenir sa pleine efficacité lorsque les tailles du problème réduit et du problème complémentaire sont significativement plus petites que celles de PL. Cela arrive théoriquement lorsque les ratios p/m et p^*/m sont proches de 0,5. On pourra remarquer que si p/m est proche de 1 alors c'est la taille du problème

réduit qui est proche de celle de PL, et si p^*/m est proche de 0 alors c'est la taille du problème complémentaire qui est proche de celle de PL.

En ce qui concerne la solution initiale dans IPS, Raymond *et al.* (2009) ont expliqué comment il est possible dans l'étape 1 de la figure 4.3, de faire démarrer IPS d'une solution initiale réalisable donnée, plutôt que d'une base réalisable initiale B. Cela peut être avantageux puisqu'une solution heuristique a plus de chance d'être proche de l'optimalité qu'une base réalisable initiale quelconque. De plus, le temps de calcul pour trouver une solution initiale heuristique peut être bien inférieur au temps nécessaire pour trouver cette base réalisable initiale. Enfin, on peut penser que si cette base initiale réalisable est vraiment proche de l'optimalité, le problème réduit PR_B sera plus pertinent et donc moins de boucles seront nécessaires pour en terminer la résolution jusqu'à l'optimalité. Cela devrait donc induire un temps de calcul nettement inférieur à celui que l'algorithme du simplexe primal nécessiterait pour résoudre le problème.

Enfin, pour la résolution en nombres entiers, une version d'IPS intégré dans un algorithme d'évaluation et séparation dans lequel le branchement se fait sur les bornes des variables du problème réduit PR_B, a récemment été implémentée au GERAD. Nous avons donc testé cette version selon différentes stratégies de branchement afin d'examiner les performances d'IPS dans une telle structure. Cela n'avait encore jamais été fait.

Ces hypothèses et ces interrogations sont examinées sur des instances du problème LESC de taille et densité différentes dans la sous-section suivante. Rappelons qu'il est indéniable que les méthodes duales soient plus performantes que les méthodes primales en présence de dégénérescence, cependant, le but des sections suivantes ainsi que celui du mémoire est d'analyser l'amélioration que présente IPS par rapport à l'algorithme du simplexe primal sur le problème LESC qui possède des caractéristiques intéressantes. Des résultats sur les méthodes duales sont quand même présentées, mais simplement à titre d'information.

4.2 Tests sur la relaxation linéaire

4.2.1 Instances et paramètres

Afin de tester les performances d'IPS, nous avons créé des instances du problème LESC de densités et de tailles différentes. Ces instances sont créées sur un modèle légèrement différent de celui des instances Galvão-Raggi. Elles présentent les caractéristiques suivantes :

- $n = m$,
- les ensembles des sites et des clients sont différents,
- les coûts d'installation sont choisis aléatoirement selon une loi normale de moyenne 1800 et d'écart type 500,
- les coûts d'affectation sont pris de manière à ce que le problème représente un graphe biparti (l'ensemble 1 est l'ensemble des sites et l'ensemble 2 est l'ensemble des clients) dont la densité δ des arêtes est variable selon les instances. Les coûts sur les arcs sont choisis aléatoirement selon une loi uniforme sur l'intervalle $[20, 50]$. La matrice d'affectation est donc non symétrique et sa diagonale est non nulle.
- les valeurs des coûts d'affectation c_{ij} sont soit infinies s'il n'y a pas d'arc entre le site i et le client j (c'est-à-dire que la variable x_{ij} correspondante n'est pas créée dans le problème résolu par le solveur), soit égales au plus court chemin du site i au client j dans le graphe s'il y a un arc.
- Six tailles différentes de problème ont été choisies : 200×200, 300×300, 400×400, 500×500, 600×600 et 700×700.
- Pour chacune de ces tailles, des densités différentes ont été testées : 0,05 ; 0,1 ; 0,2 ; 0,3 ; 0,4 ; 0,5 ; 0,6 ; 0,7 ; 0,8 ; 0,9 et 1.
- Pour chaque taille et chaque densité, cinq instances ont été crées et résolues, par conséquent chaque point d'articulation des courbes est le résultat d'une moyenne sur cinq instances.

En ce qui concerne la recherche tabou, nous avons choisi de fixer les paramètres de la manière suivante :

$$\begin{aligned} a &= 5 \\ b &= n/4 \\ K &= 20 \\ NbrIt &= 20n\ . \end{aligned}$$

Ces paramètres ont été choisis de cette manière pour que le temps de recherche de la solution initiale ne prenne pas trop de temps par rapport à celui d'IPS. Il est a noter que lorsque les densités δ sont faibles, cela fait ressembler les instances à celles de la famille Grand saut de dualité vues au chapitre précédent. Au contraire, lorsque les densités sont fortes, cela les fait ressembler davantage à celles de la famille Koerkel-Ghosh asymétrique. Par conséquent, comme ces deux familles sont très différentes, il est important que les paramètres choisis permettent une recherche efficace quelque soit l'instance résolue. Après un certain nombre d'essais, nous avons pu observer que ces paramètres donnaient en moyenne en peu d'itérations des solutions de bonne qualité même avec cette grande diversité d'instances. Enfin, pour IPS comme pour Cplex, la version 10.1.1 de Cplex a été utilisée pour l'ensemble des tests effectués.

4.2.2 Résultats

Les résultats obtenus sont présentés dans les figures 4.4 à 4.9.
Deux graphiques sont dessinés dans chacune de ces figures :

- le premier donne les temps de calcul pour la résolution de la relaxation linéaire du problème : la courbe *IPS avec sol. ini.* donne les temps d'IPS avec solution initiale (englobant le temps mis pour obtenir la solution initiale), la courbe *IPS sans sol. ini.* donne les temps d'IPS sans solution intiale, la courbe *Cplex primopt* donne les temps de Cplex utilisant l'algorithme du simplexe primal, la courbe *Cplex dualopt* donne les temps de Cplex utilisant l'algorithme du simplexe dual, et la courbe *Cplex baropt* donne les temps de Cplex utilisant l'algorithme barrière.
- Le deuxième graphique donne les ratios des temps de calcul pour la résolution de la relaxation linéaire du problème : la courbe *Cplex primopt / IPS avec sol. ini.* donne les ratios des temps de Cplex utilisant l'algorithme du simplexe primal par rapport à ceux d'IPS avec solution initiale, et la courbe *Cplex primopt / IPS sans sol. ini.* donne les ratios des temps de Cplex utilisant l'algorithme du simplexe primal par rapport à ceux d'IPS sans solution initiale.

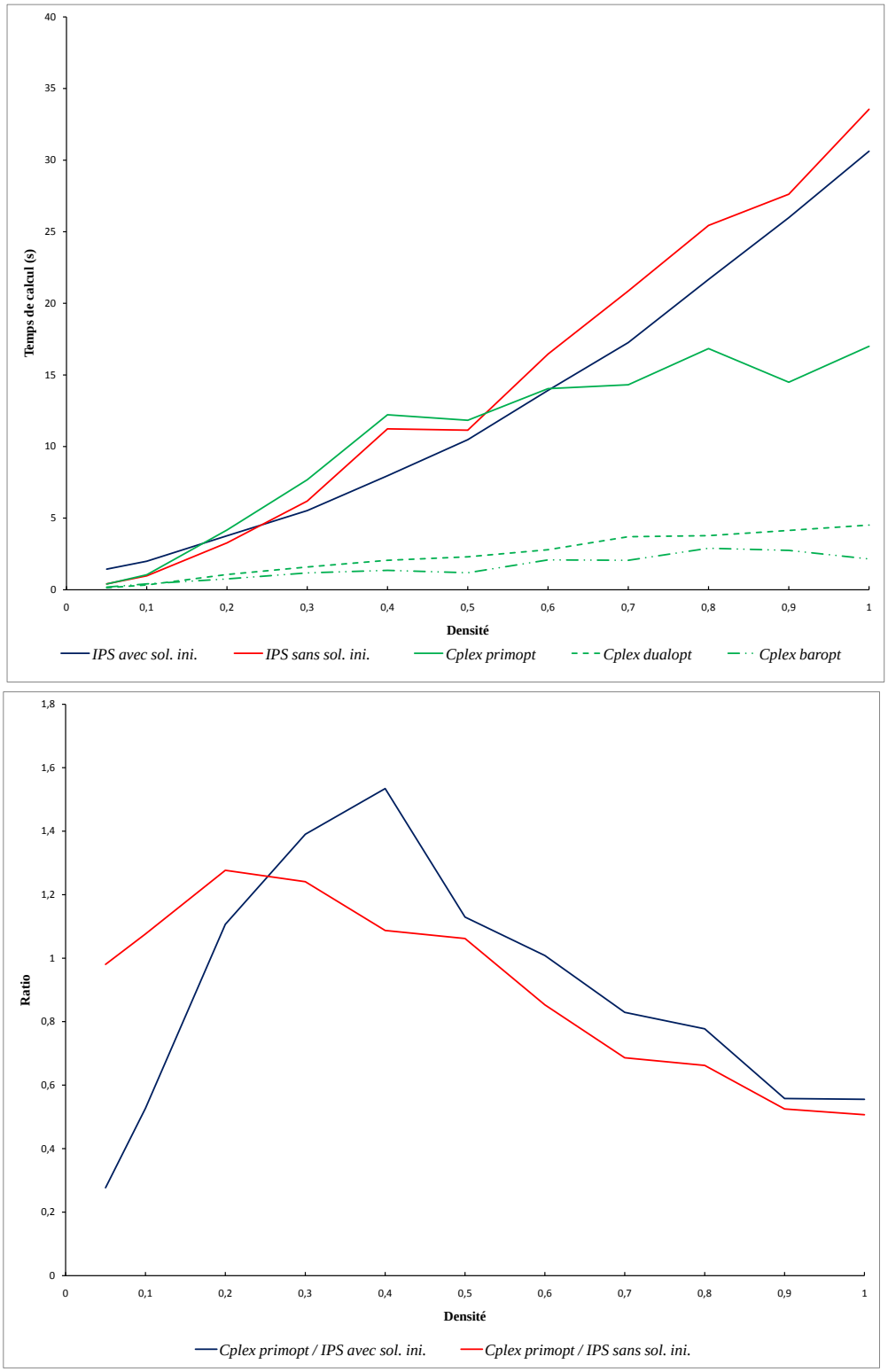
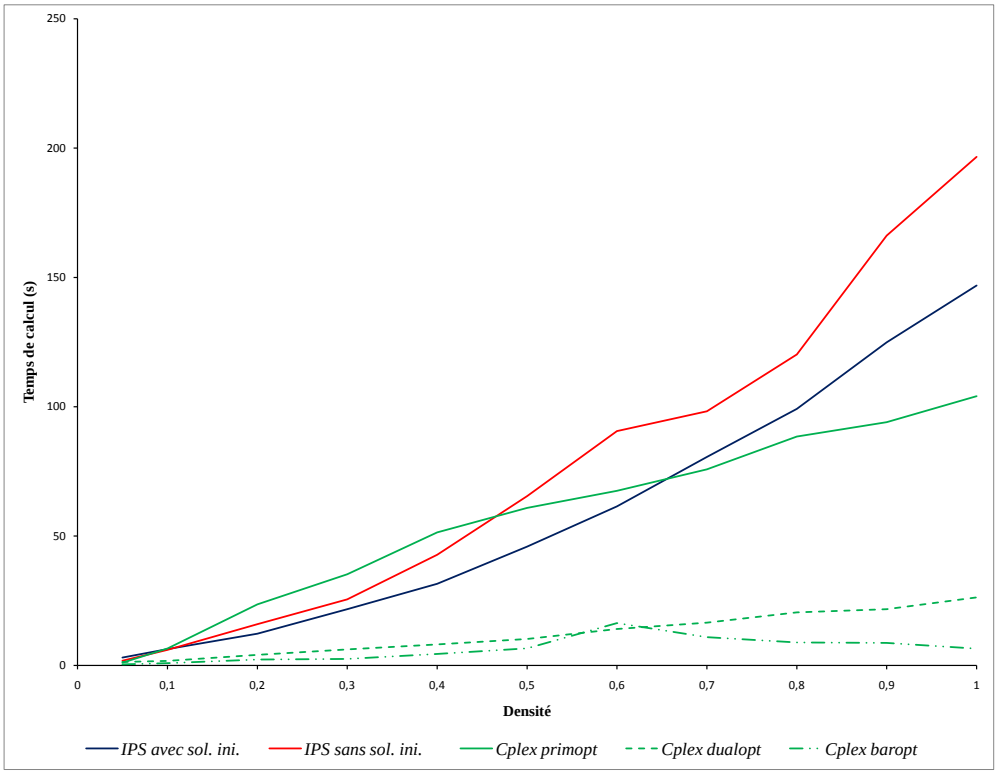


FIGURE 4.4 Résultats IPS instances 200x200, relaxation linéaire



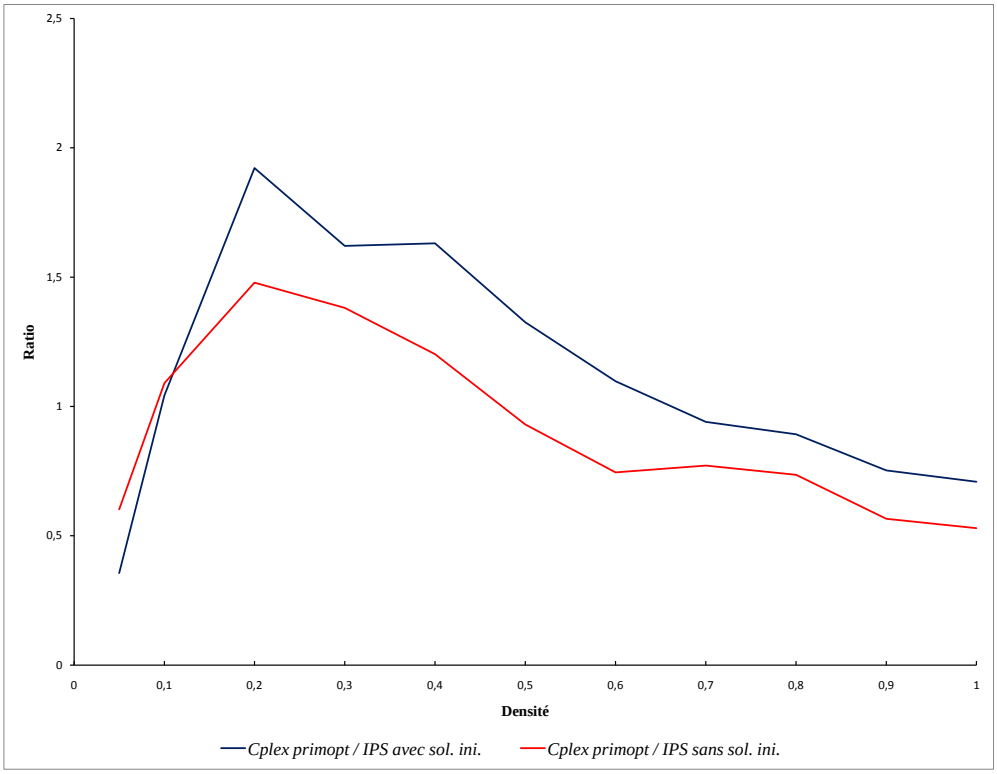
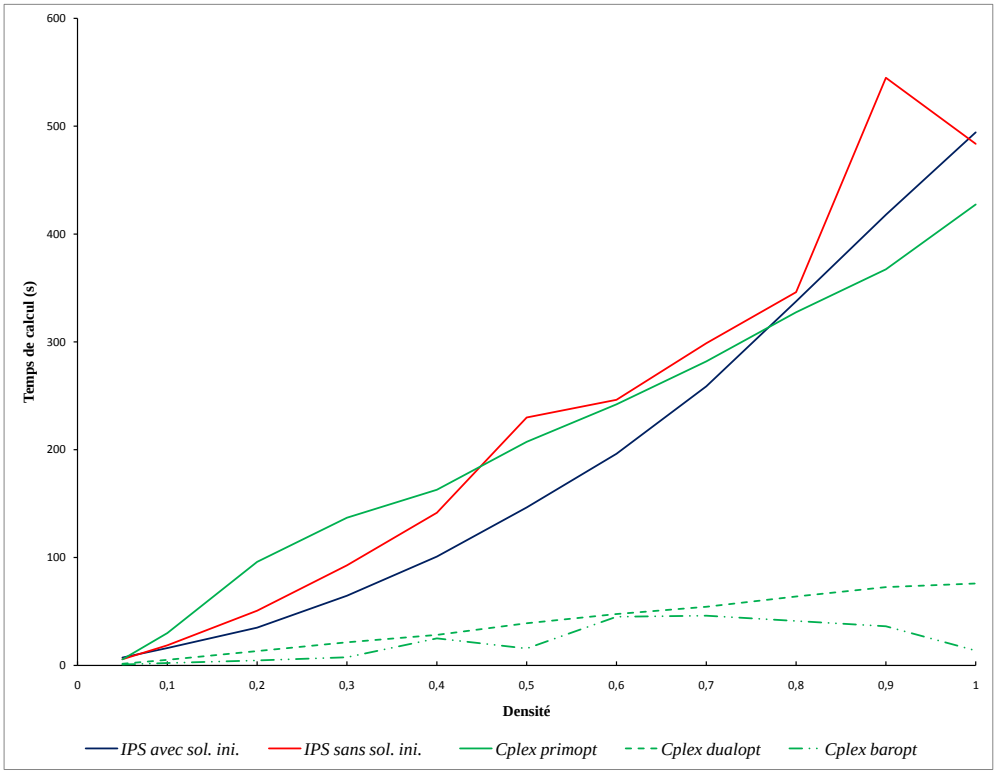


FIGURE 4.5 Résultats IPS instances 300x300, relaxation linéaire



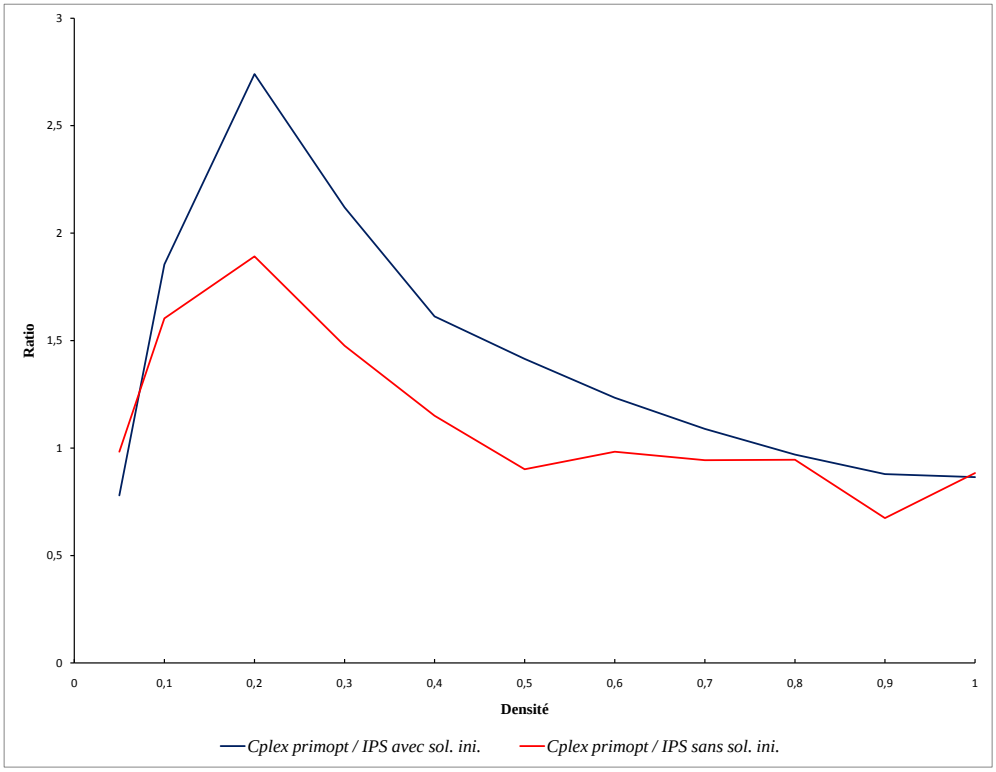
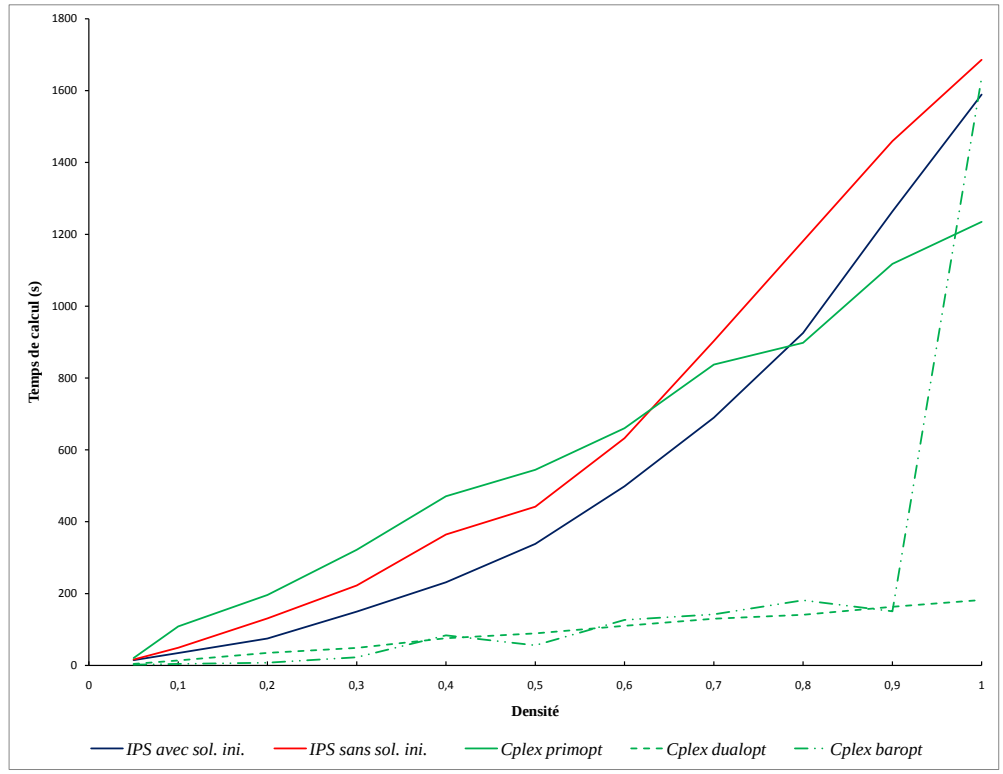


FIGURE 4.6 Résultats IPS instances 400x400, relaxation linéaire



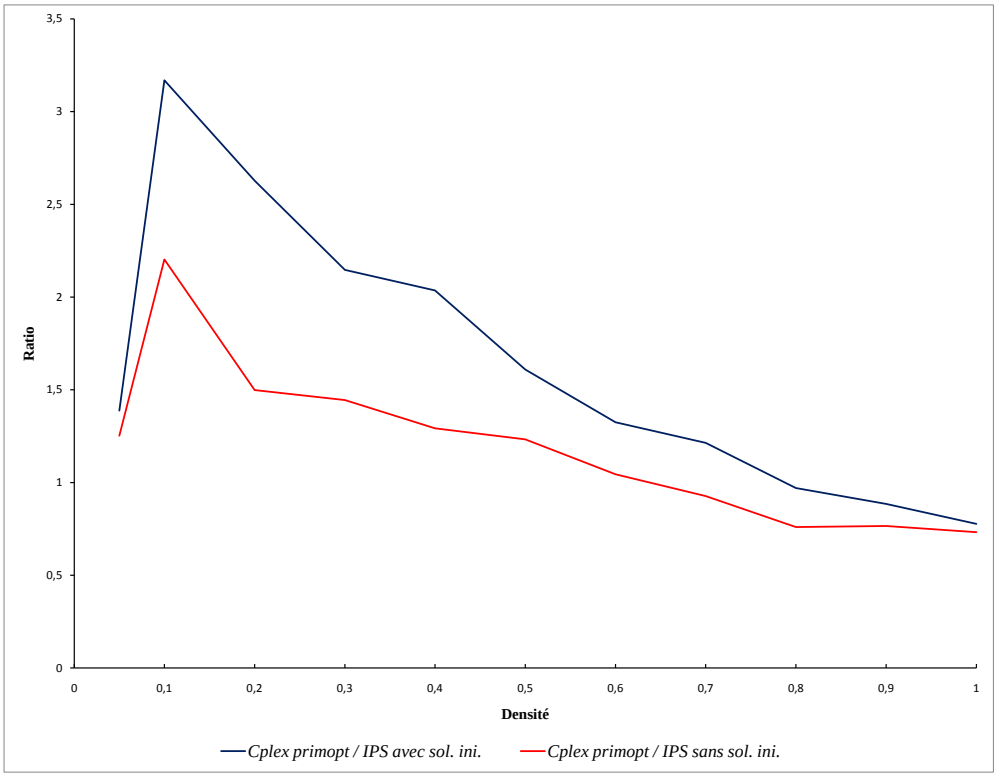
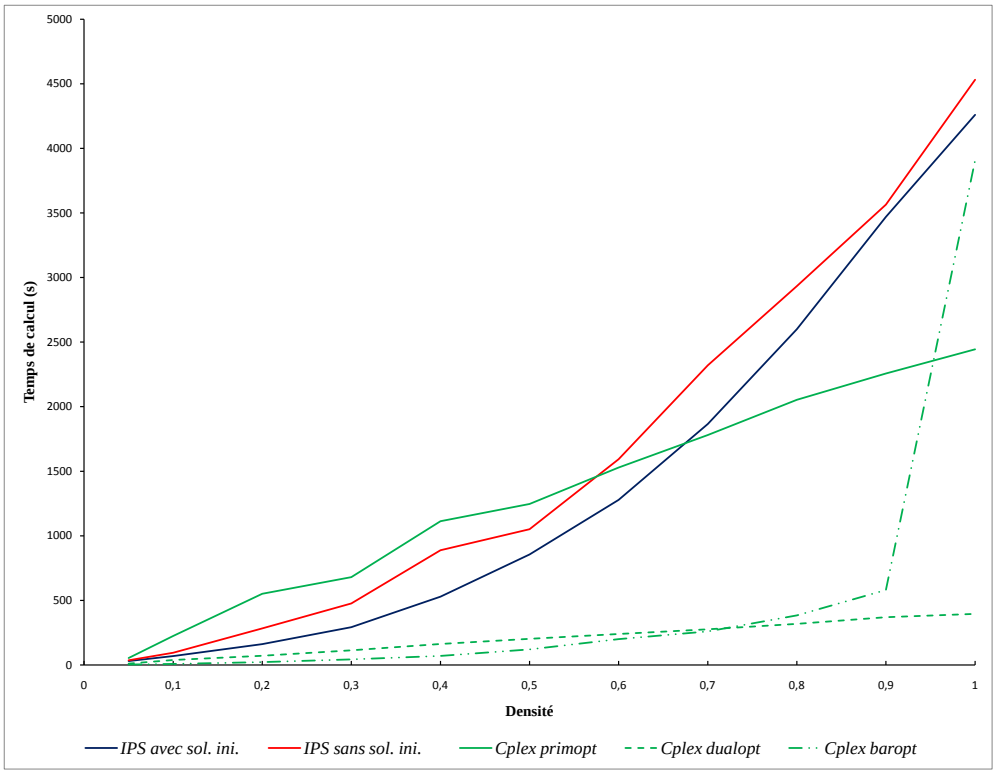


FIGURE 4.7 Résultats IPS instances 500x500, relaxation linéaire



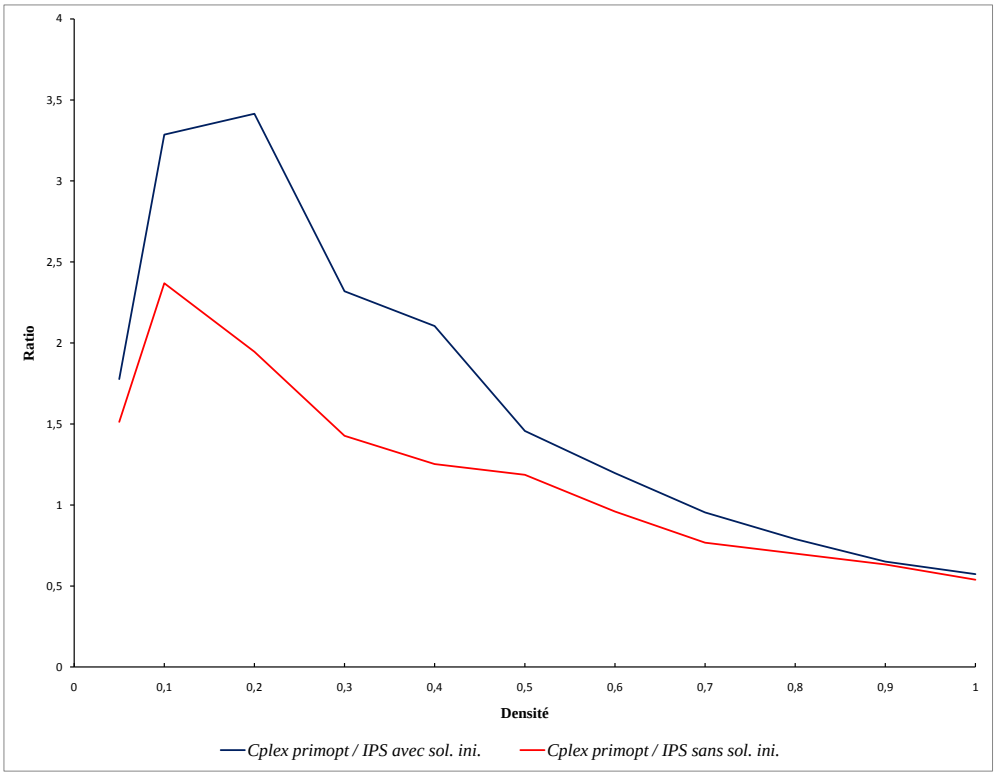
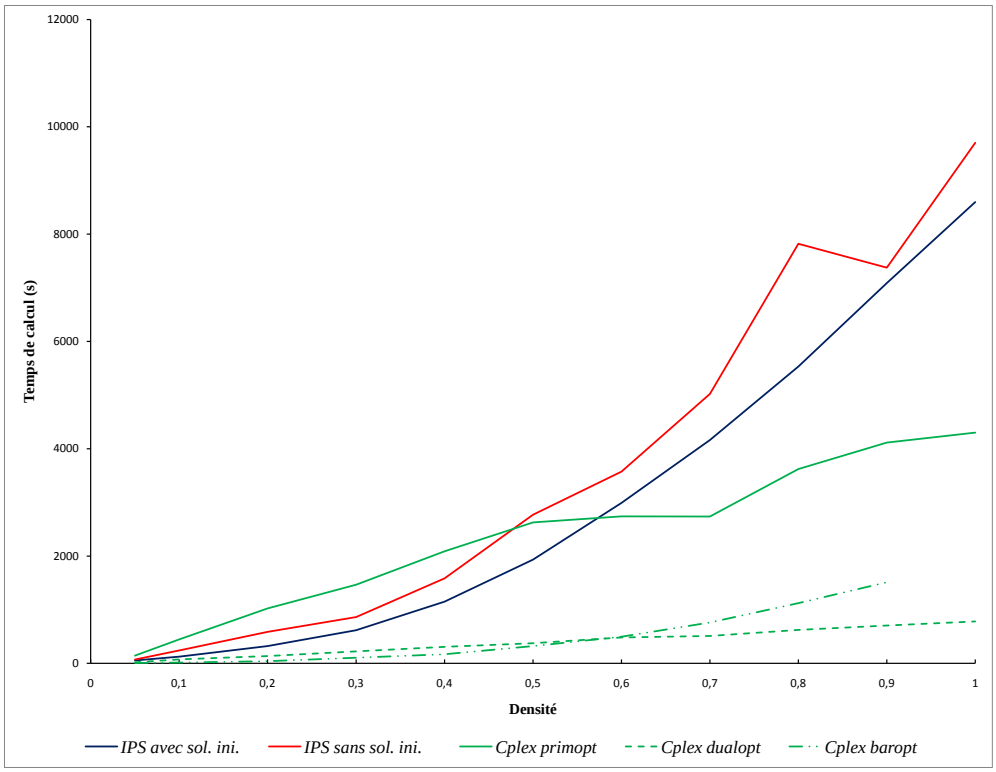


FIGURE 4.8 Résultats IPS instances 600x600, relaxation linéaire



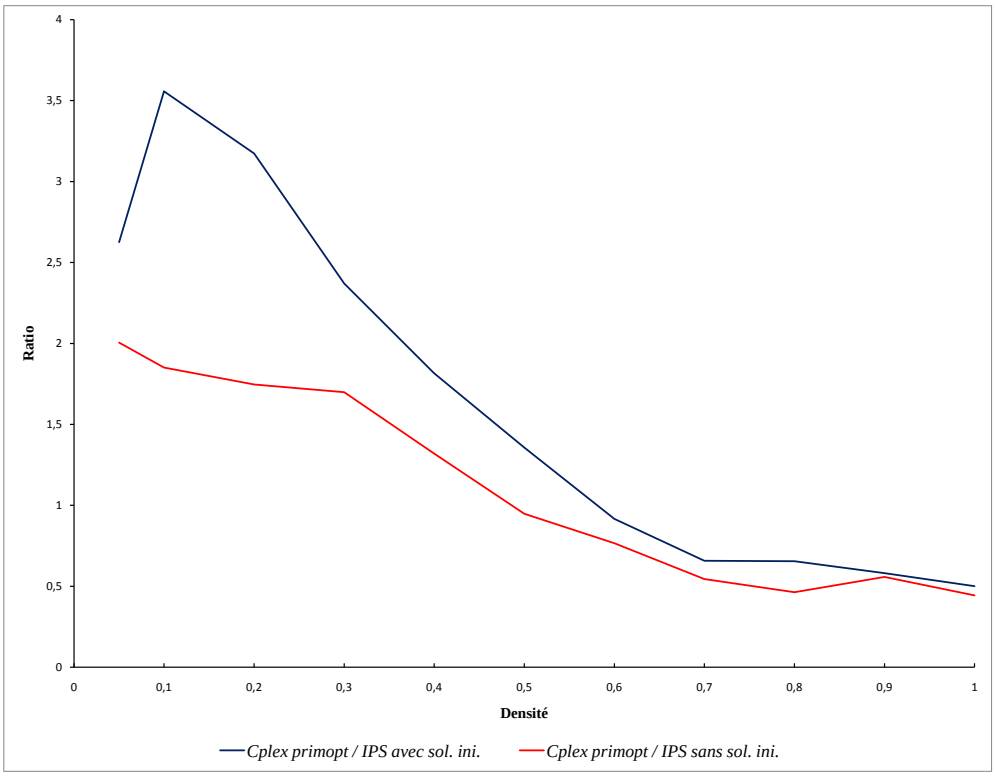


FIGURE 4.9 Résultats IPS instances 700x700, relaxation linéaire

Ces courbes montrent tout d'abord que les temps de calcul augmentent exponentiellement avec la taille des instances. Ensuite, on peut observer que IPS avec solution initiale est presque toujours plus rapide qu'IPS sans solution initiale. Cette supériorité est particulièrement marquée dans l'intervalle de densité $[0,1\ ;\ 0,5]$, avec une diminution du temps de calcul pouvant aller jusqu'à un facteur proche de 2 comme c'est le cas pour les instances 600×600 de densité 0,2. Les seuls cas où cela n'est pas vérifié arrivent lorsque les instances sont de petite taille et de densité très faible ($\leq 0,1$) car dans ce cas, la relaxation linéaire est résolue tellement rapidement que le temps mis par la recherche tabou pour obtenir la solution initiale devient supérieur au gain de temps engendré par l'apport de cette solution.

D'autre part, il est important d'observer que les temps de calcul d'IPS avec solution initiale et sans solution initiale sont presque toujours inférieurs à ceux de Cplex utilisant l'algorithme du simplexe primal pour les intances de faible densité. Pour IPS avec solution initiale résolvant les instances 200×200 et 300×300, cette supériorité n'est vérifiée respectivement qu'à partir des densités 0,2 et 0,1, à cause du même phénomène décrit dans le paragraphe précédent où le temps mis par la recherche de la solution initiale est trop important. Cette supériorité est d'ailleurs très marquée dans l'intervalle de densité $[0,1\ ;\ 0,3]$, avec des ratios parfois supérieurs à 3 comme c'est le cas pour les très grandes instances (500×500, 600×600 et 700×700). Au-delà d'une densité limite où Cplex utilisant l'algorithme du simplexe primal et IPS avec solution initiale sont équivalents, Cplex prend le dessus. Cette densité limite varie selon la taille des instances et est atteinte lorsque le ratio *Cplex primal / IPS avec sol. ini.* $= 1$. On peut d'ailleurs remarquer que cette densité limite se déplace vers les densités croissantes quand les tailles des instances augmentent pour les instances de taille 200×200 à 500×500, jusqu'à atteindre un maximum de densité 0,8 pour les instances 500×500. Elle semble ensuite stagner autour des densités 0,5 et 0,6 pour les instances de plus grande taille ($\geq 600 \times 600$).

En ce qui concerne les pics des ratios *Cplex primal / IPS avec sol. ini.* et *Cplex primal / IPS sans sol. ini.* situés à des densités comprises dans l'intervalle $[0,1\ ;\ 0,3]$, on peut expliquer cela par le fait que ces densités génèrent la "quantité" de dégénérescence idéale qui permet à IPS de prendre toute son efficacité par rapport à Cplex utilisant l'algorithme du simplexe primal. En effet, lorsque les densités sont trop faibles (δ autour de 0,05), certes les problèmes réduits deviennent très petits et donc très faciles à résoudre, mais les problèmes complémentaires deviennent presque aussi

grands que le problème original résolu par Cplex, ce qui empêche IPS de montrer ses avantages. Inversement, quand les densités se rapprochent de 1, c'est-à-dire que les problèmes ont des densités très fortes, cela donnent lieu à moins de dégénérescence, ce qui fait que Cplex est moins soumis aux problèmes des pivots dégénérés, et IPS résout des problèmes réduits de taille plus grande, ce qui rapproche les deux algorithmes en termes de performance.

Enfin, pour ce qui est des courbes *Cplex dualopt* et *Cplex baropt*, elles permettent de comparer l'algorithme du simplexe primal et IPS à d'autres algorithmes de résolution très utilisés lorsqu'il y a de la dégénérescence. Elles montrent que même avec l'accélération procurée par IPS pour les faibles densités, les algorithmes du simplexe dual et barrière continuent à être nettement plus performants sur le problème LESC que IPS et l'algorithme du simplexe primal.

4.3 Tests sur la résolution en nombres entiers

4.3.1 Instances et paramètres

Pour tester les performances d'IPS pour la résolution en nombres entiers, nous avons choisi 5 instances de taille 200×200 et de densité $\delta = 0,4$. En effet, sur les instances de plus grande taille, le temps de calcul est beaucoup trop long. De plus, lorsque l'on choisit des instances de densité trop faible, le saut d'intégrité augmente fortement, ce qui donne lieu à un arbre de branchement particulièrement grand, et de ce fait, beaucoup plus de relaxations linéaires doivent être résolues. La densité $\delta = 0,4$ a été choisie car c'est celle qui permet d'obtenir le meilleur ratio entre Cplex utilisant l'algorithme du simplexe primal et IPS avec solution initiale pour les instances 200×200 ; ce choix permettra d'observer si IPS conserve ou non son avantage par rapport à Cplex lorsque la résolution se fait en nombres entiers.

En ce qui concerne la recherche tabou, nous avons gardé les paramètres précédents :

$$\begin{aligned} a &= 5 \\ b &= n/4 \\ K &= 20 \\ NbrIt &= 20n\ . \end{aligned}$$

Enfin, par souci d'égalité entre les deux méthodes de résolution, nous avons

désactivé dans Cplex les techniques d'optimisation dans l'arbre de branchement auxquelles IPS n'a pas accès : c'est-à-dire la génération de coupes et l'heuristique de recherche de solutions entières.

Ensuite, pour des raisons d'efficacité dans les statégies de branchement de l'algorithme d'évaluation et séparation, nous imposons que pour un branchement meilleur d'abord, le choix des variables se fasse en priorité sur les variables ayant une valeur fractionnaire la plus proche de 0,5 (paramètre "mip strategy variableselect -1" dans Cplex), et pour une stratégie de branchement profondeur d'abord, nous imposons au contraire que le choix des variables se fasse en priorité sur les variables ayant une valeur fractionnaire la plus éloignée de 0,5 (paramètre "mip strategy variableselect 1" dans Cplex). De cette manière, Cplex comme IPS fonctionnent à "forces égales".

Ces paramètres s'appliquent à tous les tests de la sous-section suivante exceptés ceux concernant *Cplex défaut*.

4.3.2 Résultats

Les résultats de la résolution en nombres entiers sont présentés dans les tableaux 4.1, 4.2 et 4.3. Le tableau 4.1 contient les résultats de la résolution en nombres entiers faite par un algorithme d'évaluation et séparation utilisant IPS avec solution initiale pour résoudre les relaxations linéaires des nœuds de branchement. Deux stratégies de branchement sont testées : une stratégie meilleur d'abord et une stratégie profondeur d'abord. Pour chacune de ces deux stratégies, le temps de calcul en secondes (*Tps (s)*) nécessaire à la résolution des instances est donné, ainsi que le nombre de nœuds de branchement (*Nb nœuds*) qu'il a fallu à la méthode pour y arriver. Pour les temps de calcul, le temps d'obtention de la solution initiale est négligeable devant le temps mis par l'algorithme d'évaluation et séparation. Dans ce tableau 4.1, on peut d'abord remarquer que dans tous les cas, le temps nécessaire pour la stratégie meilleur d'abord est plus faible que le temps mis par la stratégie profondeur d'abord. Cela se voit bien au niveau du nombre de nœuds évalués qui met en valeur le fait que la stratégie profondeur d'abord nécessite la résolution de nettement plus de relaxations linéaires que la stratégie meilleur d'abord.

Le tableau 4.2 présente, quant à lui, les résultats d'IPS sans utiliser de solution initiale. Comme précédemment, deux stratégies de branchement sont testées : une stratégie meilleur d'abord et une autre profondeur d'abord. Les temps de calcul ainsi

que le nombre de nœuds évalués sont donnés. Les conclusions sont les mêmes que pour le tableau 4.1 : la stratégie meilleur d'abord est plus efficace. On pourra cependant remarquer que l'utilisation d'une solution initiale amène à nouveau des gains dans les temps de calcul, mais seulement de l'ordre de 8,7% et de 6,2% pour les stratégies meilleur d'abord et profondeur d'abord, respectivement. Il est normal que ces gains ne soient pas élevés car l'utilisation de la solution initiale a surtout un impact sur la résolution de la première relaxation linéaire.

N° instance	*IPS meilleur d'abord*		*IPS profondeur d'abord*	
	Nb nœuds	*Tps (s)*	*Nb nœuds*	*Tps (s)*
1	831	1987	9925	8997
2	1101	2811	11657	10433
3	507	1105	7193	5712
4	979	2172	7099	6216
5	1661	3604	15559	12170
Moyenne	1016	2336	10287	8706
Écart-type	424	935	3519	2749

TABLEAU 4.1 Résultats de la résolution en nombres entiers d'IPS avec solution initiale

N° instance	*IPS meilleur d'abord*		*IPS profondeur d'abord*	
	Nb nœuds	*Tps (s)*	*Nb nœuds*	*Tps (s)*
1	829	2180	10545	9514
2	1099	3043	12157	11246
3	505	1271	7115	6019
4	989	2506	7123	6539
5	1659	3795	16279	13071
Moyenne	1016	2559	10644	9278
Écart-type	423	944	3837	3018

TABLEAU 4.2 Résultats de la résolution en nombres entiers d'IPS sans solution initiale

Enfin, le tableau 4.3 procure les résultats obtenus par Cplex, mais cette fois-ci selon 3 stratégies différentes :

– La première est celle utilisée par Cplex tel qu'il est programmé par défaut (colonne *Cplex défaut* dans le tableau) : on lui laisse choisir lui même l'algorithme

de résolution qu'il désire pour résoudre les relaxations linéaires des nœuds de branchement (dans notre cas, Cplex choisit l'algorithme du simplexe dual), il a le droit au présolveur, à la stratégie qu'il désire dans l'arbre de branchement (dans notre cas, Cplex adopte une stratégie meilleur d'abord avec des heuristiques permettant d'affiner ses choix de nœuds) et il a le droit d'utiliser toutes les coupes qu'il désire pour réduire le nombre de nœuds parcourus.

– La deuxième stratégie est la stratégie meilleur d'abord avec obligation pour Cplex d'utiliser l'algorithme du simplexe primal sur tous les nœuds de branchement (colonne *Cplex Meil. d'abord* dans le tableau). On a interdit l'utilisation des coupes et des heuristiques d'optimisation qui ne sont pas présentes dans l'algorithme d'évaluation et séparation utilisant IPS comme on l'a vu précédemment.
– La troisième statégie est similaire à la seconde sauf qu'elle explore l'arbre selon la stratégie profondeur d'abord (colonne *Cplex Meil. d'abord* dans le tableau).

N°	*Cplex défaut*		*Cplex meil. d'abord*		*Cplex prof. d'abord*	
instance	*Nb nœuds*	*Tps (s)*	*Nb nœuds*	*Tps (s)*	*Nb nœuds*	*Tps (s)*
1	1551	316	2242	1824	8488	3321
2	1976	380	2709	2354	9432	3942
3	894	153	782	697	6334	2307
4	2850	439	2005	1529	6764	2606
5	2499	475	3061	2397	13482	4870
Moyenne	1954	353	2160	1760	8900	3409
Écart-type	772	127	872	697	2855	1035

Tableau 4.3 Résultats de la résolution en nombres entiers de Cplex

On pourra remarquer dans le tableau 4.3 que Cplex utilisant la stratégie meilleur d'abord est plus performant que Cplex utilisant la stratégie profondeur d'abord. En effet, comme c'était le cas pour IPS dans les tableaux 4.1 et 4.2, la stratégie meilleur d'abord nécessite nettement moins de nœuds de branchement avec en moyenne 4 fois moins de nœuds résolus. Cela se ressent logiquement au niveau des temps de calcul de Cplex avec la stratégie meilleur d'abord qui résout les instances en nombres entiers en moyenne 2 fois plus rapidement qu'avec la stratégie profondeur d'abord.

Ensuite, les résultats de Cplex avec ses paramètres par défaut montrent des

résultats nettement supérieurs à ceux de *Cplex meil. d'abord* avec des temps de calcul en moyenne 5 fois plus faibles. Ce gain en temps de calcul est principlament dû à l'algorithme utilisé pour résoudre les relaxations linéaires dans Cplex avec sa configuration par défaut (algorithme du simplexe dual) puisque le nombre moyen de nœuds de branchement résolus entre *Cplex défaut* et *Cplex meil. d'abord* est particulièrement proche. Les coupes et l'heuristiques pour le choix des nœuds à résoudre sont quand même efficaces pour certaines instances (instances 1, 2 et 5) dont le nombre de nœuds résolus est inférieur. Enfin, *Cplex défaut* engendre un écart encore plus important par rapport à *Cplex prof. d'abord*, d'une part à cause de l'algorithme utilisé pour résoudre les relaxations linéaires, et d'autre part à cause de l'utilisation de la stratégie meilleur d'abord qui est nettement plus efficace.

À travers ces tableaux, une chose importante se dégage : l'avantage qu'IPS offrait par rapport à l'algorithme du simplexe primal lors de la résolution de la relaxation linéaire, qui a été observé dans les résultats de la section 4.2, s'inverse dans le cas de la résolution en nombres entiers. Cela est vrai aussi bien pour IPS avec solution initiale que pour IPS sans solution initiale. En effet, si l'on regarde les ratios des temps de calcul sur le nombre de nœuds évalués, Cplex utilisant l'algorithme du simplexe primal devient 2 à 3 fois plus rapide qu'IPS.

Cela peut être expliqué par le fait qu'actuellement, tel que l'algorithme d'évaluation et séparation est implémenté et pour des raisons techniques, l'algorithme ne permet à IPS de faire des réductions sur le problème qu'au premier nœud de l'arbre de branchement. Par conséquent, comme au fur et à mesure que la résolution avance il est nécessaire d'augmenter la taille du problème réduit pour aboutir à l'optimalité, après la résolution de quelques nœuds de branchement, le problème réduit devient quasiment aussi grand que le problème original. Donc finalement, après quelques nœuds de branchement, les problèmes réduits résolus par IPS deviennent aussi difficiles à résoudre que le problème original résolu par Cplex. Ce phénomène s'observe bien dans la figure 4.10 où l'on voit la croissance rapide des nombres de contraintes et de colonnes du problème réduit pour l'instance 1, dans le cas de sa résolution par IPS avec solution initiale. Le nombre de contraintes du problème réduit plafonne d'ailleurs à 16449 et 16535 pour les stratégies meilleur d'abord et profondeur d'abord, respectivement, alors que le nombre de contraintes du problème original est de 16704.

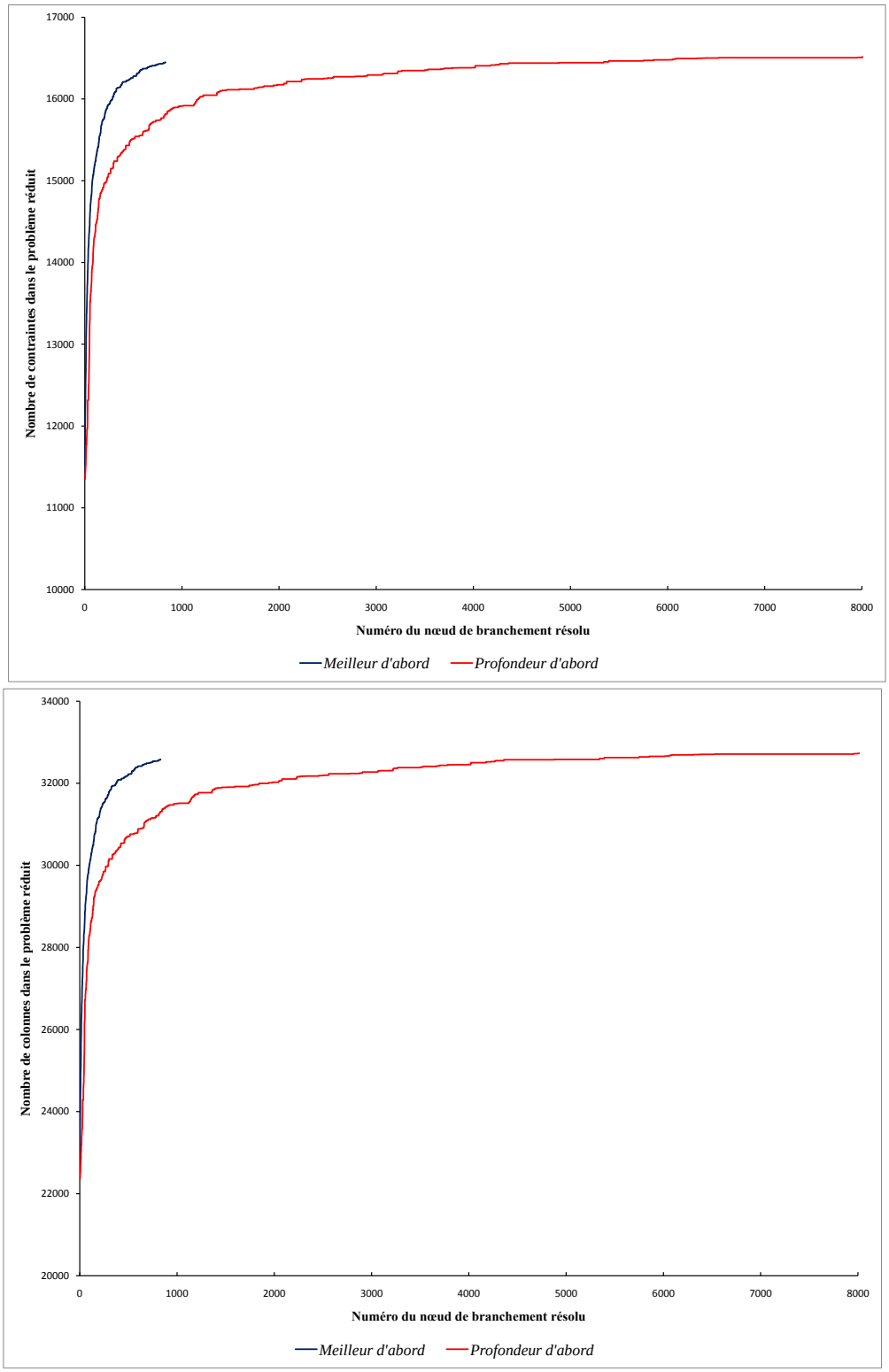


FIGURE 4.10 Évolution du nombre de contraintes et de colonnes du problème réduit

De même, le nombre de colonnes du problème réduit plafonne à 32580 et 32776 pour les stratégies meilleur d'abord et profondeur d'abord, respectivement, alors que le nombre de colonnes du problème original est de 33208. Des évolutions similaires sont observées sur les temps de calcul des résolutions des relaxations linéaires.

En plus de cela, à chaque démarrage d'une résolution de relaxation linéaire par IPS, un temps de calcul est requis pour "préparer" la résolution (fabrication du problème réduit, etc.), qui normalement devrait être comblé par la meilleure efficacité de l'algorithme IPS, mais qui dans ce cas, fait augmenter d'autant plus le temps de calcul. Ainsi, le gain d'efficacité d'IPS ne se fait sentir que sur les premiers nœuds de branchement ce qui est négligeable devant le temps de résolution total. Après ces premiers nœuds de branchement, l'inverse se produit : Cplex gagne du temps sur chaque relaxation linéaire. Cplex comble donc rapidement son retard sur IPS et accroît ensuite l'écart jusqu'à la fin de la résolution.

4.4 Analyse

À travers l'ensemble des tests qui ont été effectués sur la résolution des relaxations linéaires et les résolutions du problème en nombres entiers, nous avons remarqué que bien que la méthode IPS engendre un avantage non négligeable sur la résolution des relaxations linéaires des instances de densité comprise entre 0,1 et 0,6, cet avantage se perd dans la résolution en nombres entiers. Ceci s'explique par l'impossibilité actuelle de pouvoir faire des réductions sur les sous-problèmes des nœuds de l'arbre de branchement à l'exception du premier nœud. IPS perd ainsi tout l'avantage dont il disposait sur l'algorithme du simplexe primal après quelques nœuds de branchement. Nous avons même pu observer que cela avait une très forte incidence sur les temps de calcul puisque les temps obtenus par IPS deviennent 2 à 3 fois moins bons que ceux de Cplex. Ce phénomène s'observe logiquement aussi bien pour la résolution utilisant IPS avec solution initiale que celle utilisant IPS sans solution initiale.

Il est aussi intéressant de mentionner que même si les heuristiques et le système de coupe implanté dans Cplex avec sa configuration par défaut ont un certain impact dans la résolution du problème en nombres entiers, la grande part du gain en temps de calcul est dû à l'utilisation de l'algorithme du simplexe dual. En effet, celui-ci est bien plus efficace que l'algorithme du simplexe primal pour résoudre les relaxations linéaires du problème LESC.

Il est important de noter que même si le problème LESC est un problème intéressant pour tester IPS car il contient beaucoup de dégénérescence, les caractéristiques de son dual font que même avec les améliorations qu'apporte IPS par rapport à l'algorithme du simplexe primal, ces deux méthodes restent toujours moins adaptées que les méthodes exploitant le dual du problème (les méthodes vues dans la revue de littérature et l'algorithme du simplexe dual). Ce phénomène s'observe aussi bien dans les courbes de la section 4.2.2 qui montrent la supériorité de l'algorithme du simplexe dual, que dans la colonne *Cplex défaut* du tableau 4.3 où l'on peut observer que cette efficacité se répercute logiquement dans la résolution en nombres entiers.

Chapitre 5

CONCLUSION

Nous avons exploré une nouvelle façon de résoudre de manière exacte le problème LESC : dans un premier temps, une première solution proche de la solution optimale est construite à l'aide d'un algorithme glouton suivi d'une recherche tabou, et dans un deuxième temps, cette solution initiale est donnée comme point de départ pour l'algorithme IPS qui termine la résolution jusqu'à l'optimalité. La recherche tabou donne de bons résultats comme le montrent les tests effectués sur une librairie d'instances, et les temps de calculs sont courts grâce aux optimisations qui ont été apportées telles que le calcul incrémental.

Cette solution initiale, qui permet à IPS de terminer la résolution de la relaxation linéaire plus rapidement, nous a permis d'observer que la méthode proposée permettait des réductions importantes du temps de calcul pour la résolution des relaxations linéaires des instances de densité comprise dans l'intervalle [0,1 ; 0,3]. Ces réductions atteignent un facteur de 3,5 dans certains cas. Il a également été remarqué que cette efficacité augmentait avec la taille des instances.

Ensuite, pour aboutir à la solution optimale du problème en nombres entiers, nous avons testé l'imbrication d'IPS dans deux types d'algorithmes d'évaluation et séparation : le premier utilisant une stratégie profondeur d'abord et le deuxième utilisant une stratégie meilleur d'abord. Il a été vu que la statégie meilleur d'abord était la plus efficace avec une réduction du nombre de nœuds de branchement évaluée à un facteur 10. On a cependant remarqué que l'algorithme d'évaluation et séparation tel qu'il est implanté actuellement dans IPS ne supplante pas l'algorithme du simplexe primal classique imbriqué dans un même algorithme d'évaluation et séparation à cause de l'impossibilité de faire des réductions sur les problèmes résolus après le premier nœud de branchement.

Enfin, nous avons vu que la méthode IPS et l'algorithme du simplexe primal ne sont pas les méthodes les plus adaptées pour résoudre le problème LESC de manière exacte. En effet, les méthodes utilisant le dual de ce problème telles que l'algorithme

du simplexe dual sont bien plus efficaces.

Il serait donc intéressant pour des améliorations futures de réussir à faire en sorte que les réductions de problème soient possibles sur tous les nœuds de branchement de l'algorithme d'évaluation et séparation utilisant IPS, ce qui devrait permettre à la méthode globale de conserver son avantage sur la résolution des instances de densité comprise entre 0,1 et 0,6. Ensuite, il pourrait être intéressant d'appliquer ce nouvel algorithme utilisant IPS au dual du problème qui possède une structure plus simple que celle du primal et qui pourrait peut-être permettre de battre les autres algorithmes de résolution exacte actuels.

Références

ALVES, M. et ALMEIDA, M. (1992). Simulated annealing algorithm for simple plant location problem. *Revista investigacion*, pp. 12.

BALINSKI, M. (1965). Integer programming : methods, uses computation. *Management Science*, vol. 12, pp. 254–313.

BELTRAN-ROYO, C., VIAL, J. et ALONSO-AYUSO, A. (2007). Solving the uncapacitated facility location problem with semi-Lagrangian relaxation. *Statistics and Operations Research*, pp. 1–23.

BILDE, O. et KRARUP, J. (1977). Sharp lower bounds and efficient algorithms for the simple plant location problem. *Annals of Discrete Mathematics*, vol. 3, pp. 79–97.

CONN, A. et CORNUEJOLS, G. (1990). A projection method for the uncapacitated facility location problem. *Mathematical Programming*, vol. 46, pp. 273–298.

CORNUEJOLS, G., NEMHAUSER, G. et WOLSEY, L. (1990). The uncapacitated facility location problem. *Discrete location theory*, vol. 3, pp. 119–171.

EFROYMSON, M. et RAY, T. (1966). A branch-bound algorithm for plant location. *Operations Research*, vol. 14, pp. 361–368.

ELHALLAOUI, I., METRANE, A., SOUMIS, F. et DESAULNIERS, G. (2007). An Improved Primal Simplex Algorithm for Degenerate Linear Programs. *G-2007-66, Les cahiers du GERAD*.

ERLENKOTTER, D. (1978). A dual-based procedure for uncapacitated facility location. *Operations Research*, vol. 26, pp. 992–1009.

FELDMAN, E., LEHRER, F. et RAY, T. (1966). Warehouse location under continuous economies of scale. *Management Science*, vol. 12, pp. 670–684.

GALVAO, R. (2004). Uncapacitated facility location problems : contributions. *Pesquisa Operacional*, vol. 24, pp. 7–38.

GALVAO, R. et RAGGI, L. (1989). A method for solving to optimality uncapacitated location problems. *Annals of Operations Research*, vol. 18, pp. 225–244.

GHOSH, D. (2003). Neighborhood search heuristics for the uncapacitated facility location problem. *European Journal of Operational Research*, vol. 150, pp. 150–162.

GLOVER, F. (1986). Future paths for integer programming and linkage to artificial intelligence. *Computers & Operations Research*, vol. 13, pp. 533–549.

GOLDENGORIN, B., G. A.TIJSSEN, GHOSH, D. et SIERKSMA, G. (2003). A data correctiong algorithm for the simple plant location problem. *Journal of Global Optimization*, vol. 25, pp. 377–406.

GOLDENGORIN, B., GHOSH, D. et SIERKSMA, G. (2004). Branch ans peg algorithms for the simple plant location problem. *Computers & Operations Research*, vol. 31, pp. 241–255.

GUIGNARD, M. (2003). Lagrangian relaxation. *TOP (Journal of the Spanish Society of Statistics and Operations Research)*, vol. 11, pp. 151–228.

GUNER, A. et SEVKLI, M. (2008). A discrete particle swarm optimization algorithm for uncapacitated facility location problem. *Journal of Artificial Evolution and Applications*.

HANSEN, P., BRIMBERG, J., UROSEVIC, D. et MLADENOVIC, N. (2007). Primal-dual variable neighborhood search for the simple plant location problem. *Informs journal on computing*, vol. 19, pp. 552–564.

HARM, G. et HENTENRYCK, P. (2005). *A multistart variable neighborhood search for uncapacitated facility location*. MIC2005 : The sixth metaheuristics international conference. Vienna.

HOEFER, M. (2003). Experimental comparison of heuristic and approximation algorithms for uncapacitated facility location. *the second international workshop on experimental and efficient algorithms (WEA 2003)*. pp. 165–178.

HOEFER, M. (2004). *UFLlib*. http ://www.mpi-inf.mpg.de/departments/d1-/projects/benchmarks/UflLib/.

JANACEK, J. et BUZNA, L. (2008). An acceleration of Erlenkotter-Korkel's algorithms for the uncapacitated facility location problem. *Annals of Operations Research*, vol. 164, pp. 97–109.

JANACEK, J. et KOVACIKOVA, J. (1997). Exact solution techniques for large location problems. *the International Conference : Mathematical Methods in Economics*. pp. 80–84.

KHUMAWALA, B. (1972). An efficient branch and bound algorithm for the warehouse location problem. *Management Science*, vol. 18, pp. 718–731.

KOCHETOV, Y. et IVANENKO, D. (2003). Computationally difficult instances for the uncapacitated facility location problem. *the 5th Metaheuristics International Conference (MIC)*. pp. 41 :1–41 :6.

KORKEL, M. (1989). On then exact solution of large-scale simple plant location problem. *European Journal of Operational Research*, vol. 39, pp. 157–173.

KRATICA, J., TOSIC, D., FILIPOVIC, V. et LJUBIC, I. (2001). Solving the simple plant location problem by genetic algorithm. *RAITO Operations Research*, vol. 35, pp. 127–142.

KUEHN, A. et HAMBURGER, M. (1963). A heuristic program for locating warehouses. *Management Science*, vol. 9, pp. 643–666.

MANNE, A. (1964). Plant location under economies-of-scale-decentralization and computation. *Management Science*, vol. 11, pp. 213–235.

MELO, M., NICKEL, S. et DA GAMA, F. S. (2009). Facility location and supply chain management - A review. *European Journal of Operational Research*, vol. 196, pp. 401–412.

MICHEL, L. et HENTENRYCK, P. (2004). A simple tabu search for warehouse location. *European Journal of Operational Research*, vol. 157, pp. 576–591.

RAYMOND, V., SOUMIS, F. et METRANE, A. (2009). Improved Primal Simplex Version 3 : Cold Start, Generalization for Bounded Variable Problems and a New Implementation. *G-2009-15, Les cahiers du GERAD*, vol. 15.

RAYMOND, V., SOUMIS, F. et ORBAN, D. (2010). A new version of the Improved Primal Simplex for degenerate linear programs. *Computers & Operations Research*, vol. 37, pp. 91–98.

RESENDE, M. et WERNECK, R. (2006). A hybrid multistart heuristic for the uncapacitated facility location problem. *European Journal of Operational Research*, vol. 174, pp. 54–68.

REVELLE, C. et EISELT, H. (2005). Location analysis : A synthesis and survey. *European Journal of Operational Research*, vol. 165, pp. 1–19.

ROODMAN, G. et SCHWARZ, L. (1975). Optimal and heuristic facility phase out strategies. *AIIE Transactions*, vol. 7, pp. 177–184.

SULTAN, K. et FAWZAN, M. (1999). A tabu search approach to the uncapacitated facility location problem. *Annals of Operations Research*, vol. 86, pp. 91–103.

SUN, M. (2006). Solving the uncapacitated facility location problem using tabu search. *Computers & Operations Research*, vol. 33, pp. 2563–2589.

TEITZ, M. et BART, P. (1968). Heuristic methods for estimating the generalized vertex median of a weighted graph. *Operations Research*, vol. 16, pp. 955–961.

VAN ROY, T. et ERLENKOTTER, D. (1982). A dual-based procedure for dynamic facility location. *Management Science*, vol. 28, pp. 1091–1105.

WARSZAWSKI, A. (1973). Multi-dimensional location problems. *Operational Research Quarterly*, vol. 24, pp. 165–179.