Résolution du problème du p-médian, application à la restructuration de bases de données semi-structurées

Jean-Christophe Gay

N° d'ordre : D.U. : 2171
EDSPIC : 536

Université Blaise Pascal - Clermont II

Ecole Doctorale
Sciences Pour l'Ingénieur de Clermont-Ferrand

THÈSE

présentée par

Jean-Christophe GAY

pour obtenir le grade de

Docteur d'Université

Spécialité : Informatique

Résolution du Problème du p-médian, Application à la Restructuration de Bases de Données Semi-Structurées

Soutenue publiquement le 19 octobre 2011
devant le jury composé de :

Président :
Alain QUILLOT — Professeur des Universités, Université Blaise Pascal, LIMOS

Rapporteurs :
A. Ridha MAHJOUB — Professeur des Universités, Université Paris Dauphine, LAMSADE
François VANDERBECK — Professeur des Universités, Université Bordeau 1, IMB

Examinateurs :
Fatiha BENDALI — Maître de Conférence, Université Blaise Pascal, LIMOS
Farouk TOUMANI — Professeur des Universités, Université Blaise Pascal, LIMOS
Eric GOURDIN — Chercheur chez Orange Labs R&D

Directeur de thèse :
Mourad BAÏOU — Chargé de Recherche CR1 CNRS, LIMOS

Remerciements

Je tiens tout d'abord à remercier mon directeur de thèse, Mourad Baïou, pour m'avoir encouragé tout au long de cette thèse. Il a su me motiver quand il fallait et me faire découvrir la recherche.

Je remercie Ridha Mahjoub, François Vanderbeck pour avoir pris le temps de relire ma thèse ainsi que pour leurs conseils et les différentes corrections qu'ils ont pu apporter.

Je remercie Fatiha Bendali, Farouk Toumani et Eric Gourdin pour avoir accepté de participer à mon jury de thèse et pour leurs remarques pertinentes qui ont permis l'amélioration de ce manuscrit.

Je remercie mes collègues et amis (en particulier Romain, Cédric, Frédérique, Olivier, Hélène, Ren, Heitor, Raksmey, Virginie, Christophe, Pascale... pour ne citer qu'eux) pour tous les moments de détente que l'on a pu passer ensemble et leurs conseils.

Enfin je tiens à remercier ma conjointe pour m'avoir toujours encouragé et pour le soutient moral qu'elle a su m'apporter pendant toute la durée de cette thèse.

Résumé

Les problèmes que nous considérons dans cette thèse sont de nature combinatoire. Notre principal intérêt est le problème de restructuration de données semi-structurées. Par exemple des données stockées sous la forme d'un fichier XML sont des données semi-structurées. Ce problème peut être ramené à une instance du problème du p-médian. Le principal obstacle ici est la taille des instances qui peut devenir très grande. Certaines instances peuvent avoir jusqu'à 10000 ou 20000 sommets, ce qui implique plusieurs centaines de millions de variables. Pour ces instances, résoudre ne serait-ce que la relaxation linéaire du problème est très difficile. Lors d'expériences préliminaires nous nous sommes rendu compte que CPLEX peut résoudre des instances avec 1000 sommets dans des temps raisonnables. Mais pour des instances de 5000 sommets, il peut prendre jusqu'à 14 jours pour résoudre uniquement la relaxation linéaire. Pour ces raisons nous ne pouvons utiliser de méthodes qui considère la résolution de la relaxation linéaire comme une opération de base, comme par exemple les méthodes de coupes et de branchements. Au lieu d'utiliser CPLEX nous utilisons une implémentation parallèle (utilisant 32 processeurs) de l'algorithme du Volume. L'instance pour laquelle CPLEX demande 14 heures est résolue en 24 minutes par l'implémentation séquentielle et en 10 minutes par l'implémentation parallèle de l'algorithme du Volume.

La solution de la relaxation linéaire est utilisée pour construire une solution réalisable, grâce à l'application d'une heuristique de construction gloutonne puis d'une recherche locale. Nous obtenons des résultats comparables aux résultats obtenus par les meilleures heuristiques connues à ce jour, qui utilisent beaucoup plus de mémoire et réalisent beaucoup plus d'opérations. La mémoire est importante dans notre cas, puisque nous travaillons sur des données de très grandes tailles.

Nous étudions le dominant du polytope associé au problème du p-médian. Nous discutons de sa relaxation linéaire ainsi que de sa caractérisation polyédrale. Enfin, nous considérons une version plus réaliste du problème de restructuration de données semi-structurées. Grosso modo, nous ajoutons au problème du p-médian original des nouveaux sommets s'ils aident à réduire le coût global des affectations.

Mots-clés : p-médian, algorithme du Volume, relaxation linéaire, implémentation parallèle, base de donnée semi-structurée.

Abstract

The problems we consider in this thesis are of combinatorial nature. Our main interest is the problem of approximating typing of a semistructured data. For example XML is a semistructured data. This problem may be reduced to an instance of the p-median problem. The main obstacle here is the size of the instances that may be very huge, about 10000 and 20000 nodes which imply several hundreds of million variables. For these instances, even solving the linear relaxation is a hard task. In some preliminary results we noticed that Cplex may solve instances of size 1000 in an acceptable time. But for some instances having 5000 nodes, it may needs 14 days for solving only the linear relaxation. Therefore, we cannot use methods that consider the linear relaxation as an elementary operation, as for example branch-and-cut methods. Instead of using Cplex we use the Volume algorithm in a parallel implementation (32 processors). For the instance where the Cplex needs 14 hours, the Volume algorithm in sequential implementation needs 24 minutes and in parallel implementation it needs 10 minutes.

The solution of the linear relaxation is used to produce a feasible solution by first applying a greedy and then a local search heuristic. We notice that the results we obtain are relatively the same as those given by the best method known up today, which produces more effort and consumes more memory. Memory is important in our case since the data we consider are huge.

We study the dominant of the polytope associated with the p-median problem. We discuss linear relaxation and a polyhedral characterization. Finally, we consider a more realistic version of the p-median problem when applied to the problem of approximating typing of a semistructured data. Roughly speaking, we add new nodes to the underlying graph if this help to reduce the overall cost.

Keywords : p-median, Volume algorithm, linear relaxation, parallel implementation, semi-structured database.

Table des matières

Liste des Figures

Liste des Tableaux

Liste des Algorithmes

Introduction

Dans cette thèse, nous nous intéressons au problème de restructuration de données semi-structurées. Nous montrerons comment ce problème peut se réduire à une instance du problème du p-median et comment le résoudre pour des instances de très grandes tailles. Le problème du p-médian est un problème d'optimisation combinatoire, c'est-à-dire qu'il peut se formuler par un programme linéaire en nombres entiers. La différence entre les programmes linéaires classiques et les programmes linéaires en nombres entiers est que, dans cette deuxième classe, les variables de la solution optimale doivent être entières alors que, dans les programmes classiques, nous n'avons pas cette restriction.

Le problème du p-median est une variante de la classe des problèmes de localisation dans le domaine de l'optimisation combinatoire. Malgré une riche littérature dans ce domaine et les nombreuses méthodes développées pour résoudre ce genre de problème, il n'existe pas de méthode pour résoudre des instances de très grandes tailles. Il est même difficile de résoudre la relaxation linéaire classique associée à ce problème. Bien entendu, plusieurs heuristiques ont été développées et donnent des résultats satisfaisants, mais sans une évaluation de la borne inférieure (nous traitons un problème de minimisation) les résultats des heuristiques qui ne se basent pas sur la résolution de la relaxation linéaire ne peuvent pas être interprétés. Une des bornes inférieures connues dans la littérature est la valeur de la solution optimale de la relaxation linéaire, bien entendu, on pourra l'améliorer avec l'ajout de contraintes valides. Une raison de plus qui rend indispensable la résolution de cette relaxation linéaire, et il est crucial de pouvoir la résoudre dans des temps raisonnables.

Etant donné un graphe dirigé $G = (V, A)$ et des coûts $c(u, v)$ associés à chaque arc (u, v), le problème du *p-médian* (pMP) consiste en la sélection de p sommets appelés des *centres* et l'affectation des sommets non sélectionnés aux centres tout en minimisant la somme totale de l'affectation. Ce problème est NP-Complet même lorsque la fonction coût c est une métrique [49, 70, 87, 80]. Il est polynomial dans certaines classes de graphes comme les arbres [70] où la fonction coût a une certaine forme. Il est aussi polynomial dans des classes de graphes définies par des structures interdites [8].

La relaxation linéaire classique du problème du p-median associé au graphe G, noté par RLpMP(G), est la suivante :

$$\mathrm{RL}p\mathrm{MP}(G) = \begin{cases} \text{minimiser} \displaystyle\sum_{(u,v)\in A} c(u,v)x(u,v), & \\ \displaystyle\sum_{v\in V} y(v) = p, & \\ \displaystyle\sum_{(u,v)\in A} x(u,v) + y(u) = 1 & \forall\ u \in V, \\ x(u,v) \leq y(v) & \forall (u,v) \in A, \\ x(u,v) \geq 0 & \forall (u,v) \in A, \\ y(u) \geq 0 & \forall u \in V. \end{cases}$$

Les instances qui nous intéressent contiennent plusieurs centaines de millions de variables. Et, compte tenu de la dégénérescence naturelle du problème, les solveurs commerciaux basés sur la méthode du simplexe ne peuvent pas résoudre de telles instances. Une des méthodes utilisées pour résoudre cette relaxation linéaire est la relaxation Lagrangienne [38, 41, 17, 5, 22] qui s'avère très performante, mais le nombre de variables dans les instances traitées est moins important que les centaines de millions de variables que nous voulons traiter. De plus, la relaxation Lagrangienne est utilisée comme outil pour obtenir une borne inférieure et non pour obtenir la solution de la relaxation linéaire. Dans cette thèse, nous utilisons l'algorithme du Volume [2] pour obtenir une solution approchée de la solution optimale de RLpMP(G). Cet algorithme utilise la relaxation Lagrangienne et la méthode du sous-gradient pour maximiser la fonction Lagrangienne de la relaxation. À chaque étape de la méthode du sous-gradient, l'algorithme du Volume produit une solution primale par des combinaisons convexes des différentes solutions du sous-problème Lagrangien. Les coefficients de cette combinaison sont mis à jour à chaque itération. L'algorithme s'arrête quand la violation des contraintes relaxées est négligeable. Nous verrons que l'algorithme du Volume résoud des instances intraitables par CPLEX, mais les temps d'exécution peuvent être importants. Pour les réduire, nous proposons une parallélisation de l'algorithme du Volume, nous avons noté que la diminution du temps est fortement liée au nombre de processeurs de la machine. Plusieurs expérimentations sont menées pour évaluer la performance et la comparaison de plusieurs heuristiques.

Nous nous intéresserons également au problème de localisation de dépôts sans capacités (LDSC), un problème lié au problème du p-median. Dans le problème LDSC, le nombre de centres à ouvrir n'est pas fixé à l'avance et chaque ouverture entraine un coût fixe. Donc, le problème LDSC est défini par la donnée d'un graphe dirigé $G = (V, A)$, un coût $c(u, v)$ pour chaque arc $(u, v) \in A$ et un coût $f(u)$ pour chaque sommet $u \in V$. La relaxation linéaire classique du problème LDSC, notée par LDSC(G), est la suivante:

$$\mathrm{LDSC}(G) = \begin{cases} \text{minimiser} \displaystyle\sum_{(u,v)\in A} c(u,v)x(u,v) + \sum_{u\in V} f(u)y(u), & \\ \displaystyle\sum_{(u,v)\in A} x(u,v) + y(u) = 1 & \forall u \in V, \\ x(u,v) \leq y(v) & \forall (u,v) \in A, \\ x(u,v) \geq 0 & \forall (u,v) \in A, \\ y(u) \geq 0 & \forall u \in V. \end{cases}$$

Comme pour le problème du p-médian la résolution de LDSC(G) par CPLEX est difficile pour les instances de très grandes tailles, même si le problème est un peu plus facile que celui du p-médian. L'algorithme du Volume a été appliqué dans [12] pour résoudre LDSC(G). Des modifications de la méthode du simplexe pour traiter les contraintes $x(u, v) \leq y(v)$ séparément des autres contraintes ont été proposées dans [95, 101], alors que dans [83] ces contraintes sont traitées comme des coupes et sont ajoutées au fur et à mesure quand c'est nécessaire. Nous donnons la description du dominant du polytope de la relaxation linéaire LDSC(G) et nous montrons la similitude entre l'optimisation sur ce dominant et la méthode dans [83].

Les problèmes de localisation proches de la forme étudiée dans cette thèse ont été introduits dans les années 50, l'application typique est la distribution d'un certain bien depuis des entrepôts potentiels aux clients au moindre coût [16]. Un entrepôt potentiel signifie que la construction d'un entrepôt dans un site, qui gènère un certain coût pris en compte dans le coût total, n'est effective que s'il contribue à la solution optimale. Donc, si nous connaissons les entrepôts de la solution optimale, les seuls coûts qui restent variables sont ceux de livraison (transport) aux clients qui est un problème facile à résoudre. Dans sa version originale, les entrepôts peuvent avoir des capacités et les clients des demandes. La version plus simple (sans capacités) formulée par le programme linéaire en nombres entiers LDSC(G) auquel nous ajoutons l'exigence sur l'intégrité des variables a été initialement rencontrée dans [77, 11]. D'autres types de problèmes de localisation ont été traités à savoir quand le site d'installation d'un entrepôt n'est pas connu et se situe dans une région "continue" (voir [47]). Dans cette thèse nous étudions la version dans laquelle les entrepôts se situent dans une région discrète (voir [82]).

Cette thèse est organisée comme suit. Dans le Chapitre 1, nous présenterons diverses applications du problème du p-médian ainsi que des méthodes utilisées pour le résoudre. Nous présenterons aussi, plus en détails, la formulation classique de ce problème et deux reformulations de la littérature.

Dans le Chapitre 2, nous introduirons le problème d'extraction de schéma d'une base de données semi-structurée. Nous discuterons de sa forme générale qui est intraitable et ensuite nous présenterons une version simplifiée et nous montrerons comment elle peut se réduire au problème du p-médian.

Le Chapitre 3 introduit l'algorithme du Volume et son application pour résoudre la relaxation linéaire RLpMP(G) du problème du p-médian. Nous discuterons des détails qui permettent une implémentation parallèlle de ce problème. Dans ce même chapitre, plusieurs expérimentations seront effectuées pour estimer la qualité de cette parallélisation et nous présenterons quelques heuristiques naturelles qui se basent sur la solution de RLpMP(G) pour donner une solution réalisable.

Enfin, dans le Chapitre 4, nous présentons le problème de localisation de dépôt et la caractérisation du dominant du polytope de sa relaxation linéaire, LDSC(G). Il sera discuté quelques cas où ce dominant défini un polyèdre entier. Des résultats expérimentaux seront donnés pour comparer la résolution de LDSC(G) et celle sur le dominant par la méthode du simplex. Dans ce même chapitre, nous présenterons des heuristiques permettant de résoudre une variation du problème de restructuration des bases de données semi-structurées.

Les notations et définitions seront données juste avant leur première introduction. Les notations suivent en général ce qui est souvent utilisé dans la

littérature.

Chapitre 1

Applications et Méthodes

1.1 Quelques applications

1.1.1 Le problème des centres de commutation dans les réseaux de télécommunication [54, 55]

L'information qui transite dans un réseau général doit passer par un certain nombre de commutateurs $S_1, \ldots, S_p$. Cette information peut être le contenu d'un message dans un réseau de télécommunication. Nous imaginons un réseau de télécommunication comme un graphe $G = (V, A)$, où les sommets représentent des centres téléphoniques et les arêtes des connexions potentielles (en fibre optique) entre ces centres téléphoniques. Chaque sommet v est associé avec un poids h_v qui désigne le nombre de fils (lignes) nécessaires pour connecter le sommet v à un commutateur, et chaque arête (u, v) désigne le coût unitaire d'utilisation de cette connexion. Il est important de noter que les commutateurs sont installés sur les sommets et/ou sur les arêtes. Le problème est de trouver l'installation qui minimise la somme totale de la longueur des lignes entre les sommets et les commutateurs.

La distance entre toute paire de sommets u et v, notée par $d(u, u)$, est la longueur de la plus courte chaîne dans G entre u et v. La longueur d'une chaîne est la somme des poids de ses arêtes. Soit X_p un ensemble de points $x_1, \ldots, x_p$ dans G qui ne sont pas nécessairement tous des sommets. Soit

$$d(v_i, X_p) = \min \{d(v_i, x_1), \ldots, d(v_i, x_p)\},$$

où les v_i sont précisément les sommets de G. Le problème revient à trouver un ensemble "p-médian" de points X_p^* tel que pour n'importe quel ensemble de points X_p de cardinalité p, nous avons :

$$\sum_{v_i \in V} h_i d(v_i, X_p^*) \leq \sum_{v_i \in V} h_i d(v_i, X_p).$$

Il n'est pas difficile de voir qu'il existe toujours un ensemble de points X_p^* qui correspond à un ensemble de sommets V_p^* de V. Mais la réciproque n'est pas vraie. Dans l'article [55], Hakimi propose une réduction au problème de couverture de sommets, ensuite il énumère toutes les solutions possibles de ce problème via les fonctions booléennes [78, 61]

1.1.2 Le problème des lieux d'ouverture de comptes bancaires [38]

Le nombre de jours nécessaires pour encaisser un chèque dans une banque située dans une ville i depend de la ville j où le chèque a été déposé. Par conséquent, une compagnie qui doit régler ses factures à plusieurs clients a intérêt à ouvrir des comptes dans plusieurs endroits stratégiques afin de maximiser ses fonds disponibles. Elle aimerait payer un client de la ville i à partir d'une banque d'une ville j qui maximise le nombre de jours nécessaires pour l'encaisser.

La formulation de ce problème est la suivante : soit $I = \{1, \ldots, m\}$ l'ensemble de sites où se situent les clients et $J = \{1, \ldots, n\}$ les sites potentiels où des comptes bancaires peuvent être ouverts. Soient d_j le coût fixe pour maintenir un compte au site j, f_i la valeur (en euros) totale des chèques à payer aux clients dans le site i et ϕ_{ij} est le nombre de jours nécessaires pour encaisser un chèque dans le site i et qui est déposé dans le site j. Soient

$$y_j = \begin{cases} 1 & \text{si un compte est maintenu dans le site } j, \\ 0 & \text{sinon,} \end{cases}$$

$$x_{ij} = \begin{cases} 1 & \text{si le client dans le site } i \text{ est payé depuis un compte dans le site } j, \\ 0 & \text{sinon.} \end{cases}$$

Donc le problème des lieux d'ouverture de comptes bancaires peut se formuler par le programme linéaire en nombres entiers suivant:

$$(LOCB) \begin{cases} \text{Maximiser} & \displaystyle\sum_{i\in I}\sum_{j\in J} c_{ij}x_{ij} - \sum_{j\in J} d_j y_j & \\ & \displaystyle\sum_{j\in J} x_{ij} = 1, & \forall i \in I \\ & \displaystyle 1 \leq \sum_{j} y_j \leq K, & \\ & 0 \leq x_{ij} \leq y_j \leq 1 & \forall i \in I, j \in J \\ & x_{ij}, y_j \in \{0,1\}, & \forall i \in I, j \in J. \end{cases}$$

1.1.3 Le problème des proxys web [75]

Un *proxy* est un programme servant d'intermédiaire pour accéder à un réseau, généralement Internet. Dans ce cas, le réseau est supposé être une arborescence T avec un sommet racine r. L'ensemble des sommets de T est noté par $V(T)$. Les arcs de T sont dirigés des feuilles vers la racine. Un sommet v de T fait une demande de service, comme la connexion à une adresse http. Cette demande se propage du sommet v vers la racine à travers l'arborescence. Cette propagation s'arrête quand un proxy est rencontré. Le problème revient à placer un nombre fixe de proxys dans les sommets de T de sorte que la distance parcourue par les demandes faites au niveau de chaque sommet soit minimale.

Chaque sommet v de T a un poids $w(v)$ qui correspond en général à la fréquence des demandes passées depuis ce sommet. Chaque arc (u, v) est associé avec une distance $d(u, v)$ qui prend en compte le trafic et la congestion observée sur cet arc. La définition de la distance peut être étendue d'un sommet à un de ses ancêtres et pas seulement d'un sommet à son père. C'est-à-dire que la distance d'un sommet u à v est la somme des distances des arcs composants le chemin de u à v (v est un ancêtre de u donc il est sur le chemin de u vers la racine

r). Soit $V' \subseteq V(T)$ un sous-ensemble de sommets de T, si v est un sommet de T, v' désigne l'ancêtre le plus proche de v dans V'. Donc, le problème revient à définir les p proxys qui minimisent $\{c(T, V') : V' \subseteq V(T), |V'| = p\}$, où

$$c(T, V') = \sum_{v \in V(T)} w(v)d(v, v').$$

La différence entre ce modèle et le problème du p-médian classique pMP introduit plus haut, réside dans l'affectation des sommets non-sélectionnés aux centres. Dans le modèle classique, l'affectation se fait d'un voisin à un autre, alors que, dans ce modèle, elle se fait d'un sommet u à un sommet v même s'ils ne sont pas voisins, mais il faut qu'ils se trouvent dans le chemin unique de u vers t la racine de l'arborescence.

Un algorithme de complexité $O(p^2n^2)$, où $n = |V(T)|$, a été donné dans [70]. Un autre algorithme de complexité $O(pn^3)$ a été developpé par Hsu [66]. Par l'application de la programmation dynamique un algorithme de complexité $O(pn^2)$ est présenté dans [98]. Plus récemment, nous relevons un algorithme en $O(Pp^2)$ [103] basé sur la programmation dynamique, où P désigne la somme du nombre d'ancêtres de chaque sommet de l'arborescence. D'autres algorithmes basés sur les mêmes techniques existent pour des cas plus simples. Dans le cas d'une chaîne, il existe un algorithme en $O(pn)$ [60]. Dans le cas d'arbres, pour $p = 2$ et $p = 3$, des algorithmes en $O(n \log n)$ et $O(n \log^2 n)$ ont été développés par Chrobak et al. [33].

1.2 Méthodes

La définition du problème du p-médian et sa relaxation linéaire

La donnée est une constante p et un graphe dirigé $G = (V, A)$, où chaque arc $(u, v) \in A$ est associé avec un poids $d(u, v)$ (qui, en général et dans plusieurs applications, est une métrique) et chaque sommet $v \in V$ est associé avec un poids $w(v)$. Etant donné un sous-ensemble $S \subseteq V$ et $v \in V \setminus S$, le sommet S_v est défini comme suit :

$$S_v = argmin_{u \in S}\{w(v)d(v, u)\}.$$

Noter que pour deux sommets différents v et v' dans $V \setminus S$, S_v et S'_v peuvent coïncider. Les sommets dans S sont appelés *médians* ou *centres*.

Le problème du p-médian (classique) pMP revient à déterminer un sous-ensemble $S \subseteq V$, $|S| = p$, qui minimise $\sum_{v \in V \setminus S} w(v)d(u, S_v)$.

Quand on écrit pMP(G), c'est pour préciser que le problème du p-médian est associé au graphe G où, au préalable, nous avons défini les poids des arcs et des sommets. Le problème pMP(G) peut ainsi se formuler par le programme linéaire en nombres entiers suivant :

$$Z(p\text{MP}(G)) = \text{minimiser} \sum_{(u,v)\in A} w(u)d(u,v)x(u,v) \quad (1.1)$$

$$\sum_{v\in V} y(v) = p, \quad (1.2)$$

$$\sum_{v:(u,v)\in A} x(u,v) = 1 - y(u) \quad \forall u \in V, \quad (1.3)$$

$$x(u,v) \leq y(v) \quad \forall (u,v) \in A, \quad (1.4)$$

$$y(v) \geq 0 \quad \forall v \in V, \quad (1.5)$$

$$x(u,v) \geq 0 \quad \forall (u,v) \in A, \quad (1.6)$$

$$x \in \{0,1\}^{|A|},\ y \in \{0,1\}^{|V|}. \quad (1.7)$$

La variable $y(u)$ prend la valeur 1, si le sommet u est sélectionné c'est-à-dire qu'il est considéré comme un médian, et 0 sinon. La variable $x(u,v)$ prend la valeur 1 quand un sommet u non-sélectionné est affecté au médian v. Donc l'égalité (1.2) impose la sélection d'exactement p médians. L'addition des inégalités (1.4) qui indiquent qu'aucun sommet ne peut être affecté à un sommet qui n'est pas un médian, implique que, dans les égalités (1.3), si u n'est pas un médian alors il doit être affecté à un médian.

Nous donnons maintenant une définition du problème du k-centres pour permettre aux lecteurs de différencier les deux problèmes k-centres et p-médian. Même si parfois nous appelons les médians des centres, les deux problèmes sont différents.

Le problème du k-centres. *La donnée du problème et l'ensemble des solutions réalisables sont les mêmes que pour le problème du p-médian ci-dessus. Par contre, nous cherchons un sous-ensemble $S \subseteq V$, $|S| = k$ qui minimise la fonction objective suivante:*

$$max\{min_{u\in S}\{w(v)d(u,v)\};\ v \in V \setminus S\}.$$

Relaxation Lagrangienne du p-médian

Avant de rentrer dans les détails de certaines méthodes, nous allons d'abord présenter la relaxation Lagrangienne (une méthode duale) appliquée au pMP initialement appliquée pour ce problème dans [50]. Si nous dualisons les égalités (1.3) et nous notons par π le vecteur des multiplicateurs duaux, généralement appelés les multiplicateurs de Lagrange, le sous-problème Lagrangien s'écrira comme suit :

$$L(\pi) = \text{Minimiser} \sum_{(u,v)\in A} \bar{c}(u,v)x(u,v) - \sum_{u\in V} \pi(u)y(u) + \sum_{u\in V} \pi(u) \quad (1.8)$$

$$\sum_{u \in V} y(u) = p, \tag{1.9}$$

$$x(u,v) \leq y(v) \quad \text{for all } (u,v) \in A, \tag{1.10}$$

$$0 \leq y(u) \leq 1 \qquad \text{for all } u \in V, \tag{1.11}$$

$$x(u,v) \geq 0 \quad \text{for all } (u,v) \in A. \tag{1.12}$$

où $\bar{c}(u,v) = w(u)d(u,v) - \pi(u)$.

Le calcul de $L(\pi)$ est facile [18] et se fait comme suit : pour chaque sommet $v \in V$, nous calculons :

$$\mu(v) = \sum_{(u,v):\bar{c}(u,v)\leq 0} \bar{c}(u,v) - \pi(v).$$

Ensuite, nous choisissons les p plus petites valeurs $\mu(v)$ pour $v \in V$, appelons les $\mu(v_1), \ldots, \mu(v_p)$. Alors la solution optimale de (1.8)-(1.12) est obtenue comme suit : $y(v_i) = 1$ pour $i = 1, \ldots, p$; $x(u, v_i) = 1$ si $\bar{c}(u, v_i) \leq 0$, pour chaque arc $(u, v_i) \in A$, avec $u \in V \setminus \{v_1, \ldots, v_p\}$, $i = 1 \ldots, p$. Toutes les autres variables prennent la valeur 0.

Il est clair que

$$\max_{\pi \in I\!R} \{L(\pi)\} \leq z(p\text{MP}(G)).$$

Le problème

$$L(\pi^*) = \max_{\pi \in I\!R} \{L(\pi)\} \tag{1.13}$$

se résout par une méthode itérative appelée la méthode du *sous-gradient*.

Algorithme 1 Forme générique de la méthode du sous-gradient

0. Poser $t = 0$; $\pi^t = 0$.
1. Calculer $L(\pi^t)$ en résolvant (1.8)-(1.12); soit (x^t, y^t) la solution optimale obtenue; définir v^t, où $v^t(u) = 1 - y^t(u) - \sum_{(u,v)\in A} x^t(u,v)$.
2. Si $\sum |v^t| \leq \epsilon$ ou que la valeur $L(\pi^t)$ ne change pas beaucoup après un certain nombre d'itérations ou que le nombre d'itérations a atteint une certaine limite, arrêter la procédure.
3. Poser $t = t + 1$; $\pi^t = \pi^{t-1} + s * v^{t-1}$, aller à l'étape 1.

Le pas s de l'avancement de cette méthode est une valeur empirique et elle est fixée selon des critères heuristiques qui peuvent changer d'un problème à un autre.

1.3 Heuristiques

Dans ce qui suit, nous présenterons les heuristiques qui ne sont pas basées sur la solution de la relaxation linéaire (1.1)-(1.6). En effet, les heuristiques basées sur cette relaxation linéaire n'ont jamais été considérées pour le problème du p-median où les instances contiennent plusieurs centaines de millions d'arcs, car

il est déjà assez coûteux de résoudre uniquement la relaxation linéaire. Neanmoins, nous trouvons des heuristiques utilisant la valeur de la relaxation linéaire du problème de localisation de dépôts [12]. Comme le principal travail de cette thèse est l'accéleration de la résolution de (1.1)-(1.6) pour les grandes tailles, nous avons également investi plusieurs heuristiques basées sur la solution de cette relaxation, ces heuristiques seront présentées dans la Section 3.4.

- **Heuristiques gloutonnes [73, 96, 69, 38, 105].** Les heuristiques gloutonnes sont les plus simples. Initialement, nous avons un médian v^*, choisi comme étant le meilleur parmi tous les sommets lorsque nous décidons de n'en sélectionner qu'un seul. La valeur de cette solution, appelée la solution courante est $\sum_{u \in V} w(u)d(u, v^*)$. L'heuristique gloutonne classique sélectionne un par un les médians jusqu'à ce qu'elle arrive à p médians. A chaque étape, elle sélectionne le médian qui procure la plus grande diminution de la valeur de la solution courante. D'autres variantes, dont la plupart sont basées sur certaines règles de choix aléatoires ont été présentées par Resende et Werneck [90]. Nous avons introduit une heuristique gloutonne basée sur la solution de la relaxation linéaire (1.1)-(1.6), cette heuristique sera détaillée dans la Section 3.4.4.

- **Heuristiques de recherche locale [77, 99, 65, 58, 59, 48, 89]**. Le principe de ces heuristiques est de sélectionner un ensemble S de cardinalité p de sommets et de les considérer comme médians. Ensuite pour chaque sommet $v \notin S$ on cherche un sommet u dans S, on évalue la valeur de la nouvelle solution induite par les médians $(S \backslash \{v\}) \cup \{u\}$, si on diminue la valeur de la solution courante alors S sera mis à jour, il devient $(S \setminus \{v\}) \cup \{u\}$. On continue ainsi de suite jusqu'à ce qu'aucune amélioration ne soit possible, en respectant toujours cette règle d'interchanger deux sommets. Ces heuristiques varient par le choix de la solution initiale et le choix de la règle d'interchanger les sommets, comme par exemple interchanger des sommets qui génèrent la plus grande diminution ou bien le faire dès qu'il est possible.

- **Heuristiques Lagrangiennes [18].** Ces heuritiques se basent sur la solution (x, y) donnée par le sous-problème Lagrangien (1.8)-(1.12). Comme nous l'avons vu ci-dessus, la solution (x, y) est entière, alors l'heuristique choisi comme médians les p sommets tel que $y(u) = 1$. Ensuite chaque sommet avec $y(u) = 0$ est affecté au plus proche sommet v avec $y(v) = 1$. Cette heuristique est rapide, elle est donc appliquée plusieurs fois durant le processus de la méthode du sous-gradient. La meilleure solution trouvée sera gardée comme étant la solution heuristique. Nous avons remarqué que cette heuritique ne produit pas de solutions de bonne qualité. La mesure de la qualité d'une solution est toujours comparée avec la borne duale (1.13).

- **Heuristique Hybride [90].** C'est l'une des meilleures heuristiques sur le plan des résultats empiriques. Bien entendu pour pouvoir évaluer sa performance nous avons besoin de la valeur de la borne dual (1.13) qui ne peut être obtenue que par la parallélisation de l'algorithme du Volume pour les très grandes instances, parallélisation qui sera détaillée dans

la Section 3.3. L'idée principale de cette heuristique est l'application intensive de la recherche locale en suivant la procédure du *chemin-reliant* "path-relinking". Soit S_1 et S_2 deux sous-ensembles de V, chacun de cardinalité p. La procédure du chemin-reliant de S_1 à S_2 transforme S_1 en S_2 en incluant à chaque étape un élément de $S_2 \setminus S_1$ à S_1, et elle fait sortir un élément de $S_1 \setminus S_2$. Nous pourrons également imaginer un chemin-reliant de S_2 à S_1 et appliquer l'idée inversement. La procédure chemin-reliant donne la meilleure solution trouvée en allant de S_1 à S_2. Dans [90], une autre stratégie est adoptée. Elle donne un minimum local, c'est-à-dire une solution qui est succédée et précédée, dans le chemin-reliant, par des solutions moins bonnes. Si un tel minimum n'existe pas, alors elle retourne une des solutions extrêmes.

Algorithme 2 Forme simplifiée de l'heuristique Hybride [90]

0. $it = 0$; Liste-Elite=$\emptyset$
1. Si $it \leq Max\_it$, soit $S \subseteq V$, $|S| = p$, généré aléatoirement; améliorer la valeur de S par recherche locale, S est l'ensemble des médians qui donne la meilleure solution.
2. Choisir S' dans Liste-Elite selon certaines règles.
3. Si S' existe, appliquer la procédure du chemin-reliant de S à S', soit S' la solution retournée; ajouter S' à Liste-Elite; $it = it + 1$; aller à l'Etape 1.
4. Appliquer une heuristique génétique où la génération initiale est Liste-Elite. Les générations futures seront créés par la procédure du chemin-reliant entre les solutions de la génération courante. La procédure s'arrête quand aucune amélioration n'est possible.

1.4 Algorithmes d'approximations

Soit (P) un problème d'optimisation combinatoire et soit z^* la valeur d'une solution optimale de (P). Soit A un algorithme qui nous donne une solution réalisable $z(A)$ pour le problème (P), c'est-à-dire une solution qui vérifie toutes les contraintes de ce problème, mais qui n'a pas forcément la meilleure valeur. Nous dirons que A est un algorithme α-*approximation* si $z(A) \leq \alpha z^*$, $\alpha > 1$, quand (P) est un problème de minimisation. Lorsque (P) est un problème de maximisation, A sera dit un alogorithme α-*approximation* si $z(A) \geq \alpha z^*$ avec $0 < \alpha < 1$. La valeur α est appelée le facteur d'approximation. Quand ce facteur est une constante, nous parlerons alors d'algorithmes d'approximations à facteur constant.

Le premier algorithme d'approximation à facteur constant a été présenté en 1999 dans [28, 30], c'est un algorithme $6\frac{2}{3}$-approximation. Auparavant, d'autres algorithmes étaient connus avec un facteur non-constant, il dépend du nombre de sommets de l'instance. Un algorithme $O(\log n \log\log n)$-approximation est donné par Bartal [14, 15]. Le même algorithme est amélioré par Charikar et al. [29] pour atteindre un facteur de $O(\log p \log\log p)$. Dans [76], un algorithme $(1+\epsilon)$-approximation a été développé, pour tout $\epsilon > 0$, mais le nombre de médians ouverts peut aller jusqu'à $(1+\frac{1}{\epsilon})(\ln n + 1)p$. Peu de temps après, le facteur constant de $6\frac{2}{3}$ a été ramené à 6 [67, 68]. Dans cet article, les auteurs donnent un algorithme d'approximation primal-dual pour le problème de local-

isation de dépôts ayant un facteur d'approximation égal à 3. En utilisant la relaxation Lagrangienne ils transforment ce facteur en 6 pour le problème du p-median. Ce facteur a été encore réduit à 4 par Charikar et Guha [27]. Finalement les algorithmes de recherche locale atteignent un facteur d'approximation égal $3+\frac{2}{s}$ [3] où s est le nombre de sommets qu'on autorise à interchanger à chaque itération. Bien entendu, tous les algorithmes cités ci-dessus sont polynomiaux, ces résultats ne tiennent que lorsque la fonction objectif est une métrique. Sauf le résultat dans [76] qui est vrai quand la fonction objectif est positive et quelconque ; rappelons que dans ce cas le nombre de médians peut largement dépasser p.

Concernant la complexité du pMP, il est connu qu'il est même NP-dur d'obtenir un algorithme $(1+\frac{2}{e}-\epsilon)$-approximation dans le cas métrique. Ci-dessous nous donnons la preuve qu'il est NP-dur de trouver un algorithme polynomial $(1+\epsilon)$-approximation, pour tout $\epsilon > 0$, dans le cas où la matrice des coûts est symétrique (ce qui est le cas des métrique). Un algorithme $(1+\epsilon)$-approximation est appelé un *schéma d'approximation.*

Théorème 1.4.1. *[76] Le problème qui consiste à trouver pour tout $\epsilon > 0$, une solution du p-médian de valeur au plus $1+\epsilon$ fois la valeur de l'optimum est NP-dur.*

Preuve. Une réduction à partir du problème du dominant taille p sera donnée.

Le problème du dominant de taille p. Etant donné un graphe non-dirigé $G=(V,E)$ et une constante p, existe-t-il un sous-ensemble $D_p \subseteq V$ de taille p tel que tout sommet $v \in V \setminus D_p$ est adjacent à au moins un sommet de D_p.

Etant donnés un graphe non-dirigé $G=(V,E)$, une constante $0 \leq p \leq |V|$ et $\epsilon > 0$, construire le graphe dirigé complet $G'=(V,A')$ comme suit : chaque arête uv de G est remplacée par deux arcs (u,v) and (v,u) chacune avec un coût $c(u,v)=c(v,u)=1$; pour chaque non-arête de uv de G, on ajoute deux arcs (u,v) et (v,u) de même coût $c(u,v)=c(v,u)=(1+\epsilon)(n-p)+1$. Donc, le graphe G' et les coûts c définissent une instance du problème du p-médian.

Si le graphe G admet un dominant de taille p, alors par définition la valeur de la solution optimale du p-médian sera $n-p$. Donc, un algorithme $(1+\epsilon)$-approximation délivre une solution de valeur inférieure ou égale à $(1+\epsilon)(n-p)$. Par contre, si G n'admet pas un dominant de taille p, alors n'importe quelle solution du p-médian enpruntera un arc de coût $(1+\epsilon)(n-p)+1$ (cet arc correspond à une non-arête de G). En résumé, si la valeur de la solution donnée par l'algorithme $(1+\epsilon)$-approximation est inférieure ou égale à $(1+\epsilon)(n-p)$, alors G admet un dominant de cardinalité p, sinon G n'admet pas un tel dominant. □

1.5 Approche polyédrale

Avant de présenter les résultats et la méthode basée sur la programmation linéaire, nous avons besoin d'introduire quelques défnitions.

- **Polyèdre.** Un polyèdre P est un ensemble de points de $\mathbb{R}^n$ satisfaisant un nombre fini d'inégalités linéaires, c'est-à-dire $P=\{x \in \mathbb{R}^n : \ Ax \geq b\}$.
- **Polytope.** Un polyèdre borné est appelé polytope.

- **Inégalité valide.** Soit P un polyèdre dans $\mathbb{R}^n$. Une inégalité $a^T x \geq \alpha$ est valide pour P si elle est vérifiée par chacun des points de P, $P \subseteq \{x \in \mathbb{R}^n : \ a^T x \geq \alpha\}$.

- **Combinaison convexe.** Soit $S = \{x_1, \ldots, x_k\}$ un ensemble de points de $\mathbb{R}^n$. Un point $x \in \mathbb{R}^n$ peut être obtenu par combinaison convexe des points de S s'il existe des scalaires positifs $\lambda_1, \ldots, \lambda_k$ tels que

$$\begin{aligned} x &= \sum_{i=1}^{k} \lambda_i x_i, \\ \sum_{i=1}^{k} \lambda_i &= 1, \\ \lambda_i &\geq 0, \quad \text{pour } i = 1, \ldots, k. \end{aligned}$$

- **Enveloppe convexe.** Soit $S = \{x_1, \ldots, x_k\}$ un ensemble de points de $\mathbb{R}^n$. L'enveloppe convexe des points de S notée $\text{conv}(S)$ est l'ensemble de tous les points de $\mathbb{R}^n$ pouvant être obtenus par combinaison convexes de points de S.

- **Dimension** Un polyèdre P dans $\mathbb{R}^n$ est de dimension k, noté $\dim(P) = k$, si le nombre maximum de points de P affinement indépendants est $k+1$.

- **Indépendance affine.** Soit $\{x_1, \ldots, x_k\}$ un ensemble de points de $\mathbb{R}^n$. Ces k points sont dits affinement indépendants si le système

$$\begin{aligned} \sum_{i=1}^{k} \lambda_i x_i &= 0, \\ \sum_{i=1}^{k} \lambda_i &= 0, \end{aligned}$$

  a comme solution unique $\lambda_i = 0$ pour $i = 1, \ldots, k$.

- **Face.** Soit $P = \{x \in \mathbb{R}^n : \ Ax \geq b\}$ un polyèdre de $\mathbb{R}^n$ et $a^T x \geq \alpha$ une inégalité valide pour P. L'ensemble $F = \{x \in P : \ a^T x = \alpha\}$ est dit une face de P.

- **Facette.** Une face F d'un polyèdre P est une facette de P si $\dim(F) = \dim(P) - 1$.

- **Point extrême.** Soit P un polyèdre. Une face de P de dimension 0 est un point extrême de P. Cette définition est équivalente à la propriété suivante : un point $x \in P$ est un point extrême de P s'il n'existe pas $x_1, x_2 \in P$, $x_1 \neq x_2$ tel que $x = \frac{1}{2}x_1 + \frac{1}{2}x_2$.

- **Problème de séparation.** Etant donné un système linéaire $Ax \leq b$ et un vecteur y, le problème de séparation associé à ce système et à y est de vérifier si y est une solution de $Ax \leq b$ et dans le cas contraire de trouver une contrainte de ce système violée par y.

La méthode de coupes et de branchements

C'est une méthode exacte et basée sur la donnée d'une relaxation linéaire du problème d'optimisation combinatoire en question. Cette relaxation linéaire peut contenir un nombre exponentiel, en fonction du nombre de variables, d'inégalités linéaires. Supposons que pour un problème d'optimisation combinatoire nous disposons d'une formulation du problème par un programme linéaire en nombres entiers contenant un certain nombre d'inégalités valides $Ax \geq b$, c'est-à-dire nous désirons résoudre le problème suivant:

$$\text{minimiser } c^T x, \tag{1.14}$$
$$Ax \geq b, \tag{1.15}$$
$$x \in \{0,1\}^n. \tag{1.16}$$

Soit P le polyèdre dont les sommets sont les solutions de (1.15)-(1.16). Si une description complète de P par un système d'inégalités linéaires est connue et si le problème de séparation associé à ce système est polynomial alors (1.14)-(1.16) peut être résolu en temps polynomial [72, 52]. L'inverse de cette assertion est aussi vrai [52, 71, 86]. Cette équivalence entre optimisation et séparation était à l'origine des méthodes de coupes et branchements utilisées pour résoudre des problèmes d'optimisation combinatoire.

Malhereusement, pour les problèmes NP-durs, on ne connait pas l'ensemble des inégalités permettant de résoudre le problème, c'est-à-dire $P \subset \{x \in I\!R^n : Ax \geq b\}$. En général le système $Ax \geq b$ contient un nombre important d'inégalités valides qui en pratique peuvent s'avérer suffisantes pour résoudre le problème à optimalité, ou dans le cas contraire elles produisent des bornes pour la valeur de la solution optimale.

L'algorithme de coupe et de branchements associé au problème (1.14)-(1.16) commence par résoudre le programme linéaire :

$$\text{minimiser } c^T x, \tag{1.17}$$
$$A_0 x \geq b_0. \tag{1.18}$$

où $A_0 x \geq b_0$ est un sous-système de $Ax \geq b$, contenant un nombre raisonnable de contraintes. Nous supposons savoir résoudre en temps polynomial le problème de séparation associé au système $Ax \geq b$. Soit x^* la solution de (1.17)-(1.18). L'algorithme résoud par la suite le problème de séparation associé à x^* et $Ax \geq b$. Si nous avons déterminé une inégalité $a^T x \geq \alpha$ parmi $Ax \geq b$, tel que $a^T x^* < b$, alors nous ajoutons $a^T x \geq \alpha$ au système $A_0 x \geq b_0$ et nous résolvons à nouveau (1.17)-(1.18). Nous répétons cette procédure jusqu'à ce que $Ax_0 \geq b$. Si la solution courante x^* est à composantes en 0-1, alors nous avons trouvé une solution optimale de (1.14)-(1.16). Sinon, l'algorithme fait un *branchement* sur une composante fractionnaire x_i^* en construisant deux nouveaux programmes obtenus à partir du programme courant, en ajoutant respectivement les égalités (non valides) $x_i = 1$ ou $x_i = 0$ et en procédant comme ci-dessus pour les polyèdres $P \cap \{x : \ x_i = 0\}$ et $P \cap \{x : \ x_i = 1\}$.

Cette méthode a été utilisée dans [4, 5]. Dans [4], Avella et Sassano ont donné une étude polyédrale du polytope du p-médian défini seulement par rapport

aux variables d'affectations et lorsque le graphe $G = (V, A)$ est un graphe dirigé complet. Nous notons, $|V| = n$ et donc $|A| = n(n-1)$. Nous pouvons facilement, dans ce cas, vérifier que la projection du polytope défini par les inégalités (1.2)-(1.6) donne le polytope suivant :

$$x(u,v) + \sum_{(v,t)\in A} x(v,t) \leq 1, \quad \forall (u,v) \in A, \tag{1.19}$$

$$\sum_{(u,v)\in A} x(u,v) = |V| - p, \tag{1.20}$$

$$x(u,v) \geq 0, \quad \forall (u,v) \in A. \tag{1.21}$$

Soit $Q_p(G)$ le polytope défini par les inégalités (1.19)-(1.21). Dans [4] l'enveloppe convexe des points dans $Q_p(G) \cap \{0,1\}^{|A|}$ a été notée par $M_{\bar{p}}(\overrightarrow{K}_n)$ et il a été montré que sa dimension est $n(n-1)-1$. Plusieurs facettes de $M_{\bar{p}}(\overrightarrow{K}_n)$ ont été introduites. Ces facettes sont issues du polytope des stables d'un graphe. Dans cet article, les auteurs ont appliqué l'algorithme de coupes sans le branchement sur quelques instances de petites tailles de la librairie OR [19]. Ils ont montré que les familles de facettes introduites peuvent suffir dans de nombreux cas pour obtenir la solution optimale. D'autres résultats plus poussés sur des instances de plus grandes tailles allant jusqu'à 5000 sommets ont été reportés dans [5]. Pour résoudre de telles instances, les auteurs ont eu recours à la génération de colonnes combinée dans un algortihme de coupes et branchements qui englobe plusieurs familles de contraintes valides. Sans la génération de colonnes, il est impossible de pratiquer un algorithme de coupes et de branchements sur des instances de tailles avoisinant les 3000 sommets. La génération de colonnes seule ne peut pas esquiver le problème de dégénérescence qui est la cause principale de la lenteur des logiciels commerciaux comme CPLEX. Le phénomène de dégénérescence apparaîtra lors de la génération de colonnes lorsque nous ajoutons des colonnes avec un coût réduit assez grand, la valeur de la relaxation linéaire ne change pas et rapidement nous nous trouverons avec un nombre ingérable de colonnes. Pour éviter ce phénomène, les auteurs fixent des limites sur les coûts réduits et quelques autres règles. Aussi, l'ensemble initial des colonnes est obtenu en résolvant le problème Lagrangien (1.13). Cette solution servira de base pour la sélection de l'ensemble de départ.

Dans [22], un problème industriel appelé "Optimal Diversity Management" a été formulé comme un problème du p-médian par Briant et Naddef. Dans cet article, des résultats très proches de l'optimum ont été reportés sur des instances allant jusqu'à 5500 sommets. Mais la structure particulière du problème implique des graphes assez creux, donc plus faciles à résoudre. Leur technique consiste à résoudre le problème Lagrangien (1.13), ils fixent ensuite des variables en exploitant l'information délivrée par les coûts réduits. Ainsi, la taille du problème est réduite en fixant quelques variables. Le nouveau problème obtenu est introduit dans CPLEX pour trouver une solution en 0-1. Nous voyons bien que cette technique se limite à des instances de tailles raisonnables.

D'autres types d'approches ont été considérés par exemple dans [39]. Ils introduisent de nouvelles variables pour exprimer le fait que chaque sommet doit être affecté au sommet le plus proche parmi ceux qui sont ouverts. Rappelons que cette considération est prise en compte dans la fonction objectif de

la formulation (1.1)-(1.7). Plus tard dans [46] cette formulation a été améliorée en exprimant les variables additionnelles par un produit de quelques autres variables. Ensuite une linéarisation est donnée pour obtenir une nouvelle formulation linéaire. D'autres travaux expriment l'affectation d'un sommet au plus proche médian par l'ajout d'inégalités linéaires [92, 104, 35, 51, 56, 44, 36, 57, 25, 20, 94, 100, 21, 79]. Nous pouvons écrire l'inégalité la plus simple qui exprime cette affectation comme suit :

$$\sum_{w:c(u,w)>c(u,v)} x(u,w) + y(v) \leq 1 \quad \forall (u,v) \in A. \tag{1.22}$$

Les variables x et y sont telles qu'elles étaient définies dans (1.1)-(1.7) et $c(i,j) = w(i)d(i,j)$.

Soit $MP_p(G)$ le polytope associé au problème p-médian. $MP_p(G)$ est l'enveloppe convexe de solutions en 0-1 qui vérifient (1.2)-(1.6). Peu de résultats polyédraux ont été trouvés concernant le polytope $MP_p(G)$. Dans [42, 43, 107], nous trouvons une étude faciale et plusieurs facettes du polytope $MP_p(G)$. Dans [6, 97] le polytope $MP_p(G)$ est complètement décrit quand le graphe G n'admet pas deux arcs entrants dans un sommet et un autre sortant de ce même sommet comme sous-graphe. Dans [9] les auteurs traitent la relation entre le polytope associé au problème de localisation de dépôts et le polytope $MP_p(G)$.

Les facettes du polytope associé au problème de localisation de dépôts ont été étudiées dans [53, 40, 31, 32, 26]. Toutes ces facettes peuvent dans certains cas définir des facettes de $MP_p(G)$, mais dans tous les cas elles sont valides pour $MP_p(G)$ et elles peuvent donc être utilisées dans n'importe quel algorithme de coupes et de branchements.

Chapitre 2

Bases de données semi-structurées et le problème du p-médian

Internet est encore aujourd'hui en pleine expansion, même s'il fait parti de notre quotidient. Ce média est devenu ces dernières années une source d'informations sur tous les sujets ou presque. Des quatres coins du monde on peut consulter une page web traitant par exemple des derniers films hollywoodiens, ou de la dernière théorie de physique à la mode. Toutes ces pages présentent de l'information qui est organisée, mise en page et présentée de manière à ce qu'un utilisateur puisse la comprendre et la parcourir simplement. L'organisation, la mise en page et la présentation d'une information prend du temps, et le temps est une ressource que l'on ne peut pas étendre à volonté. Au débuts du web les pages étaient statiques, pour chaque article une page était créée, remplie et mise en page par un rédacteur. Cette technique de présentation à rapidement été abandonnée pour une technique de rédaction alimentant un format d'affichage qui sera rempli lors de la génération de la page avec l'information requise. Le web statique a fait place au web dynamique.

De plus en plus de sites web utilisent ces techniques de génération de pages dynamiques en rapport avec l'information demandée par l'utilisateur. Ainsi sur un site de présentation de films il n'y a pas une page écrite pour chaque film, mais un format générique qui est rempli à chaque fois qu'un utilisateur souhaite consulter la page d'un film. Pour ce faire il faut avoir deux choses, un moyen technique de réaliser ceci et un moyen d'accéder aux données requises. Les moyens techniques developpés pour résoudre ce problème ne manquent pas grâce à des outils comme les CGI (Comon Gateway Interface) ou encore le PHP (PHP: Hypetext Preprocessor), mis rapidement en relation avec des bases de données pouvant être interrogées par un langage de type SQL (Structured Query Language ou langage de requête structuré). Ces deux outils ont permis le developpement de sites web complètement dynamiques dans lesquels chaque page est générée directement à la demande des utilisateurs.

Mais les bases de données se sont rapidement développées sur internet pour atteindre des tailles très importantes. Les requêtes complexes sur ces bases prennent de plus en plus de temps alors que le but principal est de présenter

des données et non de les interroger. De plus lorsque l'on manipule des données n'ayant pas toutes la même structure il devient difficile de les stocker dans des bases de données relationnelles classiques. Une nouvelle forme de base de données a donc fait son apparition : les bases de données semi-structurées. Avec l'essort du XML (eXtensible Markup Language) ces nouvelles bases de données ont trouvé leur support de préférence et sont devenues de plus en plus utilisées. Un problème se pose cependant, car s'il est facile de récupérer de l'information au sein de ces bases, il est très complexe de faire de l'interrogation de données (par exemple trouver tous les films dans lesquel apparaît un acteur donné).

Il existe des outils d'interrogation de données semi-structurées mais ces outils ne sont pas efficaces en terme de temps de réponse et de mémoire utilisés. Il serait donc intéressant de fournir un moyen simple et efficace d'interroger ces données. L'approche que nous proposons ne consiste pas à créer un nouvel outil d'interrogation de ces données mais de chercher à restructurer ces données. Une fois les données restructurées elles pourront être enregistrées dans une base de données relationnelle classique. Cette base pourra alors être interrogée efficacement grâce aux techniques développées pour les base de données relationnelles.

Pour arriver à restructurer ces bases de données il faut d'abord trouver un moyen d'affecter chaque donnée à une table dans une base de donnée relationnelle. Cette étape de la restructuration est le typage des données, mais rapidement le nombre de types différents au sein d'une base de données semi-structurée va devenir trop important (et c'est la raison même de leur existence). Il faut donc réduire ce nombre à un nombre plus acceptable, c'est-à-dire faire un choix parmi l'ensemble des types existants pour n'en retenir qu'un sous-ensemble.

Nous verrons dans la section 2.2 comment ce problème se ramène à une instance du problème du p-médian.

Un des enjeux de ce problème de restructuration est la taille des données. Les bases de données possèdent des milliers d'enregistrements. Même si tous ces enregistrements ne donnent pas naissance à un sommet lors de la transformation du problème, la taille du p-médian résultant à résoudre peut rapidement devenir très grande.

2.1 Base de données semi-structurée

Une recherche de données sur le web va produire en général des données irrégulières. Par exemple les différentes informations présentes sur les pages des membres d'un même groupe vont contenir des informations communes (nom, prénom, adresse mail, ...) mais certaines pages vont contenir des informations supplémentaires que les autres n'auront pas (surnom, photo, ...).

Ces données semi-structurées sont nées avec Internet et la nécessité de stocker des informations diverses et variées le plus efficacement possible. Mais ces données sont aussi une réponse aux limites des bases de données relationnelles. Considérons une base de donnée dont le schéma est défini par trois tables :

- une table *journal* avec un identificateur et le titre du journal,
- une table *article* avec un identificateur, un titre, un auteur et le journal dans lequel l'article a été publié,
- une table *auteur* avec un identificateur et le nom de l'auteur.

Ces tables sont remplies comme le montre la Figure 1.

Table auteur

Id	Nom
1	E. Prié
2	O. Letréguily
3	V. Vaisman
5	P. Gonneau

Table article

Id	Titre	Auteur	Journal
6	Prise de tête	2	4
7	Mat !	5	4

Table journal

Id	Titre
4	Echecs et Mats !

Figure 1: Exemple de remplissage d'une base de données relationnelle décrivant un journal. La base comporte 3 tables et quelques enregistrements.

La structure de ces tables étant fixée, il est impossible d'avoir un auteur ayant un nom et un prénom, ou d'avoir un article ayant deux auteurs. De même comme il faut fixer la structure des tables, il est impossible de définir une table *journal* ayant un nombre d'articles non borné. Ce sont ces limitations et la nécessité de stocker un nombre de données important qui ont permis l'émergence des bases de données semi-structurées. En effet, il est facile de stocker dans un fichier des relations entre des objets, ainsi un journal va pouvoir contenir des articles, et chaque article peut avoir un ou plusieurs auteurs. Chaque auteur pourra alors être défini d'une manière particulière. Ce qui peut donner des fichiers simples mais contenant de nombreuses informations. La Figure 2 présente un fichier XML contenant un document semi-structuré.

Cette notion de document semi-structuré peut être encore affinée. Une donnée semi-structurée se représente sous la forme d'un graphe enraciné dont les arcs sont orientés et labellisés [24]. Cette représentation convient très bien pour modéliser le web par exemple. Chaque page d'un site serait un sommet du graphe et chaque lien existant serait un arc entre deux sommets. Mais cette représentation convient tout aussi bien pour afficher les relations contenues dans une base de données. La Figure 3 présente un tel graphe pour l'exemple du document de la Figure 2.

Il existe dans ces bases de données deux types d'objets différents : les objets complexes et les objets atomiques.

- **Objet Atomique** : ce sont les objets de la base porteurs des valeurs. Ces objets sont les sommets pendants du graphe associé, c'est-à-dire les objets n'ayant aucun arc sortant.

- **Objet Complexe** : ce sont les objets de la base porteurs des relations.

Nous remarquons facilement un premier intérêt à l'utilisation des bases de données semi-structurées avec ce petit exemple. En effet, il est clair que la place requise pour stocker ces informations sous la forme d'une base de donnée relationnelle est beaucoup plus importante que celle requise pour un stockage sous la forme de données semi-structurées. Mais c'est surtout la simplicité du stockage des informations qui est intéressante. Pour dire qu'un article fait partie d'un journal, il suffit d'ouvrir une balise *article* au sein de la balise *journal*

```
<document id = 0>
 <journal id = 4>
  <titre id = 12>Echecs et Mats !</titre>
  <article id = 6>
   <titre id = 11>Prise de tête</titre>
   <écritPar ref = 2 />
   <écritPar ref = 3 />
  </article>
  <article id = 7>
   <titre id = 13>Mat !</titre>
   <écritPar ref = 5 />
  </article>
 </journal>
 <auteur id = 2>
  <nom id = 9>O. Letréguily</nom>
  <co-auteur ref = 3 />
 </auteur>
 <auteur id = 3>
  <nom id = 10>V. Vaisman</nom>
  <co-auteur ref = 2 />
 </auteur>
 <auteur id = 1>
  <nom id = 8>E. Prié</nom>
 </auteur>
 <auteur id = 5>
  <prénom id = 14>Pascal</prénom>
  <nom id = 15>Gonneau</nom>
 </auteur>
</document>
```

Figure 2: Exemple de document semi-structuré. Le document est présenté sous la forme d'un fichier XML.

correspondante. Dans une base de donnée relationnelle il faut stocker la relation l'article i appartient au journal j. Et il est très facile de faire des relations complexe dans un document semi-structuré. Si deux auteurs (ou plus) ont écrit un article il suffit de mettre deux balises *auteur* (ou plus) au sein de la balise *article* correspondante. Pour gérer des articles à deux auteurs il faudrait ajouter à la description de la base de données relationnelle de nouvelles tables. Pour contenir l'ensemble des informations du fichier de la Figure 2 il faudrait :

- ajouter une table pour la gestion des articles à deux auteurs,
- ajouter une table pour la gestion des co-auteurs,
- ajouter une table pour les auteurs avec un nom et un prénom (ou alors modifier les tables existantes et remplacer les informations manquantes par un champ vide).

Il faut alors créer beaucoup de tables, gérer les dépendances entre ces tables, et faire en sorte que les auteurs de la table *auteur sans prénom* et ceux de la

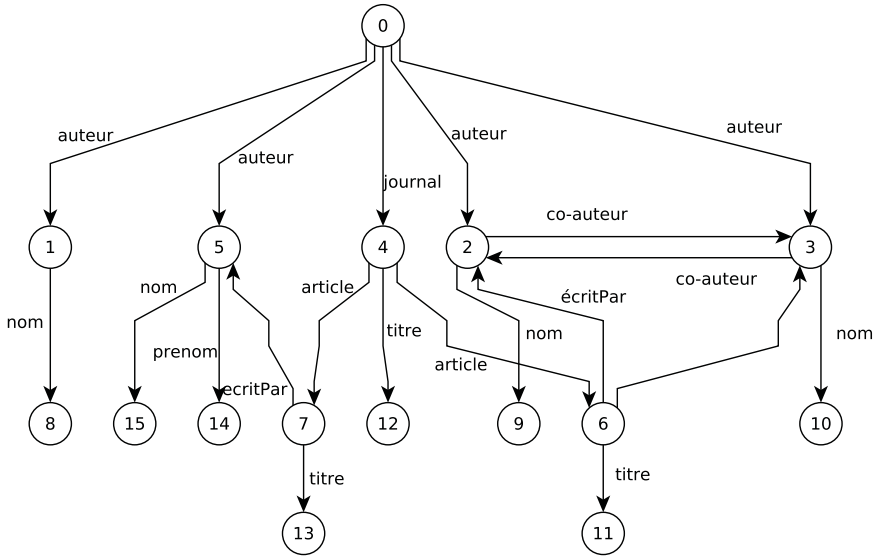


Figure 3: Représentation sous forme d'un graphe enraciné dirigé et labellisé du document semi-structuré de la Figure 2.

table *auteur avec prénom* puissent être utilisés indifférement. Ou alors il faut accepter une sur-représentation de l'information (des champs laissés vides). Si l'on a des données très irrégulières, la plupart des tables que l'on devra créer pour coller aux données ne contiendront que peu d'enregistrements, voir un seul.

Certains n'hésitent pas à dire qu'avoir des données irrégulières est la nouvelle norme en ce qui concerne les données réparties sur le web. Le problème de restructuration de ces données arrive naturellement lorsqu'une personne essaye de regrouper des informations provenant de sources hétérogènes. Ces données qui se caractérisent par une structure absente, irrégulière, partielle ou implicite se nomment données semi-structurées [1, 23].

Les système de gestion de bases de données (SGBD) sont en général pourvu d'une jolie interface graphique et de méthodes d'accès efficaces aux données. Ces deux services sont basés sur la connaissance à priori de la structure des données et du type des données. La formulation d'une requête est grandement facilitée par les interfaces homme-machine qui permettent aux programmeurs ou aux utilisateurs d'interrroger la base pour en extraire des informations sur la structure des données, et ainsi y avoir accès plus facilement. De plus cette connaissance permet d'améliorer les performances des requêtes grâce au principe des indexes [102].

Il est clair que l'utilisation de données semi-structurées ne permet pas de faire toutes ces améliorations, et même la manipulation de ces données n'est pas simple. Cependant, il existe en leur sein une structure qui doit être pris en compte même si elle n'est que partielle.

2.1.1 XML et bases de données semi-structurées

Avec son émergence et sa prise d'importance le langage XML (eXtensible Markup Language) est devenu, grâce à ses caractéristiques, le support favori des bases de données semi-structurées. Tout comme le HTML (HyperText Markup Language) dont il hérite, le XML est un langage balisé qui présente les informations qu'il contient à la fois dans le nom des différentes balises, dans leurs attributs mais aussi dans leurs contenus. A la différence de HTML, qui fournit un nombre de balises fini et limité, XML est un méta langage qui permet la création d'un nombre illimité de balises différentes.

```
<journal>
  <article>
    <titre>Le Mat</titre>
  </article>
  <article>
    <titre id = 1>Les Fous</titre>
  </article>
  <article>
    <titre ref = 1/>
    <auteur>Kasparov</auteur>
  </article>
</journal>
```

Figure 4: Exemple de fichier XML représentant une base de données

Un exemple d'une base de donnée stockée sous la forme d'un fichier XML est décrit sur la Figure 4. Cette base contient des informations sur la composition d'un journal. Un journal est composé de deux articles et de deux auteurs selon ces données. Une base de donnée peut donc être stockée dans un fichier XML, un fichier dans lequel la structure même partielle des données existe mais n'est pas connue à priori par le système.

Nous pouvons représenter ces données sous une autre forme plus visuelle, celle d'un graphe dirigé labellisé comme pour les données du document de la Figure 2. Les sommets du graphe représentent les objets de la base et les labels sur les arêtes portent les informations sémantiques à propos des relations entre les différents objets. La Figure 5 représente le graphe associé au document décrit par la Figure 4. La relation entre les objets 1 et 2 signifie simplement que 2 est un article de 1, de même pour les autres relations. L'article 2 possède un titre (l'objet 5). L'information concrète, le titre de l'article 2, est stocké directement au sein de l'objet 5, si l'on regarde à l'intérieur du fichier XML on pourra retrouver ce titre : "Le Mat". Ces objets qui sont porteurs des informations et qui ne possède que des relations d'appartenance sont représentés par des sommets pendants (en gras sur la Figure 5) dans le graphe. Ces sommets représentent les objets atomiques de la base et possèdent des valeurs attachées (pour le sommet 6 cette valeur est "Les Fous"). Ces informations sont celles qui seront contenues dans une base de donnée décrivant ce document. Cependant nous ne nous intéresserons pas à ces valeurs. Nous cherchons à comprendre la structure sous-jacente des données, et pour y arriver nous n'avons besoin que des relations entre les données. Seule la structure contenue dans le fichier nous

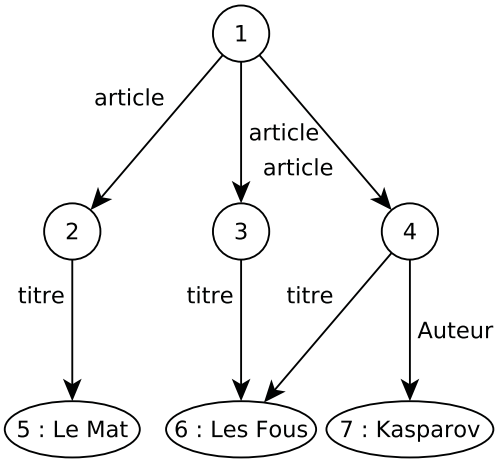


Figure 5: Représentation sous la forme d'un graphe enraciné, dirigé et labellisé des données semi-structurées contenues dans le document décrit par la Figure 4.

intéresse car elle va nous permettre de déterminer un type associé à chaque objet. Comme nous le verrons plus tard, tous les objets atomiques peuvent être associés à un type unique.

2.1.2 Typage des données en base de données

Dans leur article [85] Nestorov, Abiteboul et Motwani fournissent une méthode pour extraire un schéma d'une base de donnée semi-structurée. Leur méthode se décompose en trois étapes. La première étape consiste à assigner à chaque objet son type parfait, c'est-à-dire le type qui correspond exactement à l'objet. Dans la seconde étape ils utilisent un problème d'optimisation pour réduire ce nombre de types à un nombre plus restreint tout en cherchant à minimiser l'impact de cette réduction sur les données. Enfin dans une troisième étape ils effectuent les opérations de transformation nécessaires pour obtenir une base de donnée relationnelle.

Typage des données

Dans [85] les auteurs représentent une base de données semi-structurée par un graphe dirigé labellisé comme nous pouvons le voir sur la Figure 5. Les sommets du graphe représentent les objets et les labels contiennent la sémantique des relations entre les objets. Les sommets pendants du graphe (les sommets sans arcs sortants) représentent les objets atomiques, les objets porteurs de valeurs. Un tel graphe est stocké par ces auteurs sous la forme d'une base de donnée relationnelle à deux tables :

- **link(FromObj, ToObj, Label) :** la relation *link* contient les informations sur les arêtes. $link(o_1,o_2,l)$ correspond à une arête entre o_1 et o_2 portant le label l.

- **atomic(Obj, Value) :** cette relation contient les informations sur les valeurs des objets atomiques.

Pour définir le type d'un objet ou d'une donnée, ils définissent un programme datalog sur ces deux relations *link* et *atomic*. Le type d'un objet est une règle de la forme :

$$c(X) = A_1 \& ... \& A_k$$

pour un certain k, où les A_i, appelés les liens typés, sont définis par l'une des formes suivantes :

- $link(Y, X, l) \& c'(Y)$
- $link(X, Y, l) \& c'(Y)$
- $link(X, Y, l) \& atomic(Y, Z)$

où l est une constante (un label), X la variable de la règle et Y et Z des variables n'apparaissant pas dans un autre lien typé de la règle. Pour faciliter la lecture ils introduisent les notations suivantes :

- $link(Y, X, c) \& type_j(Y)$ sera noté $\overleftarrow{c}^j$
- $link(X, Y, c) \& type_j(Y)$ sera noté $\overrightarrow{c}^j$
- $link(Y, X, c) \& atomic(Y, Z)$ sera noté $\overleftarrow{c}^0$

Les liens vers des objets atomiques sont notés comme des liens vers le $type_0$ pour simplifier.

Exemple. *Reprenons la base de donnée décrite par la Figure 4. Le graphe qui lui est associé est présenté sur la Figure 5. Pour chacun des objets* 1 *à* 4 *nous pouvons définir le type qui lui est associé. Nous obtenons la liste des types suivante :*

$$type_1 = \overrightarrow{article}^2, \overrightarrow{article}^3, \overrightarrow{article}^4 \quad type_2 = \overleftarrow{article}^1, \overrightarrow{titre}^0$$
$$type_3 = \overleftarrow{article}^1, \overrightarrow{titre}^0 \quad type_4 = \overleftarrow{article}^1, \overrightarrow{titre}^0, \overrightarrow{auteur}^0$$

Typage parfait minimal

Pour construire le typage parfait d'une base donnée les auteurs donnent un algorithme sous la forme d'un programme de typage. Ce programme sera alors exécuté sur la base de données modélisant le graphe associé aux données semi-structurées. Etant donnée une base de donnée D, le programme de *typage parfait minimal* P_D est construit de la manière suivante :

1. Construire un programme Q_D : soit $o_1, ..., o_N$ les objets complexes de la base. A chaque objet complexe o_k nous affectons un type $type_k$. La règle pour ce type contiendra $\overleftarrow{l}^i$ si et seulement s'il existe une arête labelisée l de o_i à o_k et $\overrightarrow{l}^i$ si et seulement s'il existe une arête labelisée l de o_k à o_i. Enfin cette règle contiendra $\overrightarrow{l}^0$ si et seulement s'il existe une arête labelisée l de o_i vers un objet atomic.

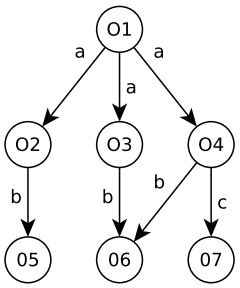


Figure 6: Une base de données simple

2. Calculer le plus grand point fixe M de Q_D pour D. On défini alors la relation d'équivalence "$\equiv$" sur les types $\{type_k | k = 1, \ldots, N\}$ par $type_i \equiv type_j$ si $M(type_i) = M(type_j)$. Les types de P_D seront alors les classes d'équivalence de la relation "$\equiv$", soit $\tau_1, ..., \tau_n$.

3. Le nouveau programme P_D est obtenu en choisissant pour chaque τ_i un type $type_k$ dans τ_i et en replaçant dans la règle du $type_k$ chaque occurence du $type_j$ par sa classe d'équivalence $[type_j]$ selon la relation "$\equiv$".

Exemple. *On considère la base de données décrite par la Figure 6. Le programme Q_D construit en* (1) *est le suivant :*

$$\begin{array}{ll} type_1 = \overrightarrow{a}^2, \overrightarrow{a}^3, \overrightarrow{a}^4 & type_2 = \overleftarrow{a}^1, \overrightarrow{b}^0 \\ type_3 = \overleftarrow{a}^1, \overrightarrow{b}^0 & type_4 = \overleftarrow{a}^1, \overrightarrow{b}^0, \overrightarrow{c}^0 \end{array}$$

Le plus grand point fixe M pour Q_D obtenu en (2) *est :*

- $M(type_1) = \{o_1\}$,
- $M(type_2) = M(type_3) = \{o_2, o_3, o_4\}$,
- $M(type_4) = \{o_4\}$.

Nous notons alors les types $\tau_1 = [type_1]$, $\tau_2 = [type_2] = [type_3]$ et $\tau_3 = [type_4]$. Le programme P_D finale est alors :

$$\begin{array}{ll} \tau_1 = \overrightarrow{a}^2, \overrightarrow{a}^2, \overrightarrow{a}^3 & \tau_2 = \overleftarrow{a}^1, \overrightarrow{b}^0 \\ \tau_3 = \overleftarrow{a}^1, \overrightarrow{b}^0, \overrightarrow{c}^0 & \end{array}$$

Ainsi nous avons défini trois types pour cette base de données, les type τ_1, τ_2, et τ_3. Le type τ_1 est le type des objets o_1, le type τ_2 est le type des objets o_2 et o_3 et le type τ_3 est le type des objets o_4.

Comment trouver le plus grand point fixe d'un programme P ?

Pour trouver le plus grand point fixe d'un programme P il existe une technique simple. Nous appliquons le programme P à la base D. Puis nous continuons d'appliquer P au dernier résultat obtenu jusqu'à ce que le résultat ne change plus. Nous avons alors le plus grand point fixe de P.

Exemple. *Reprenons l'exemple précédent. On a le programme datalog suivant :*

$$type_1 = \overrightarrow{a}^2, \overrightarrow{a}^3, \overrightarrow{a}^4 \quad type_2 = \overleftarrow{a}^1, \overrightarrow{b}^0$$
$$type_3 = \overleftarrow{a}^1, \overrightarrow{b}^0 \quad type_4 = \overleftarrow{a}^1, \overrightarrow{b}^0, \overrightarrow{c}^0$$

Nous appliquons ce programme à la base D. Quels sont les éléments de D qui contiennent la règle $type_1$? La réponse est o_1 uniquement. On note alors $P(type_1) = \{o_1\}$. De même on obtient :

- $P(type_1) = \{o_1\}$,
- $P(type_2) = \{o_2, o_3, o_4\}$,
- $P(type_3) = \{o_2, o_3, o_4\}$,
- $P(type_4) = \{o_4\}$.

Nous remarquons déjà que $P(type_2) = P(type_3)$, on peut donc fusionner ces deux type. Nous créons alors les types associés à ces ensembles, qui sont $type_1$, $type_2$ et $type_4$. On ré-applique P à ces types, pour trouver exactement le même résultat. On arrête donc, nous avons trouvé le plus grand point fixe de P sur D.

Cette technique peut prendre jusqu'à $O(N^2)$ opérations où N est le nombre d'objets si elle est implémentée de manière directe. Certaines améliorations peuvent être mises en place pour améliorer cette complexité [102].

Autre méthode

La technique mise au point par Nestorov et al. [85] nécessite la mise en place d'une base de donnée contenant l'ensemble des types de données de la base initiale. Cette technique suppose aussi une connaissance théorique en base de donnée assez importante, ce qui n'est pas notre cas. L'avantage de cette technique reste cependant qu'elle peut être aussi mise en pratique sur une base de donnée structurée pour un coût minimum. Nous avons utilisé une autre technique pour extraire d'une base de donnée semi-structurée son typage parfait minimum.

Nous supposons ici que la base de donnée est stockée sous la forme d'un fichier XML. L'extraction se fait alors simplement en parcourant le fichier XML à l'aide d'un parseur (voir Section 2.3.1). A chaque nouvel élément trouvé lors du parcours nous créons un objet et notons ses relations avec le reste des objets déjà créés ou des objets qui le seront par la suite. Nous obtenons ainsi un typage parfait pour cette base de données. Nous appliquons ensuite un algorithme de réduction à ce typage parfait. Pour chaque objet existant nous comparons son type aux autres et s'ils sont identiques, nous fusionnons les deux objets. En parcourant ainsi l'ensemble des types existants, nous obtenons le typage parfait minimal de la base de données.

Cette technique n'est pas plus rapide que celle mise en place par Nestorov et al. mais présente l'avantage de n'avoir ni base de donnée à installer, ni programme de typage à écrire, comme ils le font. De plus nous pouvons facilement et pour un coût faible ajouter à ce programme la construction de la matrice des distances entre les types. Nous pouvons ainsi passer directement de la base de donnée à l'instance du p-médian qu'il faut résoudre pour effectuer la restructuration de ces données.

2.2 Restructuration de bases de données

2.2.1 Principe

Le problème de restructuration d'une base de données semi-structurée consiste, étant donné le typage parfait minimal, à trouver un ensemble de types à conserver et à affecter les types qui ne sont pas conservés à un des types choisis. Ce problème ressemble de prime abord à un problème de p-médian, qui consiste à choisir p sommets d'un graphe, les centres, et à affecter chaque sommet non choisi à un centre. Dans le problème du p-médian on cherche à minimiser la somme des coûts des affectations.

Pour répondre au problème de restructuration de bases de données Nestorov et al. utilisent le problème du k-clustering, un problème similaire au problème du p-médian. Ils associent chaque type présent dans le typage parfait à un sommet d'un graphe et sélectionnent uniquement k des n sommets de ce graphe. Cette sélection est faite en fonction des distances entre les sommets du graphe et du nombre d'objets ayant un type donné. Les k types sélectionnés sont tels que la distance entre ces types et les autres soit la plus faible possible. C'est un problème de minimisation.

Chaque type du typage parfait sera représenté par un sommet dans un graphe, et tous les arcs seront considérés, mais il faut une valuation sur les arcs pour choisir les meilleurs k types restant (meilleurs au sens où ils permettent de minimiser la somme des affectation des autres types). Il faut donc une notion de distance entre les types.

Dans l'exemple précédent nous avons dit que les types τ_2 et τ_3 étaient proches l'un de l'autre. Il est assez simple de voir que leur définitions se ressemblent et que donc la modification de l'un en l'autre ne devrait pas faire perdre beaucoup d'informations. Il faut cependant quantifier cette perte d'informations et définir une notion mathématique de distance entre deux types.

Considérons deux types τ_1 et τ_2. La distance la plus naturelle entre ces deux types est la distance de Manhattan entre les deux points représentant ces types sur l'hypercube défini par leur liens typés. De manière simple, si nous considérons que chaque type est un ensemble d'attributs, alors la distance de Manhattan entre deux types est la cardinalité de leur différence symétrique. On note cette distance $d(\tau_1, \tau_2)$.

Exemple. *Considérons les types suivants :*

$$\tau_1 = \overrightarrow{a}^0, \overrightarrow{b}^2 \qquad \tau_2 = \overrightarrow{a}^0, \overleftarrow{b}^1$$
$$\tau_3 = \overrightarrow{b}^2, \overrightarrow{b}^1, \overleftarrow{b}^3$$

Pour τ_1, τ_2 la différence symétrique consiste en $\{\overrightarrow{b}^2, \overleftarrow{b}^1\}$, donc $d(\tau_1, \tau_2) = 2$. Pour τ_1, τ_3 la différence symétrique consiste en $\{\overrightarrow{a}^0, \overrightarrow{b}^1, \overleftarrow{b}^3\}$, donc $d(\tau_1, \tau_3) = 3$.

Cette distance permet de prendre en compte les différences entre deux types, que ce soit par excès ou déficit. Les différents liens typés présents dans un type mènent à des informations. Si un type contient le lien typé $\overrightarrow{article}^2$ cela signifie que les objets associés contiennent un article de type 2. Donc, le calcul des distances prend en compte les considérations suivantes :

- On peut considérer que le fait de convertir un type t ayant un lien typé $\overrightarrow{label}^i$ en un type t' ne contenant pas ce lien typé va faire perdre cette information, et donc doit être compté dans ce sens.

- On peut aussi considérer que le fait de convertir un type t n'ayant pas un lien typé $\overrightarrow{label}^i$ en un type t' contenant ce lien typé va faire apparaitre une information inutile, et donc doit être compté dans ce sens.

Cette distance simple ne prend pas en compte le nombre d'objets liés à un type alors qu'il faudrait. La distance utilisée dans [85] est la fonction c suivante :

$$c(w_1, w_2) = n(w_1) * d(w_1, w_2)$$

où w_1 et w_2 sont des types et $n(w_1)$ représente le nombre d'objets de la base ayant le type w_1. Il faut noter que d est une métrique alors que c ne l'est pas.

La forme générale du problème de restructuration est cependant plus compliqué que cela. En effet chaque fois que la fusion entre deux types est décidée elle peut entraîner une réduction de la taille du problème. La prise en compte des ces modifications au cours de la résolution du problème n'est pas prise en compte par les auteurs. L'exemple suivant illustre un tel cas.

Exemple. *Considérons les tpyes suivants :*

$$\begin{array}{ll} \tau_1 = \overrightarrow{a}^0, \overrightarrow{b}^3 & \tau_2 = \overrightarrow{a}^0, \overrightarrow{b}^4 \\ \tau_3 = \overrightarrow{a}^0, \overleftarrow{b}^1 & \tau_4 = \overrightarrow{a}^0, \overleftarrow{b}^2 \end{array}$$

Initiallement les quatre types sont différents. Affectons le type τ_2 à τ_1. Les types restants sont donc :

$$\begin{array}{ll} \tau_1 = \overrightarrow{a}^0, \overrightarrow{b}^3 & \\ \tau_3 = \overrightarrow{a}^0, \overleftarrow{b}^1 & \tau_4 = \overrightarrow{a}^0, \overleftarrow{b}^1 \end{array}$$

Maintenant, les types τ_3 et τ_4 sont identiques et peuvent donc être fusionnés sans le moindre coût. Imaginons maintenant que nous ayons effectué la fusion entre les types τ_3 et τ_4 en premier. Nous nous serions retrouvé avec les types τ_1 et τ_2 identiques.

Cet exemple montre deux choses. D'abord lorsque l'on réalise la fusion entre deux types, cela va avoir un impact sur l'ensemble des autres types, et donc sur les distances entre les types. Le calcul du coût réel devient alors beaucoup plus complexe. Ensuite l'ordre des fusions peut avoir son importance. Ici ce n'est pas le cas, mais il existe des situations dans lesquelles l'ordre des affectations va avoir une influence sur la qualité du résultat.

La forme général du problème de restructuration d'une base de donnée semi-structurée est donc très compliquée. En plus de devoir trouver une affectation de l'ensemble des types vers un sous-ensemble de ces même types il faut trouver un ordre pour réaliser ces affectations. Trouver un ordre ainsi que le sous-ensemble minimisant l'excès et le déficit d'informations est intraitable. Nous nous Gconcentrerons donc sur la forme la version simplifiée du problème consistant à trouver le sous-ensemble minimisant l'excès et le déficit d'informations.

2.2.2 Réduction du problème en une instance du problème du p-médian

Nous considérons maintenant le problème simplifié de restructuration d'une base de donnée semi-structurée. Nous allons voir comment ce problème se réduit en une instance du problème du p-médian au travers d'un exemple.

Exemple. *Nous utiliserons dans cet exemple la base de données décrite sur la Figure 3. Cet exemple ne décrit pas comment passer d'un fichier XML au typage parfait minimal. Cette procédure est décrite à la section 2.3.1. La base de donnée comporte 8 objets complexes (les objets 0, 1, 2, 3, 4, 5, 6 et 7) et 8 objets atomiques (les objets 8, 9, 10, 11, 12, 13, 14 et 15). Il faut donc créer 8 types. Par soucis de lisibilité nous renumérotons les objets de 1 à 16, ainsi le type $type_0$ désignera le type des objets atomiques (les objets 9 à 16). Voici la liste des types :*

$type_1 = \overrightarrow{auteur}^2, \overrightarrow{auteur}^3, \overrightarrow{auteur}^4, \overrightarrow{journal}^5, \overrightarrow{auteur}^6$
$type_2 = \overleftarrow{auteur}^1, \overrightarrow{nom}^0$
$type_3 = \overleftarrow{auteur}^1, \overrightarrow{co-auteur}^4, \overleftarrow{co-auteur}^4, \overleftarrow{ecritPar}^7, \overrightarrow{nom}^0$
$type_4 = \overleftarrow{auteur}^1, \overrightarrow{co-auteur}^3, \overleftarrow{co-auteur}^3, \overleftarrow{ecritPar}^7, \overrightarrow{nom}^0$
$type_5 = \overleftarrow{journal}^1, \overrightarrow{article}^7, \overrightarrow{article}^8, \overrightarrow{titre}^0$
$type_6 = \overleftarrow{auteur}^1, \overleftarrow{ecritPar}^8, \overrightarrow{nom}^0, \overrightarrow{prenom}^0$
$type_7 = \overleftarrow{article}^5, \overrightarrow{ecritPar}^3, \overrightarrow{ecritPar}^4, \overrightarrow{titre}^0$
$type_8 = \overleftarrow{article}^5, \overrightarrow{ecritPar}^6, \overrightarrow{titre}^0$

Les types $type_3$ et $type_4$ peuvent être regroupés en un seul type. Ce qui nous donne le typage parfait minimal suivant :

$\tau_1 = \overrightarrow{auteur}^2, \overrightarrow{auteur}^3, \overrightarrow{auteur}^3, \overrightarrow{journal}^4, \overrightarrow{auteur}^5$
$\tau_2 = \overleftarrow{auteur}^1, \overrightarrow{nom}^0$
$\tau_3 = \overleftarrow{auteur}^1, \overrightarrow{co-auteur}^3, \overleftarrow{co-auteur}^3, \overleftarrow{ecritPar}^6, \overrightarrow{nom}^0$
$\tau_4 = \overleftarrow{journal}^1, \overrightarrow{article}^6, \overrightarrow{article}^7, \overrightarrow{titre}^0$
$\tau_5 = \overleftarrow{auteur}^1, \overleftarrow{ecritPar}^7, \overrightarrow{nom}^0, \overrightarrow{prenom}^0$
$\tau_6 = \overleftarrow{article}^4, \overrightarrow{ecritPar}^3, \overrightarrow{ecritPar}^3, \overrightarrow{titre}^0$
$\tau_7 = \overleftarrow{article}^4, \overrightarrow{ecritPar}^5, \overrightarrow{titre}^0$

Nous pouvons maintenant calculer toutes les distances entre les types. Voici donc la matrice des distances correspondante :

$$\begin{bmatrix} 0 & 6 & 9 & 8 & 8 & 8 & 7 \\ 6 & 0 & 3 & 6 & 2 & 6 & 5 \\ 9 & 3 & 0 & 9 & 3 & 9 & 8 \\ 8 & 6 & 9 & 0 & 8 & 6 & 5 \\ 8 & 2 & 3 & 8 & 0 & 8 & 7 \\ 8 & 6 & 9 & 6 & 8 & 0 & 1 \\ 7 & 5 & 8 & 5 & 7 & 1 & 0 \end{bmatrix}$$

Ce qui nous permet de calculer la matrice des coûts pour ce problème :

$$C = \begin{bmatrix} 0 & 6 & 9 & 8 & 8 & 8 & 7 \\ 6 & 0 & 3 & 6 & 2 & 6 & 5 \\ 18 & 6 & 0 & 18 & 6 & 18 & 16 \\ 8 & 6 & 9 & 0 & 8 & 6 & 5 \\ 8 & 2 & 3 & 8 & 0 & 8 & 7 \\ 8 & 6 & 9 & 6 & 8 & 0 & 1 \\ 7 & 5 & 8 & 5 & 7 & 1 & 0 \end{bmatrix}$$

Cette matrice des coûts ne suffit pas à définir une instance du p-médian. Il faut en plus fournir le nombre de types que l'on souhaite conserver. Si nous voulions conserver 4 types, l'instance du p-médian associée serait donc la matrice des coûts C et $p = 4$.

Au travers de cet exemple nous avons vu comment réduire le problème simplifié de restructuration d'une base de donnée semi-structurée en un problème de p-médian. La procédure est applicable à toutes les bases de données que nous voudrions restructurer. Il faut trouver le typage parfait minimal de la base, calculer la matrice des distances entres tous les types le composant, et enfin calculer la matrices des coûts. Cette matrice et une valeur de p décrivent une instance du p-médian.

2.2.3 Variante du problème

Nous avons observé précédemment que le problème de restructuration pouvait être plus ou moins complexe suivant que l'on considère l'ordre de fusion des types entre eux ou non. Mais nous pouvons aussi considérer un autre problème dans lequel nous allons ajouter des types qui n'existent pas au typage parfait minimal. Si ces types sont bien choisis ils peuvent réduire le coût des affectations et donc la perte de données.

Exemple. *Considérons la base de données qui donne le graphe des distances de la Figure 7. La base comporte 4 types dans son typage parfait, les types τ_2, τ_3, τ_4 ont chacun 3 objets attachés alors que le type τ_1 n'en a qu'un seul. De ces types nous souhaitons n'en conserver que deux.*

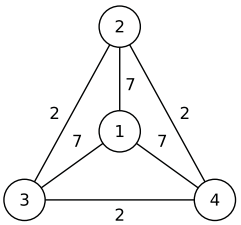


Figure 7: Graphe des distances d'une base de données.

Les types des objets 1 *à* 4 *de la Figure 7 sont les suivants :*

$$\tau_1 = (1111111000)$$
$$\tau_2 = (0000011110)$$
$$\tau_3 = (0000011101)$$
$$\tau_4 = (0000011011)$$

Les types sont présentés sous la forme de vecteurs $\{0, 1\}$*. Un "*1*" représente la présence du lien typé considéré et un "*0*" son absence.*

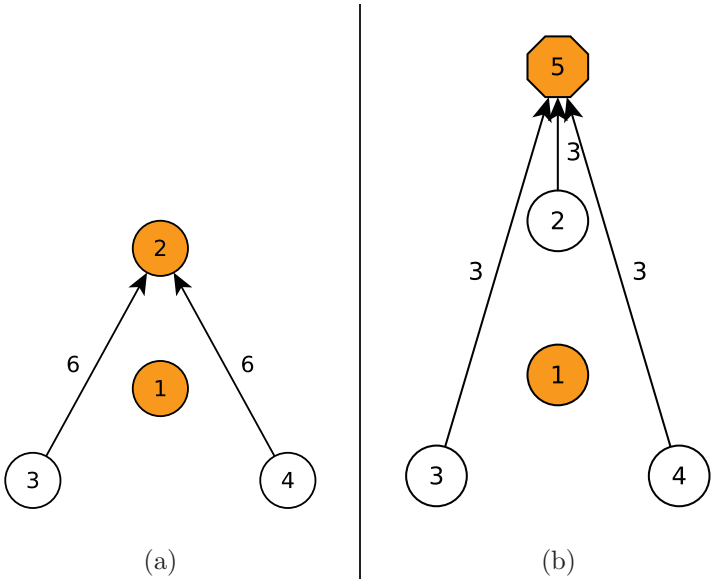


Figure 8: Solutions optimales à deux problèmes de p-médian. (a): Solution optimale au problème initial de valeur 12. (b): Solution optimale au problème modifié de valeur 9

Sur la Figure 8(a) nous pouvons voir une solution optimale au problème du p-médian associé. La solution consiste à conserver les types τ_1 *et* τ_2 *puis à affecter les objets représentés par les types* τ_3 *et* τ_4 *à* τ_2*. Cette solution optimale n'est pas unique, en effet les types* τ_2*,* τ_3 *et* τ_4 *peuvent être intervertis sans problèmes. Le coût de ces solutions est de 12. La partie (b) de la Figure 8 présente une autre solution. Nous considérons toujours les mêmes types pour* τ_1 *à* τ_4 *mais nous ajoutons un nouveau type* τ_5 *(et donc un nouveau sommet). Ce type est défini par le vecteur suivant :*

$$\tau_5 = (0000011111),$$

nous ajoutons donc un type nouveau qui n'existait pas. Nous cherchons toujours à ne conserver que 2 types parmi nos 5 types existants. La solution optimale à ce nouveau problème consiste à sélectionner les types τ_1 *et* τ_5 *et à affecter les types* τ_2*,* τ_3 *et* τ_4 *au type* τ_5*. Le coût de cette solution est de 9.*

Nous remarquons donc que le fait d'ajouter un nouveau type au problème permet de réduire le coût de la solution, et donc de réduire la perte d'information.

Ce problème est un nouveau problème. Nouveau dans le sens où personne n'a encore essayer de le résoudre. Pour résoudre ce problème nous proposons deux heuristiques développée dans la Section 4.2.

2.3 Création de données pour le p-médian

Pour obtenir des données pour le p-médian, il existe plusieurs manières de procéder. La plus évidente est de convertir un problème de restructuration de base de donnée semi-structurée. Mais nous avons aussi considéré d'autres techniques de génération aléatoire de données.

2.3.1 Création de donnée à partir d'une base de données semi-structurée

La première source de données est bien évidement la transformation d'une base de données semi-structurée en une instance du p-médian. Nous avons donc réalisé un programme qui lit en entrée un fichier XML contenant une telle base de données et fournit en sortie l'instance du p-médian associée.

La réalisation de cet outil repose sur la manière de parcourir un fichier XML. Un fichier XML, même s'il est totalement libre dans le nommage des balises et des attributs de celles-ci, respecte une syntaxe particulière. Il est donc tout à fait possible de créer un analyseur syntaxique qui fonctionne pour tous les fichiers XML correctement formatés. C'est ce qui a été fait de nombreuses fois et qui a donné naissance à plusieurs *parseurs* (francisation du terme anglais *parser*) de fichiers XML.

Ces parseurs fonctionnent selon deux modes différents. Les parseurs DOM (*Document Object Model*) définissent la structure d'un document sous la forme d'une hiérarchie d'objets. En pratique, ils parcourent le document en indexant chaque balises et construisent en mémoire un arbre contenant la totalité du document. Les parseurs SAX (*Simple API for XML*) utilisent un modèle évènementiel. Ils parcourent le document et lancent des évènements à chaque élément de la syntaxe qu'ils rencontrent (début de balise, fin de balise, etc.).

Les parseurs SAX, qui ont une utilisation de mémoire réduite et dont le côté évènementiel permet de ne traiter que la structure du document et pas son contenu, ont été retenus pour effectuer notre implémentation. Notre choix s'est porté sur la bibliothèque *Xerces-C* créée par *The Apache Software Foundation*. C'est un parseur écrit en C++ qui s'utilise très simplement par simple héritage de la classe **SAXParser** et quelques surcharges de méthodes appelées par le gestionnaire d'évènements (voir Tableau 1).

Ces quelques méthodes devront donc être surchargées pour réaliser notre outils de conversion. Les paramètres de ces méthodes permettent d'obtenir des informations sur les éléments rencontrés (nom de la balise, liste des attributs s'ils existent, etc.).

Le principe est donc très simple : une balise s'ouvre, il faut créer un nouvel objet. Tant que cette balise n'est pas fermée il est possible d'ajouter des relations à cet objet, décrite par l'ouverture d'autres balises. Une fois une balise fermée, le type de l'objet peut être déterminé. Une fois tous les types déterminés il suffit de fusionner les objets ayant le même type pour obtenir le typage parfait de la base de données.

virtual void **startDocument**() Cette méthode est appelée une fois au début de l'inspection du document.
virtual void **startElement**() Cette méthode est appelée à chaque fois qu'une balise ouvrante est rencontrée.
virtual void **endElement**() Cette méthode est appelée à chaque fois qu'une balise fermante est rencontrée.
virtual void **endDocument**() Cette méthode est appelée une fois à la fin de l'inspection du document.

Tableau 1: Liste des méthodes principales du parseur *Xerces-C*

```
<journal>
  <titre>Les échecs</titre>
  <article>
    <titre>Le Mat</titre>
    <auteur>
      <nom>G. Kasparov</nom>
    </auteur>
  </article>
  <article>
    <titre>Jouer ses cavaliers</titre>
    <auteur>
      <nom>V. Topalov</nom>
    </auteur>
  </article>
</journal>
```

Figure 9: Exemple de fichier XML

Exemple. *Le fichier présenté sur la Figure 9 est un fichier XML contenant une base de donnée. La base a trait à un journal, dont le titre est "Les échecs", contenant deux articles. Chaque article est écrit par un auteur, et chaque auteur possède un nom.*

Le déroulement de l'inspection du fichier est le suivant : La balise journal est ouverte, un type associé à l'objet est créé. La balise titre est ouverte, un type associé à l'objet est créé, un lien entre journal et titre est fait, de même qu'un lien entre titre et journal. La balise titre est fermée, aucune balise n'a été ouverte entre temps c'est donc un objet atomique, le type est détruit et remplacé par le type 0*. La balise article est ouverte, un type associé à l'objet est créé, des liens sont créés. Ainsi de suite. Les types obtenus seront :*

$$\begin{aligned}
\tau_1 &= \overrightarrow{titre}^0, \overrightarrow{article}^2, \overrightarrow{article}^3 \\
\tau_2 &= \overleftarrow{journal}^1, \overrightarrow{titre}^0, \overrightarrow{auteur}^4 \\
\tau_3 &= \overleftarrow{journal}^1, \overrightarrow{titre}^0, \overrightarrow{auteur}^5 \\
\tau_4 &= \overleftarrow{auteur}^2, \overrightarrow{nom}^0 \\
\tau_5 &= \overleftarrow{auteur}^3, \overrightarrow{nom}^0
\end{aligned}$$

Un problème majeur posé lors de la lecture d'un fichier XML est la résolution des références. En effet, les fichiers XML peuvent contenir des références vers

d'autres éléments présents dans le fichier. Ces références sont imprévisibles, il est possible que l'élément référencé soit placé avant dans le fichier, mais aussi après. La Figure 10 montre de tels exemples. Cette liberté offerte par le XML devient lourde lorsqu'il faut construire le type d'un élément.

```
<journal>
  <titre>Les échecs</titre>
  <auteur id=1>
   <nom>G. Kasparov</nom>
  </auteur>   <article>
    <titre>Le Mat</titre>
    <écritPar ref=1 />
  </article>
  <article>
    <titre>Jouer ses cavaliers</titre>
    <écritPar ref=2 />
  </article>
  <auteur id=2>
   <nom>V. Topalov</nom>
  </auteur>
</journal>
```

Figure 10: Exemple de fichier XML avec des références

Un premier problème facile à résoudre consiste à réaliser l'association entre les numéros des références dans le fichier XML en entrée et un identifiant du type correspondant dans notre structure de données. Un second problème se présente lorsqu'il faut résoudre des références qui n'ont pas encore été rencontrées. Dans l'exemple du fichier décrit sur la Figure 10 la résolution de la première référence peut être faite simplement. Lors de la création du type associé à l'auteur ayant l'identifiant 1, une entrée dans un tableau spécifiant que cet identifiant 1 se rapporte au type τ_i est crée. Il suffit alors de lire ce tableau pour connaitre le type associé au label *écritPar* lors de sa rencontre. La résolution de la seconde référence pose plus de problème. Nous ne pouvons pas simplement lire dans un tableau ce qui nous intéresse puisque l'élément référencé n'a pas encore été rencontré, et le type associé n'a donc pas encore été créé. Il faut par conséquent attendre d'avoir complètement lu le fichier XML pour être sûr d'avoir dans notre tableau la liste complète des références avec un identifiant de type associé.

Il existe dans la réalité de très grandes bases de données semi-structurées. Il suffit pour s'en persuader de considérer la base de donnée des films de cinéma de ces dernière cinquantes années qui serait construite à partir des sites internet de références. Chaque site stockant les informations suivant une logique particulière, la base résultante serait fortement semi-structurée. Mais de telles bases de données ne sont pas facile à construire et ce n'est pas le but de cette thèse. C'est pour cela que nous avons décidé de générer aléatoirement nos propres instances pour le p-médian. Nous avons essayé de faire en sorte que nos instances aient des structures proches de ce que nous aurions pu trouver dans des instances de base de données réelles.

2.3.2 Utilisation de problèmes existants

Comme nous ne pouvions utiliser de bases de données réelles pour tester notre méthode de résolution nous avons dû chercher des instances pour le problème du p-médian.

Il existe sur Internet plusieurs banque de données d'instances pour des problèmes d'optimisation. Les plus connues étant la **TSP Library** [91] et l'**OR Library** [19].

L'ORLIB a été introduite par Beasley [19] et regroupe des instances de problèmes de recherche opérationnelle. Parmi les différents problèmes présent se trouve le problème du p-médian. Cette bibliothèque fournit 40 instances du problème du p-médian. Chaque instance est un graphe avec une valeur de p. Chaque sommet de ces graphes est à la fois un client et un centre potentiel. Le coût de service d'un client par un centre est la longueur du plus court chemin de ce client à ce centre. Leur principal défaut est qu'elles sont de petites tailles (le nombre de sommets varie de 100 à 900) alors que nous nous intéressons à des tailles beaucoup plus importantes. Nous ne nous sommes donc pas servis de cette bibliothèque d'instances.

La TSPLIB [91] présente des instances du voyageur de commerce (symétrique et asymétrique) ainsi que des instances pour le problème du cycle hamiltonnien. Les instances du voyageur de commerce peuvent aisément être utilisées comme instances du p-médian. En effet chaque instance présente un ensemble de points sur un plan. Chaque point est considéré comme un centre potentiel et un client. Le coût de service d'un client par un centre est simplement la distance euclidienne entre les deux points considérés. Ces distances sont calculées de manière exacte puis tronquées à l'entier inférieur pour éviter des erreurs de calculs à cette étape. Ces instances ont été utilisé dans la litérature par Hasen et al. [58, 59] en premier. Les instances de cette bibliothèque sont beaucoup plus nombreuses et beaucoup plus grandes. La taille des instances varie de 16 sommets à plus de 13000. Comme nous souhaitions avoir des points de comparaison avec l'heuristique d'Avella et al. [5] nous avons considéré le même ensemble d'instance à tester : rl1304, fl1400, u1432, vm1748, d2103, pcb3038, fl3795, rl5934. Chaque instance peut être résolue pour des valeurs de p différentes.

2.3.3 Génération aléatoire de données

L'idée, ici, est de générer des données qui ressemblent aux problèmes venus des bases de données. Nous allons utiliser des définitions de types pour générer un ensemble d'objets ayant un type. On considère qu'un type est représenté par un vecteur de 0 et de 1 d'une certaine taille k. Une **définition de type**, notée tp_i est simplement un vecteur de $[0,1]^k$ qui va permettre de générer aléatoirement des types. Chaque type est généré de la façon suivante : pour chaque élément k de tp_i on tire un nombre aléatoire. Si ce nombre est plus grand que tp_i^k alors la $k$ième composante du type généré sera 0, sinon elle sera 1.

Nous nous sommes servis de cette technique de génération de données pour générer des instances pouvant être utilisées pour tester nos algorithmes. L'idée reste la même, nous générons des objets qui vont suivre une définition de type, puis nous regroupons les objets identiques de manière à obtenir le typage parfait des données et nous convertissons ce typage parfait en instance du p-médian.

L'algorithme est simple, nous commençons par créer nos définitions de types.

Algorithme 3 Generation_BDD(q, k, n)

Require: q: Nombre de définition de type à créer
Require: k: Nombre de liens typés différents par objet
Require: n: Nombre d'objets à générer
Ensure: mat: Matrice contenant les distances entre les différents types du typage parfait (τ)

```
1: for i de 1 à q do
2:    td[i] ← genererDefinition(k)
3: end for
4: for i de 1 à n do
5:    j ← rand()%q
6:    type[i] ← genererType(td[j], k)
7: end for
8: n' ← perfectTyping(type, τ)
9: calcDist(τ,n',mat)
```

Ces définitions sont simplement générées aléatoirement, chaque probabilité est un nombre aléatoire compris entre 0 et 1. Ensuite, pour chaque objet que nous devons générer nous déterminons la définition de type qu'il va suivre. Suivant cette définition l'ensemble des liens typés qu'il possède est créé. Nous générons simplement un vecteur de $\{0,1\}^k$. Une fois tous les objets générés, nous les regroupons s'ils sont identiques de manière a obtenir le typage parfait minimal de ces données. Ensuite, il nous suffit de transformer ce typage parfait minimal en une instance du p-médian. Cette étape est relativement simple, étant donnés deux types τ_i et τ_j nous calculons la distance $d(\tau_i, \tau_j)$ qui est simplement la distance de Manhattan. Nous pouvons alors simplement calculer les coûts du p-médian en posant $c_{ij} = d(\tau_i, \tau_j) * w_i$ où w_i est le nombre d'objet des données ayant le type τ_i.

L'ensemble de toutes ces étapes est récapitulé dans l'algorithme 3.

Nous avons généré plusieurs instances pour le problème du p-médian en utilisant cette technique. Les valeurs des paramètres utilisés pour générer chaque instance sont visibles dans le Tableau 2. Chaque instance est représenté par un numéro (colonne 1). Les valeurs des paramètres q, k et n de l'algorithme 3 sont repportées dans les colonnes "# Def Type", "# Liens Typés" et "# Objets". La dernière colonne "# Typage Parfait Minimal" donne les valeurs du nombre de type formant le typage parfait minimal. Cette valeur est aussi le nombre de sommets de l'instance du p-médian générée.

2.3.4 Génération de données à partir d'un DAG

Dans cette technique de génération de données nous avons souhaité rester plus proche de la forme des bases de données. Dans leur article [85] les auteurs représentent leur base de donnée de départ sous la forme d'un graphe dirigé labellisé. Cette représentation peut-elle être générée directement ? Si oui il ne nous faudra plus que quelques étapes pour obtenir une instance du p-médian. Du graphe nous déduisons l'ensemble des liens typés différents existants, puis nous calculons l'ensemble des types présents dans la base en fonction de ces liens typés. De là, nous calculons le typage parfait minimal de la base, puis l'instance

Num	# Def Type	# Liens typés	# Objets	# Typage Parfait Minimal
1	4	15	4000	2528
2	4	15	4000	2758
3	4	15	4000	2742
4	4	15	4000	2389
5	4	15	4000	2766
6	4	20	4000	3491
21	1	40	4000	3999
22	1	60	4000	4000
23	1	100	15000	15000
24	1	80	4000	4000
25	1	100	4000	4000
26	1	40	6000	5999
27	1	40	8000	8000
28	1	40	10000	9999
29	1	40	9000	8990
30	1	40	20000	20000
32	1	40	15000	15000
33	1	40	15000	14960
34	1	40	15000	14997
35	1	40	16000	15994

Tableau 2: Liste des instances du p-médian générées aléatoirement à partir d'une liste d'objets respectant des définitions de types avec les paramètres employés pour leur génération.

de p-médian associée.

Le principal problème est ici de générer un graphe qui puisse être un graphe représentant une base de données semi-structurée. Pour ce faire nous avons choisis de générer des graphes dirigés acycliques ou DAGs (Directed Acyclic Graphs) suivant la méthode proposée par Melançon et al. [81].

La méthode utilisée pour générer un DAG est assez simple. Partant de G_0 le graphe vide à n sommets nous allons numéroter tous nos graphes, G_i étant le graphe obtenus à la $i$ème itération. Nous tirons à chaque itération un arc potentiel, $e = (u, v)$. Si cet arc existe déjà alors nous le retirons du graphe, soit $G_{i+1} = G_i \backslash e$. Si cet arc n'existe pas dans le graphe, et si le fait de l'ajouter ne créer pas de cyle alors nous l'ajoutons, soit $G_{i+1} = G_i \cup e$. Si rien de tout cela n'est vrai alors on ne modifie pas G_i et $G_{i+1} = G_i$.

Cette technique de génération est longue. D'après les auteurs de [81] il faut n^2 itérations de leur algorithme pour se rapprocher d'une distribution normale des arcs dans le graphe. Il faut donc effectuer au moins n^2 itérations de l'algorithme pour générer un tel graphe, avec n le nombre de sommets du graphe. Chaque itération ne pouvant être effectuée en moins de $O(n^2)$ la complexité de génération d'un graphe est donc $O(n^4)$. Cette complexité est beaucoup trop grande pour pouvoir générer des graphes et les utiliser ensuite.

2.3.5 Environement de test

L'ensemble des tests ont été effectués sur un serveur HP Proliant DL 785 8356, avec 8 processeurs AMD Opteron 2.3 GHz Quad Core (Soit 32 cœurs logiques), avec $32 * (2 * 4Gb) = 256Gb$ de mémoire vive (PC2-5300) et muni du système d'exploitation Red Hat (4.1.2). Les programmes ont été codés en C++ et compilés avec g++ (gcc version 4.1.2) et avec l'option de compilation -O2. Les temps mesurés pour les algorithmes séquentiels sont les temps retournés par la fonction `getrusage`() et correspondent aux temps d'utilisation du processeur par le programme. Les temps mesurés pour les algorithmes parallèles sont les temps réels renvoyés par la fonction `gettime`(). Ces temps n'inculent pas le temps passé à lire une instance depuis le disque. Le générateur de nombres aléatoires utilisé est `drand48`().

Chapitre 3

Résolution du problème du p-médian

Pour résoudre un programme linéaire il existe plusieurs techniques. La première et la plus simple, est d'utiliser un solveur de programmes linéaires, comme CPLEX, de manière à obtenir une solution optimale du problème. CPLEX base sa résolution sur l'algorithme du simplexe. Cependant cette technique bien qu'efficace en terme de qualité des solutions ne l'est absolument pas en terme de temps de calculs pour notre problème. Nous avons du nous tourner vers une solution plus rapide, l'algorithme du Volume.

3.1 L'algorithme du Volume

L'algorithme du Volume a été mis au point par Barahona et Anbil [2] pour combler certaines lacunes de l'algorithme du sous-gradient. L'algorithme du sous-gradient est un algorithme très utilisé depuis les travaux de Held et Karp [62, 63] et ceux de Held, Wolfe et Crowder [64] dans le début des années 70. Il a été surtout utilisé pour trouver des bornes pour des problèmes de grandes tailles [84]. Cet algorithme bien qu'efficace en temps de calculs présente quand même certaines lacunes. Même s'il permet de trouver une bonne estimation d'une solution duale optimale, il ne permet pas d'avoir une solution primale du problème. Il faut donc appliquer d'autres techniques après la fin de l'algorithme du sous-gradient pour obtenir les variables primales, techniques principalement basées sur les écarts complémentaires. L'algorithme du Volume est une extention de cet algorithme qui permet de produire une approximation des variables primales tout en concervant un temps de calcul faible. De plus, cet algorithme permet d'avoir un bien meilleur critère d'arrêt.

3.1.1 Relaxation Lagrangienne et sous-gradient

L'algorithme du Volume est basé sur la méthode du sous-gradient, et plus particulièrement sur la résolution d'une relaxation Lagrangienne du problème à résoudre. Considérons le programme linéaire suivant :

$$\begin{cases} \text{minimiser } c^T x \\ Ax = b \\ Dx = e \\ x \geq 0 \end{cases} \tag{3.1}$$

et supposons que

$$\{x | Dx = e, x \geq 0\} = \{x | x = \sum \lambda_i g_i, \sum \lambda_i = 1, \lambda \geq 0\}.$$

En fait, les g_i sont les points extrêmes du polytope défini par $Dx = e$ et $\lambda_1, \ldots, \lambda_n$ ce sont les variables. La méthode du sous-gradient est basée sur le fait que (3.1) est équivalent au problème (3.2).

$$\begin{cases} \text{minimiser } \sum (c^T g_i)\lambda_i \\ \sum (Ag_i)\lambda_i = b \\ \sum \lambda_i = 1 \\ \lambda \geq 0 \end{cases} \tag{3.2}$$

et son problème dual est

$$\begin{cases} \text{maximiser } z \\ z + \pi(Ag_i - b) \leq cg_i \end{cases} \tag{3.3}$$

L'algorithme du sous-gradient est décrit par l'Algorithme 4.

Algorithme 4 SubGradient(P)

1: Etant donné $\bar{\pi}$ trouver une inégalité serrée dans (3.3) en résolvant

$$\begin{cases} \text{minimiser } z = (c - \bar{\pi}A)x + \bar{\pi}b \\ Dx = e \\ x \geq 0 \end{cases} \tag{3.4}$$

2: Soit $\bar{x}$ une solution de ce problème.
3: $v = b - A\bar{x}$ # *v est un sous-gradient pour $\bar{\pi}$*
4: $\bar{\pi} = \bar{\pi} + sv$ # *s est la taille du pas, pour le calculer la formule usuelle est*

$$s = f\frac{UB - \bar{z}}{\|v\|^2} \tag{3.5}$$

où f est un nombre entre 0 et 2 et UB une borne supérieure pour la valeur de l'optimum.

Pour résoudre (3.2) nous utilisons la génération de colonnes. Nous commençons avec un certain nombre de solutions extrêmes de $Dx = e$, ensuite pour définir la nouvelle colonne à ajouter nous résolvons le problème (3.3). C'est la méthode de Dantzig-Wolfe.

Ceci fonctionne aussi si certaines des égalités de $Ax = b$ sont des inégalités, à ce moment là il faut prendre $max\{\bar{\pi}_i, 0\}$ pour les valeurs duales, car elles ne peuvent pas être négatives. En général f décroit après un certain nombre

d'itérations sans amélioration. Le critère d'arrêt est un certain nombre total d'itérations ou un nombre d'itérations sans amélioration.

Si le sous-problème (3.4) peut être résolu de manière efficace alors chaque itération a un coût de calcul faible. Cette procédure s'est révellée efficace dans de multiples cas de la litérature mais elle ne permet pas de générer un vecteur λ qui soit une approximation de la solution optimale de (3.2).

L'algorithme du Volume donne non seulement une borne duale mais aussi une solution approchée du point extrême optimal du problème dual. Cet algorithme est décrit par l'Algorithme 5.

Algorithme 5 Volume(P)

1: On commence avec un vecteur $\bar{\pi}$ et on résoud (3.4) pour obtenir $\bar{x}$ et $\bar{z}$.
2: $t \leftarrow 1$
3: Calculer $v^t = b - A\bar{x}$ et $\pi^t = \bar{\pi} + sv^t$ pour un pas s calculé grâce à (3.5).
4: Résoudre (3.4) avec π^t, soient x^t et z^t les solutions obtenues.
5: $\bar{x} \leftarrow \alpha x^t + (1-\alpha)\bar{x}$ # *où* $0 \leq \alpha \leq 1$
6: ***if*** $z^t > \bar{z}$ ***then***
7: $\quad \bar{\pi} \leftarrow \pi^t$
8: $\quad \bar{z} \leftarrow z^t$
9: ***end if***
10: $t \leftarrow t + 1$
11: *Retour à 3.*

Si les résolutions successives de (3.4) donnent les solutions $x^0, ..., x^t$ alors

$$\bar{x} = \alpha x^t + (1-\alpha)x^{t-1} + ... + (1-\alpha)^t x^0$$

Donc $\bar{x}$ est une combinaison convexe de $\{x^0, ..., x^t\}$. La valeur α peut être fixée au début de l'algorithme et peut être diminuée au cours de l'éxécution. L'idée utilisée dans l'algorithme du Volume est reprise de la méthode du sous-gradient conjugué [106, 74]. Soit $\bar{v} = b - A\bar{x}$, $v^t = b - Ax^t$, et α_{max} une borne supérieure sur α. Alors on calcul α_{opt} comme la valeur qui minimise $\|\alpha v^t + (1-\alpha)\bar{v}\|$. Si $\alpha_{opt} < O$ alors on pose $\alpha = \alpha_{max}/10$, sinon $\alpha = min\{\alpha_{opt}, \alpha_{max}\}$.

En ce qui concerne la valeur de f dans le calcul de s (à chaque itération un pas s doit être calculé, la formule classique pour ce calcul est $s = f\frac{UB-\bar{z}}{||v||^2}$) trois types d'itérations sont définis. Chaque itération qui n'apporte aucune amélioration à la valeur de $\bar{z}$ est appellée rouge. Si une amélioration est trouvée alors on calcul $d = v^t(b - Ax^t)$. Si $d < 0$ ceci veut dire qu'une avancée plus grande dans cette direction aurait amenée une valeur plus petite pour z^t, ces itérations sont appelées jaunes. Si $d > 0$ l'itération est appelée verte. A chaque itération verte f est multiplié pas 1.1. Après une séquence de 20 itérations rouges f est multiplié par 0.66.

3.1.2 Application au problème du p-médian

L'utilisation de l'algorithme du Volume (voir l'Algorithme 5) repose sur la résolution du programme linéaire (3.4). Si cette résolution est rapide, alors la résolution du problème initial sera rapide, comme pour le sous-gradient. Nous allons maintenant voir comment appliquer cet algorithme à notre problème.

Rappelons la relaxation linéaire associée au problème du p-médian :

$$Z(p\text{MP}(G)) = \text{minimiser} \sum_{(u,v)\in A} w(u)d(u,v)x(u,v) \quad (3.6)$$

$$\sum_{v\in V} y(v) = p, \quad (3.7)$$

$$\sum_{v:(u,v)\in A} x(u,v) = 1 - y(u) \quad \forall u \in V, \quad (3.8)$$

$$x(u,v) \leq y(v) \quad \forall (u,v) \in A, \quad (3.9)$$

$$y(v) \geq 0 \quad \forall v \in V, \quad (3.10)$$

$$x(u,v) \geq 0 \quad \forall (u,v) \in A. \quad (3.11)$$

Et les contraintes à relaxer sont les contraintes (3.8) et ainsi nous obtenons le sous-problème Lagrangien suivant qui correspond au problème (3.4) :

$$L(\pi) = \text{Minimiser} \sum_{(u,v)\in A} \bar{c}(u,v)x(u,v) - \sum_{u\in V} \pi(u)y(u) + \sum_{u\in V} \pi(u) \quad (3.12)$$

$$\sum_{u\in V} y(u) = p, \quad (3.13)$$

$$x(u,v) \leq y(v) \quad \text{for all } (u,v) \in A, \quad (3.14)$$

$$y(u) \geq 0 \quad \text{for all } u \in V, \quad (3.15)$$

$$x(u,v) \geq 0 \quad \text{for all } (u,v) \in A. \quad (3.16)$$

où $\bar{c}(u,v) = w(u)d(u,v) - \pi(u)$.

Le calcul de $L(\pi)$ est facile [18]. et il est résumé dans l'Algorithme 6. Le principe est simple. Pour chaque sommet v du graphe on calcul le potentiel, $\mu(v)$ de ce sommet. Ce potentiel est tout simplement l'apport de ce sommet à la fonction objectif s'il était choisi en tant que centre. Ces potentiels sont calculés de la ligne 1 à la ligne 3. Si nous supposons que le sommet v est ouvert, alors, sa participation dans la fonction objectif est simplement

$$\sum_{u:(u,v)\in A} (\bar{c}(u,v))^{-} - \pi(v)$$

Ensuite, il est clair que si v est un centre et que l'on a un sommet u tel que $\bar{c}(u,v) \leq 0$ alors le fait de poser $x(u,v) = 1$ va diminuer la valeur de la fonction objectif qui est initialement 0 puisque $x = 0$ et $y = 0$ est une solution réalisable pour le sous-problème Lagrangien. Ainsi, pour chaque sommet le potentiel $\mu(v)$ représente la modification de $L(\pi)$ si le sommet v était ouvert comme centre. Donc l'ouverture de v réduira $L(\pi)$ que lorsque $\mu(v)$ est négatif. Pour trouver la solution optimale, il suffit de prendre les p plus petits potentiels (lignes 4-7), et de choisir comme centre les sommets correspondants. Une fois les p centres définis, il suffit de relier les autres sommets du graphe à eux, si et seulement si le coût réduit de l'arc est négatif (lignes 8-12).

Algorithme 6 Résolution du sous-problème Lagrangien

Require: π : vecteur des multiplicateurs duaux
Ensure: (x, y) : la solution optimale du sous-problème
1: **for** $v \in V$ **do**
2: $\mu(v) = \sum_{u:(u,v)\in A} (\bar{c}(u,v))^- - \pi(j)$
3: **end for**
4: $J^* = \{v_1, v_2, \ldots, v_p\}$ tel que $\mu(v_1) \leq ... \leq \mu(v_p) \leq \mu(v_{p+1}) \leq ... \leq \mu(v_n)$
5: **for** $v \in J^*$ **do**
6: $y(v) = 1$
7: **end for**
8: **for** $(u, v) \in A$, $v \in J^*$ **do**
9: **if** $\bar{c}(u, v) \leq 0$ **then**
10: $x(u, v) = 1$
11: **end if**
12: **end for**
13: Toute variable non mis à 1 est mise à 0.

Ainsi pour un vecteur π donné nous avons un algorithme linéaire en fonction du nombre d'arcs permettant de calculer $L(\pi)$.

L'algorithme du Volume appliqué à notre problème s'écrit comme suit :

Algorithme 7 Algorithme du Volume appliqué au problème du p-médian

1: Poser $\bar{\pi} = 0$;
2: Calculer $L(\bar{\pi})$ en résolvant (3.12)-(3.16); soit $(\bar{x}, \bar{y})$ la solution optimale obtenue; $Target = 10$; Poser $t = 1$;
3: Calculer v^t où $v^t(u) = 1 - \bar{y}(u) - \sum_{w:(u,w)\in A} \bar{x}(u, w)$;
4: Calculer $s = \lambda \frac{Target - L(\bar{\pi})}{||v^t||^2}$; Calculer $\pi^t = \bar{\pi} + sv^t$;
5: Calculer $L(\pi^t)$ en résolvant (3.12)-(3.16); soit (x^t, y^t) la solution optimale obtenue;
6: Mettre à jour $(\bar{x}, \bar{y})$, $(\bar{x}, \bar{y}) = \alpha(x^t, y^t) + (1 - \alpha)(\bar{x}, \bar{y})$ où $0 \leq \alpha \leq 1$ et $z(\bar{x})$;
7: Mettre à jour $Target$, $Target = 1.1L(\pi^t)$;
8: Si $L(\pi^t) > L(\bar{\pi})$, mettre à jour $\bar{\pi}$ et $L(\bar{\pi})$, $\bar{\pi} = \pi^t$ et $L(\bar{\pi}) = L(\pi^t)$;
9: Mettre à jour UB toutes les k itérations ;
10: Si Stop-test est faux, $t = t + 1$ et aller à la ligne 3.

Dans l'Algorithme 7 le λ utilisé est le même que celui utilisé dans [12].

3.2 Résultats de calcul préliminaires

Nous avons réalisé un premier programme pour mettre à l'épreuve l'efficacité de cet algorithme pour la résolution de notre problème. Ce premier programme était basé sur l'implémentation de l'algorithme du Volume faite par Barahona et Ladanyi [13] et mise à disposition au sein du projet COIN-OR (COmputational INfrastructure for Operations Research) [37]. Cette implémentation se présente sous la forme d'une bibliothèque pour le C++. Quelques classes

sont présentes dans cette bibliothèque mais ce sont surtout des classes internes à la bibliothèque, une seule est destinée à être spécialisée pour acceuillir le code spécifique à notre problème. L'implémentation complète sera détaillée plus loin. Dans les exemples fournis avec la bibliothèque, on peut trouver une implémentation pour le problème de Localisation de Dépôts Sans Capacités (LDSC). Nous l'avons adaptée au problème du p-médian.

Tout d'abord nous avons comparé CPLEX et l'algorithme du Volume en ce qui concerne à la fois le temps de calcul et la qualité de notre relaxation.

CPLEX est un solveur linéaire édité par la société Ilog. Il permet, entre autre, de résoudre des programmes linéaires. Dans un premier temps nous avons évalué les performances de CPLEX pour résoudre la relaxation linéaire du p-médian. Les résultats de cette étude sont présentés dans le Tableau 3.

Ces premiers résultats ne concernent que la résolution de la relaxation linéaire de notre problème et sont déjà très longs. Sur plusieurs instances de petites tailles[1] CPLEX n'arrive pas à trouver la solution en moins de 3h. Nous avons cependant laissé CPLEX terminer ses calculs pour l'instance *fl3795* pour $p = 300$. CPLEX nous a donné la solution après plus de 14h de calculs.

Nous avons souhaité accélerer les temps de résolution et avons considéré l'algorithme du Volume. Les tableaux 4 et 5 présentent la comparaison entre la résolution, par CPLEX et par l'algorithme du Volume, de la relaxation linéaire de notre problème.

Dans ces tableaux nous calculons le gap entre la solution de CPLEX et la solution de l'algorithme du Volume de la manière suivante :

$$Gap = \frac{LB_{CPLEX} - LB_{Volume}}{LB_{CPLEX}}$$

Ces résultats sont très encourageants. En effet on remarque que sur cet ensemble d'instances prises dans la TSPLIB[2] l'algorithme du Volume est compétitif en terme d'approximation de la solution de la relaxation linéaire, mais en plus très efficace en temps. Dans la majorité des cas le gap ne dépasse pas 0.01%. Dans tous les cas il est inférieur à 0.4%. La valeur du gap la plus élevée est 0.36%, mais nous avons peu de valeurs supérieures de 0.2%. L'algorithme du Volume fourni donc pour la relaxation linéaire de notre problème une excellente estimation de la solution.

En terme de temps on peut noter une réelle accélération des calculs. Cette fois nous résolvons la relaxation linéaire de notre problème pour des tailles de graphes allant de 1304 à 3795 sommets. Ces tailles sont encore dans l'ensemble que l'on qualifie de petites.

Le Tableau 6 donne des résultats de l'algorithme du Volume pour des tailles plus importantes du problème. On voit clairement que le temps de résolution augmente avec la taille du problème que l'on souhaite résoudre. Cependant ce n'est pas la seule valeur influençant les résultats, en effet il faut aussi tenir compte de la valeur de p.

[1] Pour nous, les petites tailles sont toutes les taille inférieure à 5000 sommets. Les tailles comprisent entre 5000 et 10000 sommets sont des tailles moyennes, les tailles de graphe supérieures à 10000 sommets sont des instances de grandes tailles.

[2] La tsplib est une bibliothèque d'instances pour le problème du voyageur de commerce (Traveler Salesman Problem). Elle regroupe des instances allant de 16 à 13508 sommets qui sont répartis sur un plan. Les coûts sur les arcs sont simplement les distances entre les sommets. C'est une bibliothèque très utilisée dans la litérature pour le problème du p-médian. voir aussi section 2.3.2

Instance		Cplex RL	
Label	p	LB_{RL}	Temps
rd400	5	67 909	57,35
rd400	10	45 905	69,76
rd400	20	31 755,80	60,28
rd400	50	17 136	17,84
rd400	100	9 950	10,48
rd400	200	4 424,33	7,51
rd400	300	1 445	3,61
pr1002	5	1 922 870	865,26
pr1002	10	1 262 890	354,97
pr1002	20	868 732	1 067,73
pr1002	50	503 199	569,67
pr1002	100	331 165	257,72
pr1002	200	199 940	187,16
pr1002	300	139 127	153,17
pr1002	400	103 988	128,41
pr1002	500	78 245	63,99
rl1304	5	3 099 070	4 582,28
rl1304	10	–	>3h
rl1304	20	1 412 110	3 326,96
rl1304	50	795 012	1 232,08
rl1304	100	491 506	995,36
rl1304	200	268 573	616,46
rl1304	300	177 318	371,04
rl1304	400	128 332	256,15
rl1304	500	97 018	216,22
fl1400	5	174 877	974,18
fl1400	10	100 601	710,58
fl1400	20	57 191	571,26
fl1400	50	28 486	530,68
fl1400	100	15 960,90	762,91
fl1400	200	8 792,29	906,01

Instance		Cplex RL	
Label	p	LB_{RL}	Temps
fl1400	300	6 091,81	738,03
fl1400	400	4 635,43	445,05
fl1400	500	3 755,73	605,88
u1432	5	1 210 130	2 321,94
u1432	10	849 759	8 004,57
u1432	20	588 720	9 319,99
u1432	50	361 723	8 758,85
u1432	100	243 758	1 718,18
u1432	200	159 867	779,06
u1432	300	123 674	708,03
u1432	400	103 410	1 659,25
u1432	500	93 200	255,37
vm1748	5	4 479 420	6 988,96
vm1748	10	–	>3h
vm1748	20	–	>3h
vm1748	50	1 004 320	2 743,74
vm1748	100	636 417	2 457,22
vm1748	200	390 350	1 363,55
vm1748	300	286 036	926,94
vm1748	400	221 522	557,56
vm1748	500	176 976	562,72
d2103	5	–	>3h
d2103	10	–	>3h
d2103	20	–	>3h
d2103	50	–	>3h
d2103	100	–	>3h
d2103	200	117 735	2 132,20
d2103	300	90 423	2 225,22
d2103	400	75 290	1 625,45
d2103	500	63 951	1 446,77
fl3795	300	39 547	51 713

Tableau 3: Temps de calculs obtenus avec CPLEX lors de résolution de la relaxation linéaire du p-médian pour des instances de la **TSPLIB**. La colonne "Label" indique le nom de l'instance considérée. Le nombre après la série de lettre est la taille de l'instance. La seconde colonne indique le nombre de centres à sélectionner. La colonne LB_{RL} donne la valeur de la fonction objectif fournie par CPLEX. la dernière colonne donne le temps mis par CPLEX pour fournir la solution.

En résumé nous pouvons affirmer que l'algorithme du Volume peut être utilisé pour résoudre la relaxation linéaire des problèmes de petites et moyennes tailles sans aucune difficulté.

Instance		Valeurs			Temps	
Label	p	LB_{CPLEX}	LB_{Volume}	Gap	T_{CPLEX}	T_{Volume}
rl1304	5	3 099 070	3 097 975,53	0,0353 %	4 582,28	88
rl1304	10	–	2 131 352,75	N/A	>3h	133
rl1304	20	1 412 110	1 412 082,22	0,0019 %	3 326,96	116
rl1304	50	795 012	794 950,53	0,0077 %	1 232,08	114
rl1304	100	491 506	491 447,28	0,0119 %	995,36	108
rl1304	200	268 573	268 423,79	0,0555 %	616,46	107
rl1304	300	177 318	177 225,97	0,0519 %	371,04	90
rl1304	400	128 332	128 269,98	0,0483 %	256,15	85
rl1304	500	97 018	96 953,53	0,0664 %	216,22	85
fl1400	5	174 877	174 877,00	0 %	974,18	98
fl1400	10	100 601	100 572,58	0,0282 %	710,58	145
fl1400	20	57 191	57 190,19	0,0014 %	571,26	159
fl1400	50	28 486	28 483,92	0,0072 %	530,68	174
fl1400	100	15 960,90	15 942,31	0,1164 %	762,91	160
fl1400	200	8 792,29	8 780,72	0,1316 %	906,01	128
fl1400	300	6 091,81	6 083,02	0,1442 %	738,03	131
fl1400	400	4 635,43	4 631,83	0,0775 %	445,05	126
fl1400	500	3 755,73	3 753,04	0,0715 %	605,88	101
u1432	5	1 210 130	1 209 793,41	0,0278 %	2 321,94	105
u1432	10	849 759	849 758,12	0,0001 %	8 004,57	130
u1432	20	588 720	588 612,77	0,0182 %	9 319,99	154
u1432	50	361 723	361 562,77	0,0442 %	8 758,85	139
u1432	100	243 758	243 675,73	0,0337 %	1 718,18	137
u1432	200	159 867	159 717,64	0,0934 %	779,06	132
u1432	300	123 674	123 612,45	0,0497 %	708,03	127
u1432	400	103 410	103 304,23	0,1022 %	1 659,25	129
u1432	500	93 200	92 940,64	0,2782 %	255,37	96
vm1748	5	4 479 420	4 479 357,97	0,0013 %	6 988,96	209
vm1748	10	–	2 982 171,93	N/A	>3h	267
vm1748	20	–	1 898 282,82	N/A	>3h	328
vm1748	50	1 004 320	1 003 523,52	0,0793 %	2 743,74	306
vm1748	100	636 417	635 573,24	0,1325 %	2 457,22	280

Tableau 4: Comparaison en temps et en valeur sur de petites instances entre l'algorithme du Volume et CPLEX pour la résolution de la relaxation linéaire du p-médian. La colonne "Label" indique le nom de l'instance considérée. Le nombre après la série de lettres est la taille de l'instance. La seconde colonne indique le nombre de centres à sélectionner. La colonne LB_{CPLEX} donne la valeur de la relaxation linéaire fournie par CPLEX. La colonne LB_{Volume} donne la valeur de la dernière résolution du sous problème Lagrangien lors de l'exécution de l'algorithme du Volume. La colonne "Gap" donne la différence relative entre ces deux valeurs. Le symbole "N/A" signifie que cet indicateur ne peutêtre calculé. Les colonnes "Temps" donnent les temps d'exécution des deux méthodes.

3.3 Parallélisation de la résolution du problème du p-médian

L'algorithme du Volume est un algorithme de résolution de programmes linéaires qui fourni une estimation de la solution optimale. C'est un algorithme rapide

Instance		Valeurs			Temps	
Label	p	LB_{CPLEX}	LB_{Volume}	Gap	T_{CPLEX}	T_{Volume}
vm1748	200	390 350	389 964,78	0,0986 %	1 363,55	246
vm1748	300	286 036	285 852,36	0,0641 %	926,94	242
vm1748	400	221 522	221 396,82	0,0565 %	557,56	233
vm1748	500	176 976	176 893,31	0,0467 %	562,72	203
d2103	5	–	1 005 112,56	N/A	>3h	312
d2103	10	–	687 233,05	N/A	>3h	353
d2103	20	–	482 739,72	N/A	>3h	372
d2103	50	–	301 203,35	N/A	>3h	397
d2103	100	–	194 034,73	N/A	>3h	461
d2103	200	117 735	117 583,74	0,1284 %	2132,2	327
d2103	300	90 423	90 090,26	0,3679 %	2225,22	328
d2103	400	75 290	75 176,66	0,1505 %	1625,45	323
d2103	500	63 951	63 885,72	0,1020 %	1446,77	312
pcb3038	5	–	1 777 479,81	N/A	–	598
pcb3038	10	–	1 211 703,92	N/A	–	593
pcb3038	20	–	838 924,69	N/A	–	747
pcb3038	50	–	505 993,96	N/A	–	731
pcb3038	100	–	351 244,63	N/A	–	744
pcb3038	150	–	279 933,94	N/A	–	722
pcb3038	200	–	237 099,72	N/A	–	596
pcb3038	300	–	186 623,88	N/A	–	685
pcb3038	400	–	156 210,50	N/A	–	637
pcb3038	500	–	134 668,08	N/A	–	580
fl3795	5	–	1 052 611,97	N/A	–	911
fl3795	10	–	520 754,53	N/A	–	1 158
fl3795	20	–	319 686,43	N/A	–	1 287
fl3795	50	–	150 934,87	N/A	–	1 386
fl3795	100	–	88 050,43	N/A	–	1 364
fl3795	150	–	65 756,90	N/A	–	1 340
fl3795	200	–	53 767,27	N/A	–	1 288
fl3795	300	39 547	39 546,29	0,0025 %	51713	1 464
fl3795	400	–	31 319,05	N/A	–	1 226
fl3795	500	–	25 957,69	N/A	–	1 409
fl3795	600	–	22 161,02	N/A	–	1 239
fl3795	700	–	19 620,24	N/A	–	1 062
fl3795	800	–	17 332,90	N/A	–	1 323
fl3795	900	–	15 200,67	N/A	–	1 304
fl3795	1 000	–	13 627,68	N/A	–	1 240
fl3795	1 100	–	12 698,34	N/A	–	1 054
fl3795	1 200	–	12 122,83	N/A	–	863
fl3795	1 300	–	11 615,24	N/A	–	839
fl3795	1 400	–	11 117,93	N/A	–	800
fl3795	1 500	–	10 613,08	N/A	–	798

Tableau 5: Suite du Tableau 4

Instance			Algorithme du Volume	
Label	n	p	LB_{Volume}	T_{Volume}
bd25	4 000	300	79 962,26	1 215
bd25	4 000	400	76 944,00	1 284
bd25	4 000	500	74 173,84	1 109
bd25	4 000	600	71 566,16	1 122
bd26	5 999	100	33 652,48	3 543
bd26	5 999	500	26 382,25	4 063
bd26	5 999	600	25 385,07	3 708
bd26	5 999	700	24 480,35	3 422
bd26	5 999	800	23 660,81	3 090
bd27	8 000	700	37 849,97	7 709
bd27	8 000	800	36 820,05	8 403
bd27	8 000	900	35 822,35	6 734
bd27	8 000	1 000	34 906,27	6 788
bd29	8 990	800	33 926,96	11 087
bd29	8 990	900	33 061,87	9 635
bd29	8 990	1 000	32 243,60	9 218
bd29	8 990	1 100	31 463,95	9 190
bd28	9 999	500	50 540,84	16 011
bd28	9 999	1 000	44 302,34	12 839
bd28	9 999	1 500	39 935,95	11 854
bd28	9 999	2 000	36 242,82	9 544

Tableau 6: Résultats de l'algorithme du Volume séquentiel sur des instances de petites et moyennes tailles. La colonne "Label" donne le nom de l'instance. Les colonnes "n" et "p" présentent respectivement le nombre de sommets de l'instance et le nombre de centres à ouvrir. La colonne LB_{Volume} donne la valeur de la dernière résolution du sous problème Lagrangien. La colonne T_{volume} donne le temps mis par l'algorithme pour résoudre l'instance.

et efficace qui a su faire ses preuves sur divers problèmes tels des problèmes de partitions, des problèmes de couverture ou encore des problèmes de localisation [12, 13]. Cependant tous les problèmes traités restent de taille modeste. Pour notre application en bases de données ce ne sont pas des graphes avec un millier de sommets que nous souhaitons traiter, mais des graphes avec des dizaines de milliers de sommets. Nous avons donc parallélisé cet algorithme pour profiter de l'existence de plusieurs processeurs sur une même machine.

3.3.1 Vocabulaire

Voici un ensemble de termes propres à la programmation et à la parallélisation d'un programme. Cette liste n'est pas exhaustive mais elle couvre une partie des termes qui seront utilisés par la suite.

- **processeur** ou CPU (de l'anglais *Central Processing Unit*, "Unité centrale de traitement") : composant de l'ordinateur qui exécute les programmes informatiques. C'est l'un des composants essentiels d'un ordinateur.

- **processus** (de l'anglais *process*) : opération complexe exécutable par un ordinateur et définie par :
  - un ensemble d'instructions à exécuter (un programme), parfois situé dans une mémoire morte (fréquent dans les systèmes embarqués notamment), mais le plus souvent chargé depuis une mémoire de masse vers la mémoire vive ;
  - un espace d'adressage en mémoire vive pour stocker la pile, les données de travail, etc. ;
  - éventuellement d'autres ressources, comme des descripteurs de fichier, des ports réseau, etc.

  L'exécution d'un processus dure un certain temps, avec un début et (parfois) une fin. Un processus peut être démarré par un utilisateur par l'intermédiaire d'un périphérique ou bien par un autre processus : les "applications" sont des ensembles de processus.

- **programme** : suite d'opérations pré-déterminées destinées à être exécutées de manière automatique par un appareil informatique en vue d'effectuer des travaux, des calculs arithmétiques ou logiques, ou simuler un déroulement. Un **programme exécutable** est une suite d'instructions machine qui peuvent être exécutées de manière automatique par un processeur.

- **fonction** : portion de code représentant un sous-programme, qui effectue une tâche ou un calcul relativement indépendant du reste du programme. La fonction est l'objet de base permettant de découper le problème global en plus petits calculs.

- **classe** : déclaration des propriétés communes à un ensemble d'objets. La classe déclare des attributs représentant l'état des objets et des méthodes représentant leur comportement. Une classe représente donc une catégorie d'objets. Elle apparaît aussi comme un moule ou une usine à partir de laquelle il est possible de créer des objets. On parle alors d'un objet en tant qu'instance d'une classe (création d'un objet ayant les propriétés de la classe).

- **méthode** : fonction interne à une classe.

- **integer** ou **int** : type d'une variable représentant un entier. La taille occupée par une telle variable varie d'un genre de processeur à un autre. Elle est de 4 octets (pour les processeurs 32 bits) ou de 8 octets (pour les processeurs 64 bits).

- **double** : type d'une variale représentant un nombre réel en double précision. La taille occupée par une telle variable est de 8 octets (pour les processeurs 32 bits) ou de 16 octets (pour les processeurs 64 bits).

3.3.2 Etude de l'algorithme du Volume

Avant de pouvoir paralléliser l'algorithme du Volume, il faut entrer en détail dans le code pour voir où l'on peut agir et comment nous allons pouvoir améliorer

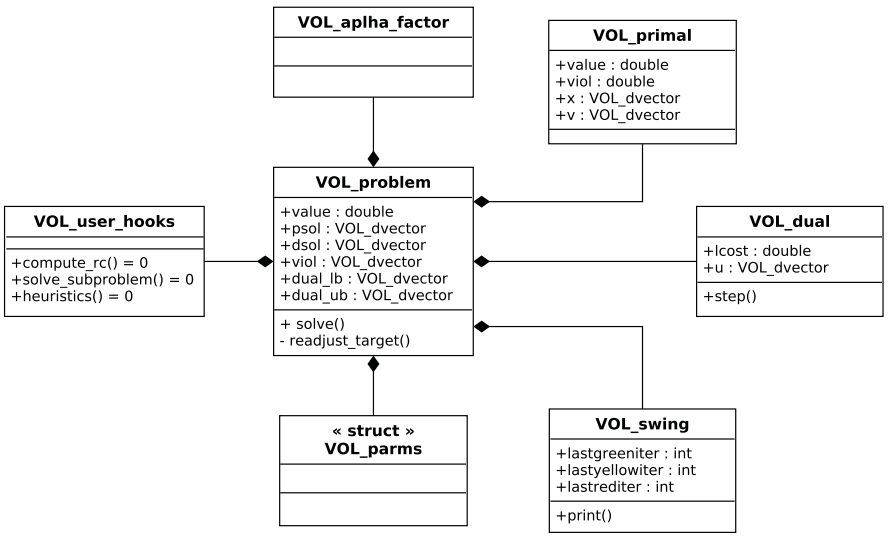


Figure 11: Simplification du diagramme de classe l'implémentation de l'algorithme du Volume.

sa vitesse. Une simplification du diagramme de classe de l'implémentation de l'algorithme du Volume peut être vu sur la Figure 11.

Même si toutes ces classes font partie de l'implémentation de l'algorithme du Volume, une seule fonction est responsable de son bon déroulement. Cette fonction, ou plutôt cette méthode, se trouve dans la classe `VOL_problem`. Cette méthode embarque toute la structure de l'algorithme, c'est donc ici sur cette méthode et dans cette classe que nous allons effectuer nos modifications.

La méthode solve()

Cette méthode reprend simplement l'algorithme général de l'algorithme du Volume. L'implémentation n'étant vraiment pas éloignée de la structure algorithmique générale, il sera plus simple de travailler sur l'algorithme que sur le code produit. L'algorithme du Volume a déjà été vu en détail plus tôt (Algorithme 7). Nous allons le revoir ligne par ligne pour voir ce qui peut être parallélisé et ce qui ne peut pas l'être.

Les lignes 2 et 5 demandent la résolution du sous-problème Lagrangien (voir Algorithme 6). Du point de vue de l'algorithme du Volume cette résolution est codée par l'utilisateur, car elles sont liés au problème et non à l'algorithme lui même. Les lignes 1, 6 et 8 ne demandent que des opérations mineures et ne gagneraient rien à être parallélisées.

Cependant, certaines tâches de l'algorithme peuvent être effectuées en arrière plan. En effet l'algorithme inclue le lancement d'heuristiques pendant la résolution de la relaxation linéaire (ligne 9). Ces heuristiques sont lancées pour obtenir une borne supérieure au problème. Cette borne est utilisé par l'algorithme du Volume pour, entre autre, définir un critère d'arrêt. Ces heuristiques pourraient être lancées sur un processus différent du processus principal, amorçant ainsi la parallélisation des calculs. De plus le calcul de la variable *target* (ligne 7) qui

Algorithme 8 Algorithme du Volume appliqué au problème du p-médian

1: Poser $\bar{\pi} = 0$;
2: Calculer $L(\bar{\pi})$ en résolvant (3.12)-(3.16); soit $(\bar{x}, \bar{y})$ la solution optimale obtenue; $Target = 10$; Poser $t = 1$;
3: Calculer v^t où $v^t(u) = 1 - \bar{y}(u) - \sum_{w:(u,w)\in A} \bar{x}(u,w)$;
4: Calculer $s = \lambda \frac{Target - L(\bar{\pi})}{||v^t||^2}$; Calculer $\pi^t = \bar{\pi} + sv^t$;
5: Calculer $L(\pi^t)$ en résolvant (3.12)-(3.16); soit (x^t, y^t) la solution optimale obtenue;
6: Mettre à jour $(\bar{x}, \bar{y})$, $(\bar{x}, \bar{y}) = \alpha(x^t, y^t) + (1-\alpha)(\bar{x}, \bar{y})$ où $0 \leq \alpha \leq 1$ et $z(\bar{x})$;
7: Mettre à jour $Target$, $Target = 1.1L(\pi^t)$;
8: Si $L(\pi^t) > L(\bar{\pi})$, mettre à jour $\bar{\pi}$ et $L(\bar{\pi})$, $\bar{pi} = \pi^t$ et $L(\bar{\pi}) = L(\pi^t)$;
9: Mettre à jour UB toutes les k itérations ;
10: Si Stop-test est faux, $t = t + 1$ et aller à la ligne 3.

est fait grâce à la fonction *readjust_target*() peut être déporté lui aussi dans un autre processus. Entre le moment où les données sont disponibles pour effectuer ce calcul et le moment où l'on en a besoin l'algorithme du Volume doit effectuer d'autres opérations. Nous avons donc naturellement placé cette fonction dans un processus à part, de manière à ce qu'elle puisse être exécutée en même temps que d'autres calculs.

Parallélisation de la résolution du sous-problème Lagrangien

La partie la plus importante est la résolution du sous-problème Lagrangien. Cette résolution est en effet exécutée à chaque itération de l'algorithme du Volume. Même si son algorithme est simple, pour des problèmes de grandes tailles son temps d'exécution devient important. Nous allons donc intervenir sur cette partie pour accélérer la vitesse d'exécution de l'algorithme. Nous avons déjà vu l'algorithme général de cette résolution, l'Algorithme 9 en est une version plus détaillée.

La notation $[k/n]$ aux lignes 12 et 13 signifie qu'il faut prendre la partie entière du rapport $\frac{k}{n}$. Cet algorithme peut se découper en trois phases. La première (ligne 1) réalise l'initialisation des différentes variables requises pour l'algorithme. La secondes (lignes 2 à 5) permet de calculer pour chaque sommet du graphe son potentiel. Une fois tous les potentiels calculés, la troisième phase de l'algorithme (lignes 7 à 17) les utilise pour créer une solution optimale au problème en choisisant les p potentiels les plus faibles et en reliant les autres sommets du graphe aux centres choisis, en fonction du coût réduit de l'affectation.

Cet algorithme est implémenté dans le premier programme permettant de résoudre notre problème. Il peut facilement être exécuté de façon parallèle. En effet les calculs des $\mu(v)$ pour chaque sommet v du graphe ne dépendent absolument pas des autres calculs ($\mu(v) = \sum_{v:(u,v)\in A} (\bar{c}(u,v))^- - \pi(v)$), mais seulement des entrées de l'algorithme. Nous avons donc développé une version parallèle de cet algorithme en se basant sur le simple fait que les différents $\mu(v)$ peuvent être calculés indépendamment les uns des autres.

Notre plan de travail est donc de découper la méthode *solve*() en plusieurs

Algorithme 9 Résolution du sous-problème Lagrangien (détails)

Require: n taille du problème
Require: p nombre de centre à ouvrir
Ensure: (x, y) Solution du sous-problème Lagragien
1: Initialisation des variables μ, ind, x et y à 0
2: **for all** $v \in V$ **do**
3: $\mu(v) = \sum_{(u,v)\in A} \bar{c}(u,v)^- - \pi(v)$
4: $ind[v] = v$
5: **end for**
6: *# Recherche des p plus petits* $\mu(v)$
7: sort(ind, μ) *fonction triant les valeurs de ind en fonction des valeurs de* μ
8: **for** $i = 1$ to p **do**
9: $y(ind[i]) = 1$
10: **end for**
11: **for** $k = 0$ to $n * n$ **do**
12: $u = [k/n]$
13: $v = k - [k/n]$
14: **if** $u \neq v$ **and** $y(v) = 1$ **and** $\bar{c}(u,v) \leq 0$ **then**
15: $x(u,v) = 1$
16: **end if**
17: **end for**

processus :

- Maître : le processus principal de notre implémentation, il va organiser les calculs et implémenter la boucle principale de l'algorithme du Volume.
- Calculateur : un processus qui va prendre un numéro de sommet en entrée et calculer la composante de μ associée. Ce processus est répliqué un certain nombre de fois de manière effectuer tous ces calculs en parallèle.
- Agregateur : un processus qui va organiser les résultats des Calculateurs de manière à en faire une solution optimale au sous-problème.
- Heuristique : un processus qui va constament exécuter une heuristique pour trouver une solution entière réalisable, soit une borne supérieure pour notre problème.
- TargetComputer : un processus qui va, quand on a toutes les données requises, calculer la nouvelle valeur de la variable *target*, il remplace l'appel de la fonction *readjust_target* dans l'algorithme du Volume.

Le code et l'algorithme du TargetComputeur sont des copies conformes du code et de l'algorithme présents dans l'implémentation originelle. Nous allons maintenant décrire les différents algorithmes de nos différents processus.

3.3.3 Algorithmes de principe

Algorithme des processus Calculateurs

Algorithme 10 Algorithme des processus Calculateurs

```
1: recevoir un sommet v
2: ρ ← 0
3: for u : (u, v) ∈ A do
4:    c̄(u, v) ← c(u, v) − π(u)
5:    if c̄(u, v) < 0 then
6:       stocker(liste, (u, c̄(u, v)))
7:       ρ ← ρ + c̄(u, v)
8:    end if
9: end for
10: μ ← ρ − π(v)
11: envoyer(v, μ, liste)
```

Le principe de l'algorithme est très simple. Ayant reçu de la part du maître un numéro de sommet du graphe en mémoire, le calculateur doit calculer la composante de μ correspondante. Pour rappel le calcul de $\mu(v)$ se fait grâce à la formule suivante :

$$\mu(v) = \sum_{u:(u,v)\in A} (\bar{c}(u,v))^{-} - \pi(v),$$

où $\bar{c}(u,v) = c(u,v) - \pi(u)$ et $\pi(u)$ est le coefficient de Lagrange associé à la relaxation de l'égalité (3.8). Le calcul est fait en plusieurs étapes, tout d'abord un calcul des coûts réduits associés aux arcs entrants du sommet v considéré. Puis si ce coût réduit est négatif, sa valeur et le numéro du sommet d'origine de l'arc sont stockés dans une liste. Une fois les calculs effectués le processus envoit au processus Agregateur trois éléments :

- le numéro du sommet v qui vient d'être traité,
- la valeur de $\mu(v)$,
- la liste des couples $(u, \bar{c}(u,v))$ tels que $\bar{c}(u,v)$ soit négatif. Cette liste est essentielle au processus Agregateur pour construire la solution optimale du sous-problème.

En pratique le processus est un peu plus complexe car il va chercher à recevoir un nouveau numéro de sommet une fois ses calculs effectués. Tant qu'il ne reçoit rien il attend. Dès qu'il reçoit un numéro valide il effectue les calculs, et s'il reçoit un numéro non valide (par exemple négatif) il considère que c'est un ordre de terminaison de la part du processus maître. L'Algorithme 10 est donc inclu dans une boucle infinie de réception, un test est effectué sur le numéro du sommet reçu pour pouvoir la quitter. L'envoi de l'ordre de terminaison est une obligation du processus maître. Si ce dernier ne l'effectue pas, il est possible que ce processus ne se termine jamais et donc que le programme ne se termine jamais non plus.

Algorithme de l'agrégateur

Le processus agrégateur va simplement attendre de recevoir les n informations concernant chacune un sommet du graphe (boucle de la ligne 3 à la

Algorithme 11 Algorithme du processus Agregateur

```
1: initPriorityQueue(FP)
2: nb_recu ← 0
3: while nb_recu < n do
4:    recevoir(u,μ(u),liste)
5:    (μ(u),u,liste) ← destocker(stock[u])
6:    enfiler(FP,μ(u),u)
7:    updateSolution((x^t, y^t),u,liste)
8:    if taille(FP) > p then
9:       v ← defiler(FP)
10:      updateSolution((x^t, y^t),v,liste)
11:   end if
12:   nb_recu ← nb_recu + 1
13: end while
```

ligne 13). Chaque fois qu'il reçoit une information il va la mettre dans une file d'attente prioritaire, la priorité étant simplement la valeur de la composante de μ considérée. Lors de chaque insertion l'appel à la fonction *updateSolution*() permet de mettre à jour la solution (x^t, y^t) en insérant le nouveau sommet parmis les centres. Cette fonction permet aussi de calculer les violations de cette solution. Si la taille de cette file dépasse p alors l'un des élements est en trop et c'est le plus prioritaire. Le processus va alors défiler cet élément (ligne 9) et recalculer la solution et les violatsions (ligne 10).

Encore une fois ici comme nous ne souhaitons pas relancer ce processus à chaque itération l'algorithme 11 est englobé dans une boucle infinie de laquelle il ne pourra sortir qu'en recevant un ordre de terminaison de la part du processus maître. A chaque nouvelle itération le processus maître devra lui envoyer un ordre de continuation, mais cette légère communication redondante n'est rien en comparaison du coût de lancement à chaque itération d'un nouveau processus sur l'ordinateur. De la même manière que pour les processus Calculateurs, il est de la responsabilité du processus maître d'envoyer un ordre de terminaison à ce processus sinon le programme peut ne pas se terminer.

Algorithme du processus Maître

Ici l'algorithme reprend fidèlement l'algorithme du Volume. Les seules modifications sont les communications permettant de synchroniser tous les autres processus. La fonction *testFin*() permet de savoir s'il y a une raison de quitter l'algorithme. Les raisons peuvent être diverses, comme la découverte d'une solution optimale par une heuristique, un gap entre la solution fractionnaire et la solution entière faible, etc.

Nous avons cinq types de processus différents : le maître, les calculateurs, l'Agregateur, le TargetComputeur et un processus dédié à une heuristique. Comme dit plus haut le TargetComputeur n'est pas vraiment intéressant car il se borne à recevoir un ordre de calcul et met à jour la variable *target* par retour au processus maître, son algorithme n'est donc pas détaillé ici. Le processus dédié à une heuristique n'est pas non plus détaillé ici, ses communications sont assez simples et son algorithme reprend celui de l'heuristique qu'il implémente (voir Section 3.4).

Algorithme 12 Algorithme du processus Maître

```
1: initParallel()
2: π̄ ← 0
3: for i de 1 à n do
4:    envoyer(calculateurs, i)
5: end for
6: recevoir(agregateur, L(π̄),(x̄, ȳ),v^t)
7: t ← 1
8: π^t ← π̄ + sv^t
9: while t < nb_it_max do
10:   for i de 1 à n do
11:      envoyer(calculateurs, i)
12:   end for
13:   recevoir(agregateur, L(π^t),(x^t, y^t),v^t)
14:   (x̄, ȳ) ← α(x^t, y^t) + (1 − α)(x̄, ȳ)
15:   if L(π^t) > L(π̄) then
16:      π̄ ← π^t
17:      L(π̄) ← L(π^t)
18:   end if
19:   t ← t + 1
20:   if testFin() then
21:      break
22:   end if
23: end while
```

3.3.4 Gestion des communications

Dans cette partie, nous allons nous intéresser à la gestion des communications inter-processus qui vont exister dans notre programme. Cette communication est très importante car dans les problèmes de parallélisation il ne faut pas perdre de vue un principe important : si le temps d'échange des données est plus important que le temps de calculs, alors la version parallèle de l'algorithme prendra plus de temps que sa version séquentielle. Ce principe est important surtout en considération de la taille des données que l'on va manipuler.

Problèmes de taille

Pour son bon déroulement l'algorithme du Volume a besoin de plusieurs tableaux de taille importante :

- un premier tableau de *double* contenant les coûts du problème ;
- un second tableau de *double* contenant l'estimation de la solution primale optimale ;
- un troisième tableau intermédiaire d'*integer* contenant la solution optimale du sous-problème ;
- un quatrième tableau d'*integer* contenant la meilleure solution entière réalisable du problème ;

- un cinquième tableau de *double* contenant les valeurs des violations des contraintes relaxées :
- un sixième tableau de *double* contenant les valeurs des coefficients Lagrangiens des contraintes relaxées ;

Les quatre premiers tableaux sont de taille $n * n$. Les deux derniers tableaux sont de taille n. On peut donc calculer la place mémoire minimale requise pour stocker toutes ces informations grâce à la relation suivante :

$$taille = n * n * (2 * sizeof(int) + 2 * sizeof(double)) + 2 * n * sizeof(double)$$

d'où pour un graphe de 50 sommets, une taille requise minimale de $50 * 50 * (2 * 8 + 2 * 16) + 2 * 50 * 16 = 119ko$. Cependant, cette taille va augmenter rapidement suivant une fonction parabollique en fonction de n. Ainsi pour $n = 5000$ la taille mémoire requise sera de $1,1Go$ et pour $n = 10000$ il faudra $4,5Go$ de mémoire.

Nous avons des données de taille importante à manipuler. Il faut éviter de les recopier trop souvent car le temps mis à copier ces données est du temps perdu, l'idéal serait même de ne pas les copier du tout. Pourtant il faut bien que les différents processus communiquent entre eux et ait tous accès à certaines de ces données.

- Le maître à besoin de toutes les données pour les transmettre à qui en a besoin ;
- Agregateur a besoin d'avoir accès à la solution entière du sous-problème pour pouvoir l'écrire une fois qu'elle est calculée ;
- les calculateurs ont besoin d'avoir accès aux coûts du problème pour pouvoir effectuer leurs calculs ;
- le processus en charge de l'heuristique a besoin d'avoir accès à la dernière solution optimale du sous-problème comme base de départ.

Utilisation des IPC

Divers outils existent pour permettre à différents processus de communiquer entre eux et sont regroupés sous le terme d'IPC (Inter-Process Communication ou Communication inter-processus). Le Tableau 7 présente une liste non exhaustive des principales méthodes de communication des IPC.

La plupart de ces techniques présentent des avantages et des inconvénients, mais peu d'entres elle permettent de ne pas recopier les données plusieurs fois, d'autre ne sont tous simplement pas adaptées au passage d'informations complexes. Nous pouvons ainsi éliminer directement :

- les fichiers : recopie d'information ;
- les signaux : peu de type d'information possibles, et surtout des information très simples sont possible uniquement, seule la réception du signal peut être faite par le processus fils ;
- les sockets : technique lourde à mettre en place, utile pour une communication sur un réseau, peu utile lors de communication sur la même machine ;

Outil	Système d'exploitation les founissant
Fichiers	La plupart.
Signaux	La plupart.
Sockets	La plupart.
Files de Message	La plupart.
Pipes	Les système POSIX et Windows
Pipes nommés	Les système POSIX et Windows
Sémaphores	Les système POSIX et Windows
Mémoires partagées	Les système POSIX et Windows
Fichiers mappés en mémoire	Les système POSIX et Windows.

Tableau 7: Liste non exhaustive des IPCs disponibles.

- les files de messages : techniques proche du système de messagerie electronique, lourde à mettre en place ;
- les pipes nommé ou non : équivalents au système de fichier ;
- les sémaphores : comme pour les signaux les sémaphores ne peuvent servir qu'à envoyer des informations simples, le sémaphore existe ou non.

Il ne reste donc que deux techniques en concurrence, les mémoires partagées et les fichiers mappés en mémoire. Ces deux techniques reposent sur la même base, une partie de la mémoire est mise à la disposition de tous les processus. Les mémoires partagées sont réellement une portion de mémoire que chaque processus considère comme lui appartenant. Les fichiers mappés en mémoire sont plutôt des fichiers mis en commun pour tous les processus. Plutôt que d'ouvrir un fichier, d'y écrire nos données et de le mapper en mémoire nous avons choisi d'utiliser des mémoires partagées pour nos tableaux. On demande au système de nous réserver un certain espace mémoire, puis on y inscrit nos données. Une fois les données écrites tous les processus peuvent y accéder.

Cette technique présente cependant un problème de important si rien n'est fait pour l'éviter. Tous nos processus peuvent accéder aux données en lecture et en écriture, donc chaque processus peut détruire le travail d'un autre et ceci sans qu'aucune indication de ce fait ne puisse être récupérée. Dans notre cas ce n'est pas un problème puisque, s'il peut y avoir des lectures concurrentes sur ces données il ne peut y avoir d'écriture conccurentes :

- Le tableau contenant l'estimation de la solution primale optimale n'a besoin d'être accessible que par le maître, aucun besoin de le partager avec les autres processus.
- Le tableau des coûts du problème n'est utilisé que par les calculateurs.
- Le tableau des violations est lu par le maître et écrit par les calculateurs.
- Le tableau des coefficients de Lagrange est lu par les calculateurs et écrit par le maître.
- Le tableau de la meilleure solution entière est lu par le maître et écrit par l'heuristique.

- Le tableau de la solution entière du sous-problème est lu par le maître et l'heuristique et écrit par l'Agregateur.

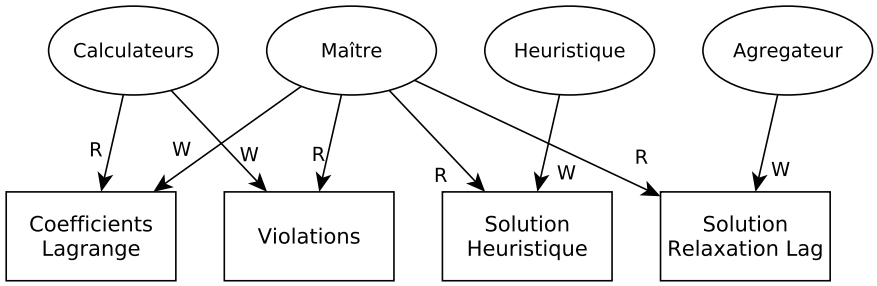


Figure 12: Récapitulatif des accès aux données par les différents processus.

L'ensemble de ces relations est rappelé sur la Figure 12. Nous avons donc choisi d'utiliser des mémoires partagées pour que chaque processus puisse avoir accès aux informations dont il a besoin simplement et rapidement. Il faudra cependant, avant de lire des données dans ces mémoires, faire attention à ce que certains processus aient terminé leur travail.

Mais il existe aussi d'autres types d'informations qu'il faut partager entre les différents processus. Par exemple le Maître doit envoyer les numéro des sommets qu'il faut traiter aux calculateurs. Les calculateurs doivent envoyer leurs résultats à l'Agregateur, lui même devant renvoyer les valeurs de sa solution au Maître. Chacune de ces informations possède l'une ou l'autre des propriétés suivante :

- l'information est de petite taille,
- l'information est de taille inconnue à priori.

Dans les deux cas l'utilisation d'une mémoire partagée n'est pas le plus efficace. Pour des informations de petites tailles il vaut mieux envoyer ces informations sur un moyen de communication direct entre les processus concernés, pour des informations de taille inconnues à priori une mémoire partagée ne sera de toute façon pas adaptée car on ne peut pas modifier sa taille au cours de l'exécution. Les pipes nous donnent accès à trois propriétés intéressantes :

- les lectures sur les pipes sont bloquantes. Le processus qui tente de lire va s'arrêter s'il n'y a rien à lire. Il reprendra son déroulement une fois que l'information sera arrivée sur le pipe. Ceci permet facilement de synchroniser des processus, s'il n'y a rien le processus se met tout seul en attente. Mais cette facilité a un prix, il ne faut pas laisser des processus sans information car ils seront alors bloqués en attente. Si nous voulons qu'un processus se termine il faudra lui envoyer un signal de terminaison ;
- si plusieurs messages sont envoyés sur un pipe alors ils vont être mis les uns au bout des autres, comme dans une file d'attente. Cette propriété est intéressante car on peut simplement envoyer toute une série d'instructions sur un pipe et le processus cible va les traiter les unes à la suite des autres ;

- plusieurs processus peuvent se connecter au même pipe et lire (ou écrire) dessus de manière concurrente. Grâce à ça et à la propriété précédente on peut facilement envoyer une masse importante d'information (comme l'ensemble des numéros des sommets d'un graphe) à un ensemble de processus (nos Calculateurs) en très peu de lignes de code et surtout ne pas avoir à gérer un partage des tâches équitable en temps de calculs (et non pas en nombre de sommets traités).

Toutes ces propriétés font des pipes un outils simple et efficace à utiliser. L'ensemble des relations par pipe peut être vu sur la Figure 13. Les pipes utilisés par le programme sont au nombre de 8 au total.

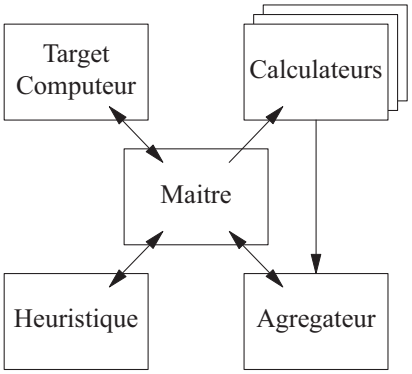


Figure 13: Récapitulatif des échanges par pipes entre les processus.

- 2 pour les communications entre le Maître et le processus TargetComputeur, un dans chaque sens. Le Maître envoit une demande de calcul et la dernière valeur trouvée de la relaxation (donnée requise par TargetComputeur pour effectuer son calcul) pendant les itérations et un signal de terminaison à la fin. Le processus renvoit la nouvelle valeur de la variable $target$.

- 2 pour les communications entre le Maître et le processus heuristique, un dans chaque sens. Le Maître envoit des ordres de démarrage d'heuristiques et un signal de terminaison au processus heuristique. Le processus heuristique envoit au Maître la valeur de la nouvelle solution si une meilleure solution a été trouvée, un message signifiant qu'aucune meilleure solution n'a été trouvée.

- 2 pour les communications entre le Maître et l'Agregateur, un dans chaque sens. Le Maître envoit simplement un signal de début d'itération pour que l'Agregateur se mette en mode récupération d'informations et un signal de terminaison à la fin de l'algorithme. L'Agregateur va quant à lui envoyer au Maître deux valeurs en guise de signal de fin de calcul : la valeur de la solution entière au sous-problème calculée et la valeur de cette même solution avec les coût originaux du problème.

- 1 pour les communications du Maître vers les calculateurs. Le Maître envoit à chaque itération l'ensemble des numéros des sommets du graphe.

u	numéro du sommet concerné par ce message
$\mu(u)$	valeur calculée par le calculateur
nb	nombre de sommets v tels que $\bar{c}(v,u)$ soit négatif
$liste$	la liste des sommets v tels que $\bar{c}(v,u)$ soit négatif

Figure 14: Liste des informations à faire passer d'un processus calculateur au processus XtComputeur

Les calculateurs n'ont alors plus qu'à se servir dans la file des messages et effectuer leurs calculs tant qu'il reste des messages. Si la file est vide, les processus se mettent en attente le temps que la solution soit calculée et qu'une nouvelle itération commence par l'envoi des numéro des sommets. Lorsque l'algorithme atteint la fin le processus maître envoit un signal de terminaison sur ce même canal pour que les calculateurs s'arrêtent.

- 1 pour les communications des calculateurs vers l'Agregateur. Chaque fois qu'un calculateur termine un calcul il envoit un message complexe à l'Agregateur. Ce dernier le reçoit et le traite (voir Section 3.3.4), tant qu'il doit en attendre (le processus sait qu'il doit recevoir n rapports de calculs et en attendra n). Une fois les n communications reçues l'Agregateur pourra passer à l'étape suivante de son calcul (voir l'Algorithme 11).

Remarque sur la trasmition des données entre les calculateurs et l'Agregateur

La plupart des communications qui sont décrites ci-dessus sont simples et se limitent à l'envoi d'une ou deux valeurs entières ou réelles. Ces communications s'implémentent très facilement grâce aux pipes qui permettent l'envoi et la reception de telles données. Par contre la communication entre les calculateurs et l'Agregateur est plus difficile à cause des données qui sont envoyées. En effet envoyer un nombre fixe de données est très facile. Si par exemple nous souhaitons envoyer n données, n ne variant jamais il nous suffit soit d'envoyer n fois une donnée et donc de la lire n fois dans le processus récepteur, soit d'envoyer directement les n données sous la forme d'un tableau, pour peu que la taille de ces données ne soit pas trop grande. Mais si nous souhaitons envoyer un nombre différent à chaque fois d'informations alors il faut mettre en place un protocole de communication plus détaillé. C'est ce qu'il a fallut faire pour la communication entre les calculateurs et l'Agregateur.

Comme nous pouvons le voir sur la Figure 14 l'ensemble des informations à transmettre entre un calculateur et le processus Agregateur contient un certain nombre d'informations :

- le numéro du sommet traité u (entier) : cette information permet à l'Agregateur de savoir à quel sommet les informations suivantes se rapportent ;

- le nombre de sommets v du graphe ayant un arc (v,u) dont le coût réduit est négatif (entier) : cette information permet à l'Agregateur de lire la liste entièrement ;

- la valeur de $\mu(u)$ pour le sommet considéré (double) : cette information permet à l'Agregateur de calculer la valeur de $L(\pi^t)$ pour un vecteur π^t donné ;
- l'ensemble des couples $(v, \bar{c}(v,u))$ tels que $\bar{c}(v,u)$ soit négatif (à chaque fois un entier et un double) : ces valeurs permettent à l'Agregateur de calculer la solution (x^t, y^t).

Ce type de message embarque donc un certain nombre d'informations qui peuvent être envoyées simplement à notre processus recepteur. Nous pourrions simplement envoyer toutes ces informations une par une dans un conteneur adapté (un entier pour envoyer le numéro du sommet traité, une structure regroupant un entier et un double pour les couples $(v, \bar{c}(v,u))$). Ceci fonctionnerait bien, si nous n'avions pas plusieurs processus pouvant écrire de manière concurentielle sur le même moyen de communication. nous aurions pu aussi donner à chaque calculateur un moyen de communication privilégié avec l'Agregateur, mais alors se serait posée la question de l'ordre dans lequel parcourir ces pipes dans le processus récepteur, mais aussi d'arriver à décoder les informations multiples en provenance d'un même calculateur. Même si ces problèmes ne sont pas insolubles, celui de l'ordre de parcours de multiples pipes est un problème important, puisque la lecture sur un pipe est bloquante (s'il n'y a rien sur le pipe que l'on souhaite écouter le processus va attendre que quelque chose arrive).

Nous souhaitons effectuer la transmition de toutes ces données sans qu'il ne puisse y avoir de collision entre les messages des différents processus. Nous avons décidé de tout mettre dans un grand tableau que l'on va remplir. La taille requise pour ce tableau est simple à calculer :

- u : 1 entier, soit 8 octets ;
- v : 1 entier, soit 8 octets ;
- $\mu(u)$: 1 double, soit 16 octets ou encore 2 entiers ;
- l'ensemble des couples $(v, \bar{c}(u,v))$: k fois 1 entier et 1 double, soit k fois 24 octets ou $3k$ entiers.

Connaissant le nombre de couples $(v, \bar{c}(v,u))$ à envoyer, k, la taille totale de ces informations est de $3k + 4$ entiers. Pour envoyer toutes ces données le principe est le suivant : nous allouons en mémoire un tableau de la bonne taille, puis nous écrivons toutes les informations. Ce tableau est ensuite envoyé à l'Agregateur qui va le lire morceau par morceau de manière à reconstruire l'information complète liée au sommet u.

Cependant il existe une limitation à l'utilisation des pipes. Sur chaque système l'administrateur spécifie une taille maximale pour les messages envoyés sur les pipes. Cette taille maximale est limitante car nous ne pouvons pas envoyer d'un seul bloc toutes nos informations dans la majorité des cas. Il faut donc découper nos messages pour les limiter à une taille acceptée par le système. La première idée est de découper notre liste de couples $(v, \bar{c}(v,u))$ en plusieurs sous-listes de manière à ne pas excéder la taille maximale autorisée. Mais tous nos processus peuvent écrire de manière concurente sur le pipe. Il se peut donc qu'entre l'écriture de deux morceaux d'une même liste s'intercale un ou plusieurs messages ayant trait à un ou plusieurs autres sommets.

u	#	$\mu(u)$	k	v_1	$\bar{c}_{v_1,u}$	...	v_k	$\bar{c}_{v_k,u}$

Figure 15: Forme des messages entre un calculateur et l'Agregateur

La solution la plus simple pour gérer ce problème est d'envoyer avec tous les messages le numéro du sommet. Ce qui supprime le problème. Le processus récupérateur peut ainsi savoir à quel sommet a trait un message. Mais comment sait-il qu'un sommet est terminé, c'est-à-dire qu'il a reçu toutes les informations ayant trait à ce sommet particulier ? Pour régler ce soucis, nous avons choisi d'inclure dans nos messages un numéro de message. Si ce numéro est 0, cela signifie que c'est le dernier message de la série. Le processus récepteur devra donc suivre le protocole suivant pour lire les messages :

1. lire les messages sur un pipe ;
2. stocker les informations reçues si le message n'est pas terminal ;
3. une fois toutes les informations reçues, les traiter.

Ces différentes opérations permettant la lecture des messages entre les calculateurs (voir Figure 15 pour la version finale de ces messages) sont un peu détaillées dans le pseudo-code 13. Un message se compose donc :

- du numéro du sommet concerné : u ;
- du numéro du message : # ;
- de la valeur de la composante de μ pour ce sommet : $\mu(u)$;
- de la taille de la liste : k ;
- de la liste des sommets pouvant être affecté à ce sommet.

Algorithme 13 Gestion des communications complexe dans un processus lecteur

```
1: lire(u,entier)
2: if u < 0 then
3:    quitter()
4: end if
5: lire(num, entier)
6: lire(nb_v, entier)
7: lire(μ, double)
8: for cmp de 0 à nb_v do
9:    lire(v, entier)
10:   lire(c̄(v,u), double)
11:   stocker(i,j,c̄(v,u))
12: end for
13: if num = 0 then
14:   destocker(u)
15:   # Appliquer ce que l'on souhaite sur les informations
16: end if
```

Cet algorithme permet de gérer tous les problèmes que l'on peut rencontrer avec des messages trop longs pour la taille d'un pipe en provenance de plusieurs processus écrivant sur le même pipe de manière concurentielle. Les messages sont découpés lors de leur envoi, récupérés par morceaux, stockés tant que le denier morceau n'est pas arrivé. Une fois la dernière partie d'un message est reçue le processus peut effectuer l'ensemble des opérations qu'il doit effectuer comme s'il avait reçu un message unique.

3.3.5 Implémentation

L'implémentation de tous ces algorithmes, et au final la réalisation d'une version parallélisée de l'algorithme du Volume pour le problème du p-médian, modifie en profondeur la structure du programme, mais pas son principe général. La structure de l'algorithme du Volume à été modifiée pour pouvoir le paralléliser. En effet la structure initiale ne permettait pas d'avoir accès simplement à certaines parties du code, en particulier la méthode `solve()`. Nous devions pouvoir ré-écrire entièrement cette méthode. C'est pourquoi nous avons ré-écrit un équivalent à la classe `VOL_problem` (voir Figure 11) au lieu d'utiliser celle fournie. Nous avons donc implémenté la classe `PMED_problem`, qui implémente une méthode `solve()` parallèle.

Les modifications ne se portent donc pas seulement sur la méthode `solve()` mais aussi sur la structure des différents éléments de l'algorithme du Volume. La nouvelle méthode `solve()` inclue en pratique tous les algorithmes décrits dans la Section 3.3.3. Elle va effectuer les tâches suivantes :

- initialiser les mémoires partagées et les variables requises par l'algorithme ;
- lancer tous les processsus fils requis ;
- appliquer l'algorithme du Volume en récupérant les informations des fils.

La technique utilisée pour le lancement des fils étant un simple `fork()`, il a fallut placer l'ensemble du code gérant les différents processus dans cette méthode.

3.3.6 Etude sur le nombre de processus calculateurs

Nous nous sommes demandé quel était l'effet du nombre de processus calculateurs sur le temps de résolution. Nous avons donc réalisé plusieurs tests de vitesse de résolution de la relaxation linéaire de notre problème grâce à l'algorithme du Volume parallélisé. Sur la Figure 16 nous montrons l'évolution de ce temps de résolution en fonction du nombre de processus calculateurs.

La Figure 16 présente deux courbes. Chacune montre l'évolution du temps nécessaire à l'algorithme du Volume pour résoudre la relaxation linéaire d'une instance du p-médian en fonction du nombre de processus calculateurs que nous utilisons. Les deux instances considérées sont les instance $bd01$ et $bd21$ générées aléatoirement, ayant respectivement 2528 et 3999 somets. Ces deux instances font partie des instances générées alétoirement vues à la section 2.3.3.

Dans un premier temps nous remarquons une forte accélération des temps de calculs avec l'augmentation du nombre de processus calculateurs. La courbe se stabilise pour des nombres de processus compris entre 18 et 30, puis croît passé 30 processus. L'accélération initiale est explicable par l'augmentation du

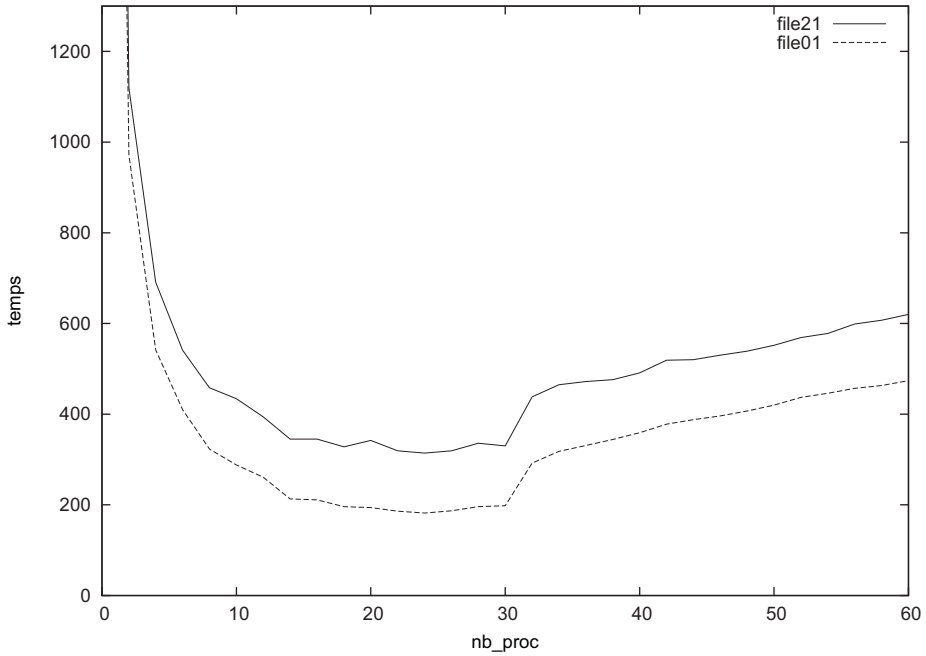


Figure 16: Evolution du temps de résolution de la relaxation linéaire de notre problème en fonction du nombre de processus calculateurs.

nombre de calculs effectuables par seconde avec l'augmentation du nombre de processus. L'augmentation des temps de calculs pour les plus grandes valeurs du nombre de processus s'explique aussi. En effet, le serveur sur lequel nous testons notre programme possède 8 micro-processeurs chacun ayant 4 cœurs, ce qui nous donne 32 processeurs au total. Nous utilisons 4 processus en dehors des calculateurs. Une fois que l'on dépasse, en nombre de processus actifs à un instant donné, le nombre de processeurs, ces processus entrent en concurrence les uns avec les autres. La machine va faire s'exécuter sur un même processeur, plusieurs processus. Ce phénomène entraine des temps de recopie des valeurs internes au processus lorsqu'il y a échange entre les processus actifs. Ces copies entrainent une perte de temps.

Dans le Tableau 8 nous montrons les différentes accélérations obtenues en fonction du nombre de processus calculateurs utilisés. L'accélération est une valeur calculée à partir des temps de résolution de la relaxation linéaire de du p-médian. Le facteur d'accélération est calculé de la manière suivante :

$$\text{acceleration} = \frac{\begin{array}{c}\text{Temps mis pour résoudre le problème avec le code} \\ \text{séquentiel}\end{array}}{\begin{array}{c}\text{Temps mis pour résoudre le problème avec le code} \\ \text{parallèle et } n\_pr \text{ processus calculateur}\end{array}}$$

Nous remarquons ici aussi que le pic d'accélération se trouve entre 24 et 28 processus calculateurs.

	Nombre de processus calculateurs								
	1	2	10	24	28	32	40	50	60
01	0,22	0,75	2,54	4,03	3,74	2,51	2,04	1,74	1,54
21	0,34	1,18	3,06	4,23	3,95	3,03	2,70	2,40	2,14

Tableau 8: Accélérations constatées pour la version parallèle de l'algorithme du Volume pour le problème du p-médian.

3.3.7 Résultats de calcul

Dans cette section nous allons montrer quelques résultats obtenus grâce à l'algorithme du Volume parallèle. Nous ferons une comparaison entre l'algorithme parallèle et l'algorithme séquentiel. Chaque résultat provient de l'utilisation de l'algorithme du Volume suivit de l'application d'une série d'heuristiques rapides pour obtenir une solution entière. L'heuristique utilisée est détaillée à la Section 3.4.2. C'est l'heuristique appelée *Random Rounding* ou d'arrondis aléatoires.

Les tableaux 9 et 10 présente une comparaison entre les versions séquentielles et parallèles de l'algorihtme du Volume pour des instances de petites tailles.

Nous remarquons que les valeurs des bornes inférieures ne sont pas identiques pour chaque version. Ceci est principalement dû au fait que le calcul de la borne supérieure utilisé dans les deux algorithmes n'est pas réalisé de la même manière. Dans la version séquentielle une heuristique est lancée toutes les 50 itérations de l'algorithme. Dans la version parallèle une heuristique est lancée en permanence. Comme la valeur de la borne supérieure est utilisée pour calculer la longueur du pas s à effectuer à chaque itération, le fait de ne pas avoir à chaque itérations la même borne supérieure change les résultats. Cependant ces différences restent minimes.

Il existe une différence entre les solutions entières. Pour tous les résultats la solution entière de la version parallèle est meilleure que la solution entière fournie par la version séquentielle. Cette différence s'explique simplement par la qualité des heuristiques employées. A la sortie de l'algorithme du Volume parallèle la solution entière fournie par l'heuristique Lagrangienne est bien meilleure que dans la version séquentielle. Cette heuristique est utilisé beaucoup plus souvent dans la version parallèle que dans la version séquentielle, d'où la différence de qualité de solution au final.

Nous remarquons enfin que les temps de calculs ne sont pas toujours à l'avantage de la version parallèle de l'algorithme. Les tailles que l'on considère ici ne sont pas assez grandes pour que la version parallèle puisse prendre l'avantage. Cependant nous pouvons noter de nettes améliorations des temps de calculs pour les plus grands problèmes. Il est aussi à noter que le nombre de centres joue un rôle dans le temps de résolution. On peut en effet noter que pour la version parallèle, plus p est grand, et moins il faut de temps pour résoudre le problème. Dans la version séquentielle ce n'est pas le cas. Ceci est dû à la gestion des composantes de μ par le processus Agregateur. En effet, le fait de stocker par défaut la nouvelle composante parmi les p meilleures, puis d'enlever la moins bonne, coûte du temps. Plus p est grand, moins ces opérations sont répétées.

Instances		Volume Séquentiel			Volume Parallèle		
Label	p	LB	UB	T_{RL}	LB	UB	T_{RL}
rl1304	5	3 097 975,53	3 100 978	88	3 097 291,05	3 099 073	295
rl1304	10	2 131 352,75	2 516 345	133	2 131 065,12	2 141 048	404
rl1304	20	1 412 082,22	1 536 190	116	1 412 044,84	1 412 108	309
rl1304	50	794 950,53	836 228	114	794 941,00	795 012	134
rl1304	100	491 447,28	537 398	108	491 446,00	491 664	93
rl1304	200	268 423,79	301 814	107	268 483,00	268 661	59
rl1304	300	177 225,97	196 607	90	177 277,00	177 345	41
rl1304	400	128 269,98	149 586	85	128 238,00	128 336	53
rl1304	500	96 953,53	115 404	85	96 934,20	97 024	44
fl1400	5	174 877,00	174 877	98	174 874,76	174 877	376
fl1400	10	100 572,58	100 601	145	100 585,38	100 601	405
fl1400	20	57 190,19	63 204	159	57 171,98	57 191	365
fl1400	50	28 483,92	29 884	174	28 482,53	28 486	249
fl1400	100	15 942,31	19 424	160	15 950,12	15 970	155
fl1400	200	8 780,72	11 087	128	8 785,71	8 828	76
fl1400	300	6 083,02	9 744	131	6 085,50	6 155	55
fl1400	400	4 631,83	6 613	126	4 630,15	4 686	68
fl1400	500	3 753,04	4 674	101	3 752,64	3 773	43
u1432	5	1 209 793,41	1 210 182	105	1 210 016,66	1 210 126	303
u1432	10	849 758,12	849 759	130	849 737,30	849 759	373
u1432	20	588 612,77	608 622	154	588 649,29	588 875	325
u1432	50	361 562,77	416 916	139	361 316,06	362 499	256
u1432	100	243 675,73	264 542	137	243 667,27	243 866	109
u1432	200	159 717,64	172 973	132	159 786,97	160 707	58
u1432	300	123 612,45	134 031	127	123 581,41	123 771	54
u1432	400	103 304,23	119 343	129	103 353,95	104 389	48
u1432	500	92 940,64	112 325	96	93 027,11	93 200	50
vm1748	5	4 479 357,98	4 479 421	209	4 479 417,42	4 479 421	571
vm1748	10	2 982 171,94	3 523 398	267	2 980 694,17	2 989 031	522
vm1748	20	1 898 282,82	2 537 103	328	1 898 900,81	1 903 021	517
vm1748	50	1 003 523,52	1 248 231	306	1 003 763,52	1 005 297	315
vm1748	100	635 573,24	799 691	280	635 910,89	640 481	261
vm1748	200	389 964,79	426 423	246	390 180,58	390 587	100
vm1748	300	285 852,37	314 925	242	285 760,83	286 416	87
vm1748	400	221 396,82	240 826	233	221 355,21	222 569	92
vm1748	500	176 893,31	196 359	203	176 809,71	177 691	89

Tableau 9: Comparaison entre l'algorithme du Volume séquentiel et parallèle pour des instances de petites tailles. La colonne "Label" indique le nom de l'instance considérée. Le nombre après la série de lettre est la taille de l'instance. La seconde colonne indique le nombre de centres à sélectionner. Les colonnes LB indiquent les valeurs des bornes inférieures trouvées par chaque algorithme. Les colonnes UB présentent les valeures des meilleures solutions entières trouvées. Et les colonnes T_{RL} donnent le temps mis par les algorithmes pour résoudre la relaxation linéaire.

Instances		Volume Séquentiel			Volume Parallèle		
Label	p	LB	UB	T_{RL}	LB	UB	T_{RL}
d2103	5	1 005 112,56	1 005 230	312	1 005 109,39	1 005 139	1 017
d2103	10	687 233,05	792 584	353	687 152,52	687 747	711
d2103	20	482 739,72	554 314	372	482 716,51	482 963	621
d2103	50	301 203,35	384 501	397	301 141,53	303 228	466
d2103	100	194 034,73	247 074	461	194 095,16	195 343	382
d2103	200	117 583,74	160 652	327	117 611,93	117 978	140
d2103	300	90 090,26	121 928	328	90 208,26	90 536	123
d2103	400	75 176,66	99 849	323	75 177,01	76 101	96
d2103	500	63 885,72	83 347	312	63 890,11	64 394	221
pcb3038	5	1 777 479,81	1 866 326	598	1 777 293,71	1 778 273	2 082
pcb3038	10	1 211 703,92	1 211 704	593	1 211 703,82	1 211 704	1 227
pcb3038	20	838 924,69	1 011 814	747	837 783,72	840 027	1 047
pcb3038	50	505 993,96	630 093	731	505 990,03	506 509	634
pcb3038	100	351 244,63	399 592	744	351 161,56	351 681	574
pcb3038	150	279 933,94	310 267	722	279 959,77	280 209	414
pcb3038	200	237 099,72	261 818	596	237 023,77	237 531	326
pcb3038	300	186 623,88	206 033	685	186 698,90	186 896	300
pcb3038	400	156 210,50	162 996	637	156 173,22	156 335	265
pcb3038	500	134 668,08	144 333	580	134 733,44	134 822	282
fl3795	5	1 052 611,97	1 052 683	911	1 052 626,93	1 052 627	3 083
fl3795	10	520 754,53	642 903	1 158	520 934,25	520 940	1 902
fl3795	20	319 686,43	363 948	1 287	319 704,37	319 722	2 089
fl3795	50	150 934,87	195 111	1 386	150 699,13	150 942	1 241
fl3795	100	88 050,43	120 283	1 364	88 115,78	88 319	1 025
fl3795	150	65 756,90	92 125	1 340	65 708,13	65 891	835
fl3795	200	53 767,27	74 804	1 288	53 857,79	53 959	757
fl3795	300	39 546,29	55 787	1 464	39 544,08	39 616	622
fl3795	400	31 319,05	45 100	1 226	31 295,40	31 381	452
fl3795	500	25 957,69	36 867	1 409	25 968,28	26 003	443

Tableau 10: Suite du Tableau 9.

3.4 Heuristiques

Nous avons un outil, l'algorithme du Volume, qui nous permet de résoudre rapidement la relaxation linéaire du problème du p-médian. Il nous faut maintenant pouvoir tranformer cette solution fractionnaire en une solution entière. Il existe plusieurs techniques pour y arriver (ajout de coupes, branch and price, branch and bound, heuristiques) mais elles peuvent devenir lourdes en terme de calculs à effectuer. Nous avons donc utilisé plusieurs heuristiques concurrentes pour arriver à transformer la solution fractionnaire fournie par l'algorithme du Volume en une solution entière.

Nous avons testé cinq heuristiques différentes. Une première heuristique est utilisée lors de la résolution même du problème. Cette première heuristique part de la solution du sous-problème Lagrangien pour arriver à une solution au problème entier. Elle permet donc d'obtenir lors de la résolution une borne

supérieure à notre problème (et donc d'avoir une estimation de la qualité de nos bornes). C'est une heuristique rapide et efficace dans les premières itérations de l'algorithme du Volume. Deux autres heuristiques sont directement appliquées à la solution fractionnaire du problème. La première est une heuristique d'arrondis aléatoires de la solution fractionnaire pour créer une solution entière. La seconde est une variation de la première mais nous ne considérons plus des probabilités pour l'ouverture des centres mais un ordre de priorité. Ces heuristiques sont très rapides et peuvent être appliquées au cours de la résolution de la relaxation linéaire. La première n'est pas efficace dans les premières itérations de l'algorithme du Volume, mais elle devient meilleure que l'heuristique Lagragienne après. La seconde heuristique n'est simplement pas efficace. Notre dernière heuristique est basée sur une heuristique gloutonne. Nous appliquons cette heuristique non pas au problème initial, mais à un problème modifié par la solution fractionnaire, ce qui permet de guider l'heuristique vers d'excellentes solutions. Enfin nous considérons une heuristique de recherche locale basée sur l'échange entre un centre et un sommet non centre.

Ci-dessous nous allons décrire toutes ces heuristiques.

3.4.1 Lagrangienne

Cette heuristique est très simple. La difficulté du problème du p-médian réside dans le choix des centres. Une fois les centres choisis le calcul de la solution optimale pour ces centres est très facile à faire. Supposons que l'on ait un ensemble $J \subset V$ tel que $|J| = p$. J est l'ensemble des centres choisis. Il suffit, pour chaque sommet $u \in V \backslash J$ de relier u au centre v le plus proche. En d'autres termes, si l'on a l'ensemble des centres J et un sommet u n'étant pas un centre, alors la solution la moins coûteuse empruntera l'arc entre u et $v_{min} \in J$ tel que $c(u, v_{min}) \leq c(u, v)\ \forall v \in J$.

Algorithme 14 Heuristique Lagrangienne

Require: $(\bar{x}, \bar{y})$: solution entière de la relaxation Lagrangienne
Ensure: $isol$: vecteur solution de (P)
Ensure: $ival$: valeur de la fonction objectif pour la solution $isol$
Ensure: renvoit **true** si une solution correcte a été trouvée, **false** sinon.

```
ival ← 0.0
ok ← true
for u de 1 à n do
  isol(u) ← ȳ(u)
end for
for u de 1 à n do
  v ← findNearestCenter(u)
  if v < 0 then
    ok ← false
  end if
  ival ← ival + c(u, v)
  isol(u, v) ← 1
end for
return ok
```

L'Algorithme 14 décrit les différentes opérations effectuées par l'heuristique.

Comme dit plus haut la difficulté consiste à choisir les centres, comme ce choix à déjà été réalisé lors de la résolution du sous-problème Lagrangien. Lors de cette résolution, voir l'Algorithme 6, on crée un ensemble de sommets J qui sont les centres. Comme nous n'avons pas relaxé la contrainte $\sum y(u) = p$ nous avons $|J| = p$. Nous utilisons donc cet ensemble pour créer une solution entière réalisable.

3.4.2 Arrondi aléatoire

Etant donnée une solution fractionnaire $(\bar{x}, \bar{y})$ de notre problème nous allons calculer une solution entière en considérant chacune des valeurs comme une probabilité de choix. Ainsi si pour un sommet u on a $y(u) = 1$ alors ce sommet sera forcément un centre dans la solution, si $y(u) = 0.5$ alors il y a 50% de chance que l'on choisisse ce sommet comme centre dans la solution.

Algorithme 15 Heuristique d'arondis aléatoires

Require: $(\bar{x}, \bar{y})$: solution entière de la relaxation Lagrangienne
Ensure: $isol$: vecteur solution de (P)
Ensure: $ival$: valeur de la fonction objectif pour la solution $isol$
1: $ival \leftarrow 0$
2: $cpt \leftarrow 1$
3: $maxIter \leftarrow 100$
4: $ok \leftarrow$ **false**
5: **while** $ok \neq$ **true and** $cpt < maxIter$ **do**
6: $ok \leftarrow$ **true** # on suppose que la solution est correcte
7: $nbOpened \leftarrow 0$
8: **while** $nbOpened < p$ **do**
9: $u \leftarrow rand(1, n)$
10: $r \leftarrow rand()$
11: **if** $\bar{y}(u) > r$ **and** $isol[u]! = 1$ **then**
12: $isol[u] = 1$
13: $nbOpened \leftarrow nbOpened + 1$
14: **end if**
15: **end while**
16: **for** u de 1 à n **do**
17: $v \leftarrow findNearestCenter(u)$
18: **if** $v < 0$ **then**
19: $ok \leftarrow$ **false**
20: **end if**
21: $ival \leftarrow ival + c(u, v)$
22: $isol[u, v] \leftarrow 1$
23: **end for**
24: **end while**

Plusieurs paramètres entrent en compte dans cette heuristique. Le premier est le nombre d'itérations de la boucle while principale. En effet, il est possible que l'heuristique n'arrive pas en un nombre limité d'essais à ouvrir p centres. Dans ce cas on recommence. Nous limitons donc le nombre de fois où l'heuristique se met à la recherche d'une solution. Un second paramètre important est le nombre d'appels à cette heuristique. En effet plus on fait d'appels plus

on a de chance de tomber sur une bonne solution entière, mais plus cela prend de temps. Dans notre implémentation nous avons fixé le nombre maximum de tentative de l'heuristique à 100 et le nombre de réplications à 10.

Cette heuristique peut aussi être appliquée lors de l'exécution de l'algorithme du Volume. Après un certain nombre d'itérations il est possible d'utiliser comme solution d'entrée $(\bar{x}, \bar{y})$ la solution courante fractionnaire et ainsi obtenir une borne supérieure rapidement. Cette heuristique est cependant moins rapide que l'heuristique Lagrangienne.

3.4.3 Respect des centres

Cette heuristique est plus simple à mettre en place que la précédente. L'idée est de récupérer le vecteur $\bar{y}$ fourni par la résolution de la relaxation linéaire du problème. Cette fois nous ne considérons pas la solution fractionnaire comme une probabilité, mais comme une liste de priorité. L'heuristique trie donc les centres potentiels en fonction de la valeur de la solution trouvée, puis affecte les non-centres aux centres ouverts.

Cette heuristique est très rapide. La seule opération coûteuse, hormis la recherche pour tous les non-centres du centre le plus proche, est le tri d'un tableau de taille n. Cependant la qualité des solutions qu'elle fournit est mauvaise. Nous n'utilisons plus cette heuristique.

3.4.4 Construction gloutonne

Dans [38] les auteurs décrivent un problème proche du problème du p-médian. Le problème des lieux d'ouverture de comptes en banque ressemble beaucoup au problème du p-médian, une seule contrainte n'est pas identique car dans leur problème les auteurs souhaitent ouvrir au plus p centres (et au moins un) au lieu de vouloir ouvrir exactement p centres. Une autre différence majeure réside dans la fonction objectif de leur problème qui n'est pas la même que le notre. Le problème des lieux d'ouverture de comptes en banque est plus détaillé à la Section 1.1.2.

L'heuristique gloutonne déjà utilisée dans [73, 96, 69, 38, 105] pour le problème du p-médian est décrite par l'Algorithme 16.

Pour résoudre leur problème les auteurs décrivent une heuristique gloutonne de construction d'une solution.

Cette heuristique permet, en partant d'un graphe $G = (V, A)$ de construire une solution entière au problème du p-médian appliqué à G. Elle fourni des résultats d'excellente qualité en plus d'être assez rapide. Cette version de l'heuristique est celle que nous qualifierons de "classique".

Nous avons légèrement modifié cette heuristique. Nous sommes partis du principe qu'après la résolution de la relaxation linéaire par l'algorihtme du Volume nous avons une solution fractionnaire. Nous modifions donc la fonction coût pour cette heuristique. Au lieu d'utiliser la fonction coût c nous utilisons la fonction coût c' définie par :

$$c'(u, v) = c(u, v) * (1 - \bar{x}(u, v))$$

où $(\bar{x}, \bar{y})$ est la solution de la relaxation linéaire du p-médian. Nous nous sommes donc posé la question de la qualité de cette solution par rapport à la version classique.

Algorithme 16 Heuristique de construction gloutonne

Require: p : nombre de centre à ouvrir.
Require: c : fonction coût associée aux arcs de A
Ensure: $isol$: vecteur solution
Ensure: $ival$: valeur de la fonction objectif pour la solution $isol$
Ensure: en sortie de l'algorithme, $|S| = p$

```
S ← ∅
while |S| < p do
   for all v ∈ V\S do
      α(v) ← Σ_{u∈(V\S)\{v}} min_{qw∈S∪{v}} c(u, w))
      v* ← argmin_{v∈V\S} α(v)
      S ← S ∪ {v*}
   end for
end while
```

Dans les tableaux 11 et 12 nous comparons les valeurs des solutions obtenues pour différentes instances du p-médian par l'heuristique gloutonne classique et notre version modifiée.

Nous avons effectué nos tests sur des instances de petites et moyennes tailles de la TSPLIB pour différentes valeurs de p. Les instances considérées sont *rl1304*, *fl1400*, *u1432*, *vm1748*, *d2103*, *pcb3038* et *fl3795*. Les valeurs de p s'échelonnent de 5 à 500 pour chaque instances. Tous les résultats sauf un montrent que la prise en compte de la solution de la relaxation linéaire améliore la valeur de la solution heuristique. Une seule instance montre un résultat différent : *fl1400* pour $p = 400$.

Notre heuristique permet de construire une solution pour le problème du p-médian à partir de la solution de la relaxation linéaire du problème. Cette heuristique construit une solution en $O(pn^2)$. Les tests menés sur des instances de tailles moyennes ont montré que cette heuristique fonctionne très bien d'un point de vue qualité du résultat.

Amélioration du temps d'exécution de l'heuristique

Pour des instances de petites tailles ou même de tailles moyennes le temps d'exécution de l'heuristique de construction reste faible devant le temps de résolution de la relaxation linéaire. Cependant pour de très grandes tailles de problèmes ce temps de construction peut devenir important.

Nous avons appliqué deux techniques d'accélération des calculs. La première consiste à réduire le champs de recherche de l'heuristique. A chaque itération cette heuristique ouvre un nouveau centre, de manière gloutonne. Originellement ce nouveau centre est choisi parmi l'ensemble des sommets du graphe n'étant pas des centres. Nous réduisons la recherche d'un nouveau centre aux sommets du graphes qui ont le plus de chance de devenir des centres. Ces sommets sont choisis en fonction de la valeur $\bar{y}(u)$ ($(\bar{x}, \bar{y})$ est la solution de la relaxation linéaire). Nous considérons comme centres potentiels les $2p$ sommets, $v_1, \ldots, v_{2p}$, tels que :

$$\bar{y}(v_1) \geq \bar{y}(v_2) \geq \cdots \geq \bar{y}(v_{2p}) \geq \bar{y}(v_{2p+1}) \geq \cdots \geq \bar{y}(v_n)$$

Instances			
Label	p	Sol_{Gr}	$Sol_{Gr_{mod}}$
rl1304	5	3 379 075	3 099 073
rl1304	10	2 245 631	2 150 276
rl1304	20	1 510 069	1 412 108
rl1304	50	864 542	795 043
rl1304	100	522 293	491 724
rl1304	200	288 248	268 724
rl1304	300	190 476	177 609
rl1304	400	138 748	128 547
rl1304	500	104 182	97 049
fl1400	5	349 619	174 877
fl1400	10	105 870	100 601
fl1400	20	63 164	57 191
fl1400	50	30 498	29 385
fl1400	100	16 803	16 012
fl1400	200	9 389	9 360
fl1400	300	6 393	6 260
fl1400	400	4 827	4 876
fl1400	500	3 861	3 773
u1432	5	1 439 810	1 210 126
u1432	10	887 205	849 759
u1432	20	629 405	588 875
u1432	50	380 859	365 679
u1432	100	259 457	246 334
u1432	200	170 940	160 707
u1432	300	130 197	124 558
u1432	400	108 981	104 776
u1432	500	93 323	93 200
vm1748	5	5 012 548	4 479 421
vm1748	10	3 295 535	2 989 031
vm1748	20	2 121 391	1 903 021
vm1748	50	1 118 694	1 005 297
vm1748	100	694 432	640 481
vm1748	200	419 615	390 587
vm1748	300	307 747	286 416
vm1748	400	238 906	222 569
vm1748	500	189 204	177 691

Tableau 11: Comparaison en valeur entre l'heuristique gloutonne "classique" et la version modifiée tenant compte de la solution de la relaxation linéaire sur des instances de petite taille de la TSPLIB. La colonne "Label" indique le nom de l'instance considérée. Le nombre après la série de lettre est la taille de l'instance. La seconde colonne indique le nombre de centres à sélectionner. Les colonnes Sol_{Gr} et $Sol_{Gr_{mod}}$ donnent respectivement la valeur fournie par l'heuristique de construction gloutonne classique et celle fournie par la version modifiée.

Instances			
Label	p	UB_G	UB_{RLG}
d2103	5	1 157 874	1 005 139
d2103	10	709 912	687 747
d2103	20	513 420	483 140
d2103	50	316 267	309 958
d2103	100	203 123	198 440
d2103	200	123 826	119 076
d2103	300	93 492	91 078
d2103	400	77 780	76 469
d2103	500	66 466	64 764
pcb3038	5	1 956 157	1 809 460
pcb3038	10	1 267 956	1 211 704
pcb3038	20	882 063	846 359
pcb3038	50	539 606	513 188
pcb3038	100	368 955	353 543
pcb3038	150	296 041	281 974
pcb3038	200	251 540	238 928
pcb3038	300	198 158	187 856
pcb3038	400	165 203	156 996
pcb3038	500	142 201	135 340
fl3795	5	1 166 588	1 052 627
fl3795	10	552 437	520 940
fl3795	20	345 906	319 722
fl3795	50	159 432	150 946
fl3795	100	95 013	88 982
fl3795	150	69 322	66 479
fl3795	200	56 831	54 416
fl3795	300	41 601	40 435
fl3795	400	33 226	31 610
fl3795	500	27 604	27 693

Tableau 12: Suite du tableau 11.

La seconde technique consiste à réaliser l'implémentation de l'Algorithme 16 en utilisant l'algorithme de la recherche locale [89]. Cette technique d'implémentation est décrite plus en détail à la Section 3.4.5. Supposons que nous ayons à notre disposition un ensemble de sommets, S, de taille $k < p$ déjà sélectionnés. Ajoutons à S, et à G, un sommet particulier, t. t est tel que son affectation à tout autre sommet ait un coût nul, et que l'affectation de tout autre sommet à lui soit impossible. Maintenant cherchons un échange entre un sommet de $S \cup \{t\}$ et un sommet de $V \backslash S$. Il est évident que le sommet sortant sera t, et la perte associée sera de 0. Il suffit alors de trouver le meilleur sommet pour l'échange. Ainsi nous augmentons la taille de S, il suffit à chaque itération de rajouter t à S puis de l'échanger avec un autre sommet du graphe.

En utilisant l'implémentation décrite dans [89] nous obtenons un algorithme rapide pour construire une solution de manière gloutonne. Comme nous utilisons la même implémentation (en partie) que celle de la recherche locale nous

n'avons pas parallélisé cette nouvelle heuristique gloutonne. Le gain réalisé par la parallélisation de la recherche locale a été faible, nous ne nous attendons pas à des gains importants à moins de ne traiter des instances de très très grandes tailles.

Instance			Greedy Améliorée			Greedy Classique			
Label	n	p	Tps	Greedy	LS	Tps	Greedy	LS	Popstar
bd02	2 758	200	43	5 238	5 182	76	5 238	5 181	5 162
bd02	2 758	400	51	3 823	3 770	149	3 804	3 773	3 768
bd02	2 758	800	152	2 562	2 479	285	2 483	2 479	2 479
bd02	2 758	1 000	271	2 083	2 079	311	2 080	2 079	2 079
bd03	2 742	200	12	5 259	5 100	75	5 165	5 097	5 094
bd03	2 742	600	106	2 970	2 950	195	2 962	2 950	2 950
bd03	2 742	800	163	2 494	2 436	281	2 437	2 436	2 436
bd03	2 742	1 000	255	2 043	2 036	348	2 037	2 036	2 037
bd04	2 389	200	12	4 520	4 383	54	4 423	4 384	4 362
bd04	2 389	600	82	2 397	2 385	140	2 386	2 385	2 386
bd04	2 389	800	134	1 968	1 923	201	1 924	1 923	1 923
bd04	2 389	1 000	203	1 530	1 523	246	1 523	1 523	1 523
bd05	2 766	200	51	5 358	5 252	77	5 358	5 251	5 233
bd05	2 766	800	166	2 521	2 478	259	2 480	2 478	2 478
bd05	2 766	1 000	259	2 091	2 078	316	2 078	2 078	2 078
bd06	3 491	200	63	7 744	7 668	113	7 748	7 665	7 628
bd06	3 491	400	73	6 082	6 015	244	6 046	6 017	6 002
bd06	3 491	800	212	4 419	4 319	473	4 341	4 319	4 320
bd06	3 491	1 000	380	3 754	3 720	526	3 730	3 720	3 722
bd21	3 999	500	125	20 580	20 381	401	20 460	20 381	20 360
bd21	3 999	750	240	18 186	18 066	589	18 127	18 068	18 062
bd21	3 999	1 250	727	14 368	14 289	974	14 332	14 288	14 290
bd21	3 999	1 500	851	12 608	12 541	1 213	12 565	12 540	12 541
bd21	3 999	2 000	1 454	9 525	9 494	1 528	9 505	9 494	9 495
bd22	4 000	500	127	32 732	32 616	516	32 698	32 618	32 606
bd22	4 000	1 250	733	23 688	23 641	1 212	23 670	23 641	23 641
bd22	4 000	1 500	860	21 070	20 973	1 400	21 016	20 973	20 973

Tableau 13: Comparaison entre les version modifiée et originelle de l'heuristique de construction gloutonne sur des petites instances. Les trois premières colonnes présentent l'instance (label, nombre de sommets et nombre de centres à sélectionner). Les trois colonnes suivantes présentent les résultats de la version modifiée de l'heuristique (temps, valeur de la solution trouvée, valeur de la solution après recherche locale. Les trois colonnes suivantes présentent les résultats de la version non modifiée. Enfin la dernière colonne donne la valeur de la meilleure solution trouvée par l'heuristique hybride.

Le Tableau 13 présente une comparaison entre la première version améliorée et la version classique de l'heuristique gloutonne et l'heuristique hybride [90]. Popstar est le nom de l'implémentation de l'heuristique hybride. Cette heuristique est la plus performante pour des problèmes de grandes tailles. Ces expériences ont été faites sans résolution de la relaxation linéaire.

Nous observons que la qualité des solutions est détérioré par la modification de l'heuristique. Cependant après avoir effectué une recherche locale à partir de la solution de l'heuristique de construction ces différences s'amenuisent. Sur les 27 instances testées ici 16 fournissent le même résultat. Dans 5 cas la version modifiée fournit de meilleurs résultats et dans 6 cas c'est la version originelle qui fournit le meilleur résultat.

La dernière colonne du Tableau 13 présente la valeur de la solution de l'heuristique hybride. La différence entre la version modifiée et la version originelle de l'heuristique de construction reste faible si nous les comparons à la valeur fournie par l'heuristique hybride. Même si le fait de restreindre le choix des centres à un sous-ensemble des sommets dégrade la solution dans certains cas, cette dégradation ne se retrouve pas forcément dans le résultat final après recherche locale.

Cette modification est intéressante du point de vue des temps de calculs. Dans certains cas elle permet de réduire de beaucoup le temps d'exécution de l'heuristique de construction.

Nous utilisons la résolution de la relaxation linéaire pour obtenir plusieurs informations. Elle nous permet d'avoir une borne inférieure pour tester la qualité des solutions entières. Nous nous servons aussi de la solution fractionnaire obtenue pour guider l'heuristique de construction gloutonne et l'accélérer. Dans le Tableau 14 nous présentons une étude de l'impact de cette modification.

Nous remarquons que toutes les solutions trouvées avec c' sont meilleures que celles trouvées avec c. Cette résolution nous permet de trouver des solutions plus proches de celles fournies par l'heuristique hybride, voir même dans certains cas d'en trouver de meilleures. Le nombre d'itérations de la recherche locale est aussi plus faible, dans la pluspart des cas, après la résolution de la relaxation linéaire.

Même si le temps de résolution de la relaxation linéaire est important, elle est utile. Non seulement elle nous permet d'obtenir une borne inférieure, mais en plus elle améliore la qualité des solutions entières.

3.4.5 Recherche Locale

Pour résoudre le problème du p-médian beaucoup d'auteurs de la litérature font appel à des heuristiques. Une des heuristique les plus utilisées est la recherche locale basée sur un échange entre un sommet non centre et un centre (swap-based local search). Cette recherche locale repose sur un principe simple. Etant donnée une solution du problème, on cherche si un échange peut être profitable. Si tel est la cas, on procède à l'échange, puis on répète cette opération tant que l'on trouve une meilleure solution.

Cette procédure très simple permet de trouver un optimum local de la fonction objectif. Si l'on se situe déjà proche de l'optimum il est possible que l'on trouve une solution optimale rapidement grâce à cette procédure.

Notations

Pour la description de cet algorithme nous allons avoir besoin de quelques notations supplémentaires. Comme toujours nous notons V l'ensemble des sommets du graphe considéré et A l'ensemble des ses arcs. On a toujours $n = |V|$ et

Instance			Valeurs				Nb Iter	
Label	n	p	Gr_{clsq}	LS	UB_{RlGrLs}	Popstar	Heurs	RL
bd21	3 999	500	20 547	20 438	20 381	20 360	57	58
bd21	3 999	750	18 236	18 126	18 066	18 062	74	50
bd21	3 999	1 000	16 236	16 122	16 071	16 064	73	34
bd21	3 999	1 250	14 453	14 345	14 289	14 290	78	26
bd21	3 999	1 500	12 703	12 598	12 541	12 541	81	35
bd21	3 999	2 000	9 632	9 543	9 494	9 495	71	14
bd22	4 000	500	32 693	32 658	32 616	32 606	29	35
bd22	4 000	750	29 481	29 417	29 364	29 356	47	41
bd22	4 000	1 000	26 532	26 457	26 395	26 392	53	35
bd22	4 000	1 250	23 782	23 700	23 641	23 641	61	15
bd22	4 000	1 500	21 115	21 039	20 973	20 973	53	37
bd22	4 000	2 000	16 115	16 022	15 971	15 970	66	5
bd26	5 999	500	26 794	26 653	26 603	26 575	81	86
bd26	5 999	750	24 362	24 236	24 176	24 153	81	69
bd26	5 999	1 000	22 403	22 283	22 177	22 166	84	63
bd26	5 999	1 250	20 653	20 527	20 427	20 416	87	42
bd26	5 999	1 500	19 063	18 964	18 853	18 849	72	72
bd26	5 999	2 000	16 063	15 955	15 858	15 850	87	39
bd27	8 000	750	37 836	37 664	37 596	37 547	102	94
bd27	8 000	1 000	35 391	35 216	35 117	35 068	99	81
bd27	8 000	1 250	33 267	33 065	32 978	32 946	134	94

Tableau 14: Comparaison entre la solution trouvée par application de l'heuristique de construction gloutonne puis d'une recherche locale et les solutions trouvée par notre méthode de résolution ainsi que la solution fournie par l'heuristique hybride[90]. Les trois premières colonnes présentent l'instance (label, nombre de sommets et nombre de centres à sélectionner). Les quatre colonnes suivantes indiquent les valeurs des différentes solutions : la solution de l'heuristique gloutonne classique (Gr_{clsq}), la solution fournie par la recherche locale en partant de la solution Gr_{clsq} (LS), la solution fournie par notre méthode de résolution (UB_{RlGrLs}) et la solution fournie par l'heuristique hybride (Popstar). Dans les deux dernières colonnes se trouvent les nombres respectifs d'itérations pour la recherche locale après l'heuristique gloutonne et après l'heuristique gloutonne modifiée.

$m = |A|$ et p un entier représentant le nombre de centres à selectionner. La relation $1 \leq p \leq n$ est valide, mais nous supposerons pour la suite que $1 < p < n$.

Dans la suite nous utiliserons les notations suivantes : u représente un sommet générique du graphe. Le coût de service de u par v est noté $c(u, v)$ ou c_{uv}. On notera $c(u, J) = min_{v \in J}\{c(u, v)\}$. Une solution, J, est n'importe quel sous-ensemble de V de taille p (une solution peut être représentée simplement par l'ensemble de ces centres). Chaque sommet du graphe doit être relié au centre $v \in J$ le plus proche (celui qui minimise $c(u, v)$). Ce centre sera noté $\phi_1(u)$. De manière similaire on notera $\phi_2(u)$ le second plus proche centre pour le sommet u. Pour faciliter les notations et la lecture nous noterons $c_1(u) = c(u, \phi_1(u))$ et $c_2(u) = c(u, \phi_2(u))$. Il sera beaucoup question de deux sommets particuliers du

graphe, un sommet de $V \backslash J$ candidat pour devenir un centre, f_i (par définition $f_i \notin J$), et un centre candidat pour ne plus en être un, f_r (par définition $f_r \in J$).

La procédure de recherche locale

La procédure classique de recherche locale pour le p-médian à été introduite par Teitz et Bart [99]. Elle est basée sur l'échange entre un centre et un non centre. Pour chaque sommet du graphe $f_i \notin J$ la procédure détermine quel centre $f_r \in J$, s'il en existe un, qui permet la plus grande amélioration de la solution si f_r et f_i devaient être échangés. Si un tel échange améliorant existe, il est alors réalisé et la procédure recommence.

Algorithme 17 localSearch(J)

Require: J : ensemble des centres de la solution initiale
  calculer(J, ϕ_1, ϕ_2)
  $bestprofit \leftarrow 1$
  while $bestprofit > 0$ **do**
    $bestprofit \leftarrow -\infty$
    for $u \in V \backslash J$ **do**
      $(profit, pf_r) \leftarrow$ findOut(J, u, ϕ_1, ϕ_2)
      if $profit > bestprofit$ **then**
        $(bestprofit, f_r, f_i) \leftarrow (profit, pf_r, u)$
      end if
    end for
    if $bestprofit > 0$ **then**
      $J \leftarrow J \cup \{f_i\} \backslash \{f_r\}$
      updateClosest(J, f_i, f_r, ϕ_1, ϕ_2)
    end if
  end while

L'Algorithme 17 décrit les différentes étapes de la recherche locale. Cet algorithme fait appel à 3 fonctions : calculer(J, ϕ_1, ϕ_2), findOut(J, f_i, ϕ_1, ϕ_2) et updateClosest(J, f_i, f_r, ϕ_1, ϕ_2). Chacune de ces trois fonctions réalise une étape importante de l'algorithme.

La fonction calculer(J, ϕ_1, ϕ_2) va simplement calculer les valeurs de ϕ_1 et ϕ_2 pour tous les sommets de V. L'algorithme de cette fonction est très simple, pour chaque sommet il faut parcourir la liste des centres et trouver le plus proche ainsi que le second plus proche. La complexité de cette fonction est $O(pn)$.

La fonction findOut(J, f_i, ϕ_1, ϕ_2) va prendre en entrée un sommet f_i et chercher l'échange le plus profitable avec ce sommet. Elle parcours tous les sommets de J et pour une paire de sommets (f_i, f_r) elle calcul le profit obtenu si l'on remplace f_r par f_i dans la liste des centres. Ce profit peut simplement ête calculé grâce à l'expression suivante :

$$profit(f_i, f_r) = \sum_{u:\phi_1(u) \neq f_r} \max\{0, c_1(u) - c(u, f_i)\} - \sum_{u:\phi_1(u)=f_r} [\min\{c_2(u), c(u, f_i)\} - c_1(u)]$$

La première somme permet de prendre en compte les sommets qui n'étaient pas affectés à f_r. Ils seront affectés à f_i si et seulement si c'est profitable, c'est-à-dire que la distance de u à f_i est plus faible que celle de u à $\phi_1(u)$ et

donc $c_1(u) - c(u, f_i) > 0$. La seconde somme permet de prendre en compte les sommets u qui étaient affectés à f_r et qui doivent être réaffectés soit à f_i soit à $\phi_2(u)$. Cette expression peut facilement être calculée en $O(n)$ pour chaque paire de sommets. Comme il existe $p * (n - p)$ paires possibles la fonction findOut peut être implémentée en $O(pn^2)$.

La dernière fonction est updateClosest(J, f_i, f_r, ϕ_1, ϕ_2). Cette fonction permet de mettre à jour les valeurs de ϕ_1 et ϕ_2 pour l'ensemble des sommets par rapport au nouvel ensemble de centres choisi J. Sans vouloir faire compliqué il suffit de reprendre la fonction calculer et de l'appliquer pour obtenir une mise à jour de ϕ_1 et de ϕ_2. Cette étape peut donc être facilement implémentée en $O(pn)$.

L'ensemble de la procédure de recherche locale peut donc facilement être implémentée en $O(pn^2)$ par itérations de la boucle principale. Cette première version peut être très lente lorsque n augmente et a été rafinée par Withaker [105] puis par Resende et Werneck [89].

Implémentation de Whitaker

Dans [105] Whitaker fournit une nouvelle implémentation de la fonction findOut qui peut être exécutée en $O(n)$. Cette implémentation est présentée dans l'Algorithme 18. Pour simplifier les écriture on notera $I = V \backslash J$.

Algorithme 18 findOut(J, f_i, ϕ_1, ϕ_2)

```
w ← 0
for f ∈ J do
  v(f) ← 0
end for
for u ∈ I do
  if c(u, f_i) < c(u, φ_1(u)) then
    w ← w + c(u, φ_1(u)) − c(u, f_i)
  else
    v(φ_1(u)) ← v(φ_1(u)) + min{c(u, f_i), c(u, φ_2(u))} − c(u, φ_1(u))
  end if
end for
f_r ← argmin_{u∈J}{v(u)}
profit ← w − v(f_r)
return (f_r, profit)
```

Dans l'algorithme w représente ce que l'on économise en réaffectant les sommets à f_i au lieu de $\phi_1(u)$. Ce calcul est indépendant du choix de f_r, il suffit de prendre en compte tous les sommets servis par des centres plus éloignés que f_i. La perte due à la réaffectation des sommets servis par f_r est elle calculée et stockée dans $v(f_r)$ qui tient compte des sommets qui sont affectés à f_r. Le profit généré par l'échange (f_i, f_r) est simple à calculer :

$$profit(f_i, f_r) = w - v(f_r)$$

Il suffit donc de choisir f_r de manière à maximiser le profit.

Cette nouvelle implémentation de la fonction findOut permet de réduire grandement le temps d'exécution de la recherche locale. Comme cette fonc-

tion a une complexité de $O(n)$ il est facile de réaliser une recherche locale en $O(n^2)$ par itération de la boucle principale.

Label	p	T_{LSv1}	T_{LSv2}
rl1304	5	17	1
rl1304	10	33	1
rl1304	20	146	5
rl1304	100	1 506	11
rl1304	200	2 151	12
rl1304	300	2 584	12
rl1304	400	2 432	9
rl1304	500	1 768	7
fl1400	5	10	1
fl1400	10	12	1
fl1400	20	54	3
fl1400	50	177	3
fl1400	100	550	6
fl1400	200	1 393	7
fl1400	300	1 609	7
fl1400	400	754	2
fl1400	500	671	2

Label	p	T_{LSv1}	T_{LSv2}
u1432	5	14	1
u1432	10	61	2
u1432	20	173	3
u1432	100	1 154	12
u1432	200	1 713	9
u1432	300	1 367	6
u1432	400	1 539	5
u1432	500	109	1
vm1748	5	26	2
vm1748	10	61	3
vm1748	20	371	9
vm1748	50	754	14
vm1748	100	2 119	21
vm1748	200	4 575	25
vm1748	300	8 969	37
vm1748	400	10 032	35
vm1748	500	8 367	26

Tableau 15: Comparaison en temps entre l'implémentation naïve et l'implémentation de Whitaker de la recherche locale pour le problème du p-médian. La colonne "Label" indique le nom de l'instance considérée. Le nombre après la série de lettre est la taille de l'instance. La seconde colonne indique le nombre de centres à sélectionner. La colonne T_{LSv1} donne le temps mis par la recherche locale implémentée naïvement. La colonne T_{LSv2} donne le temps mis par la recherche locale implémentée selon [105].

Le Tableau 15 montre plusieurs exécutions de deux recherches locales à partir d'une même solution pour différentes instances du p-médian. Pour chaque instance les procédures de recherche locale ont été exécutées à partir de la même solution initiale. De plus, nous avons choisis dans les deux cas de rechercher à chaque itération la meilleur paire de sommets $(f_i,\ f_r)$ et ensuite de réaliser l'échange.

Implémentation Rapide de la recherche locale

Dans [89] les auteurs donnent une amélioration de l'implémentation de Whitaker. Cette implémentation effectue les mêmes opérations mais elle stocke dans des structures de données des résultats intermédiaires. Ces résultats étant disponibles il suffit de les réutiliser au lieu de les recalculer. L'algorithme 19 présente les différentes étapes de cette recherche locale.

Nous utilisons trois nouvelles structures de données. La première permet de stocker, pour chaque sommet $f_i \notin J$ le gain réalisé si f_i était choisi comme centre, créant ainsi une solution avec $p+1$ centres. Ce gain correspond simplement à ce qui est économisé en affectant un sommet u à f_i plutôt qu'à $\phi_1(u)$. Ce calcul est effectué aussi dans l'algorithme de Whitaker [105] il correspond à w

dans l'Algorithme 18. Le gain peut être calculé comme suit :

$$gain(f_i) = \sum_{u \in V} max\left(0, d_1(u) - d(u, f_i)\right). \tag{3.17}$$

De la même manière est définit, pour chaque sommet $f_r \in J$, la perte si f_r devait ne plus être un centre, créant ainsi une solution avec $p-1$ centres. Cette perte correspond simplement aux réaffectations nécessaires du fait que f_r ne soit plus un centre. Cette perte, $loss(f_r)$, peut être calculée comme suit :

$$loss(f_r) = \sum_{u:\phi_1(u)=f_r} [d_2(u) - d_1(u)]. \tag{3.18}$$

Lors de la recherche locale nous cherchons à déterminer le meilleur couple (f_i, f_r), meilleur au sens où l'échange entre f_i et f_r apportera la plus grande amélioration à la solution. Nous appellons $profit(f_i, f_r)$ l'amélioration obtenue en échangeant f_i et f_r. Cette amélioration peut se calculer de la manière suivante :

$$profit(f_i, f_r) = gain(f_i) - loss(f_r) + extra(f_i, f_r), \tag{3.19}$$

pour une certaine fonction $extra(f_i, f_r)$. Cette fonction doit permettre de corriger les erreurs faites dans les estimations précédentes. Il peut y avoir une erreur dans l'estimation lorsque $\exists u \notin J : \phi_1(u) = f_r$. En effet pour tous les sommets u tels quel $\phi_1(u) \neq f_r$ il ne peut qu'y avoir un gain à l'affecter à f_i, mais ce gain est déjà pris en compte dans $gain(f_i)$. Si $\phi_1(u) = f_r$ alors il existe trois possibilités :

1. $d_1(u) \leq d_2(u) \leq d(u, f_i)$. Ce sommet doit être affecté à $\phi_2(u)$ ce qui est pris en compte dans le calcul de $loss(f_r)$.
2. $d_1(u) \leq d(u, f_i) \leq d_2(u)$. Ce sommet doit être affecté à f_i, mais nous avons considéré qu'il serait affecté à $\phi_2(u)$ dans le calcul de $loss(f_r)$. La contribution de u à $loss(f_r)$ est trop pessimiste de $d_2(u) - d(u, f_i)$.
3. $d(u, f_i) \leq d_1(u) \leq d_2(u)$. Ce sommet doit être affecté à f_i ce qui est déjà pris en compte dans $gain(f_i)$. Cependant la prédiction faite dans $loss(f_r)$ est fausse puisqu'elle tient compte de la réaffectation depuis f_r vers $\phi_2(u)$ qui n'aura pas lieu.

Ainsi nous pouvons calculer la fonction $extra(f_i, f_r)$ grâce à la formule suivante :

$$extra(f_i, f_r) = \sum_{\substack{u:\phi_1(u)=fr \wedge \\ d(u,f_i)<d_2(u)}} [d_2(u) - max\{d(u, f_i), d(u, f_r)\}]. \tag{3.20}$$

Supposons que nous avons à disposition les valeurs de *loss*, *gain* et *extra* dans des structures de données appropriées (des vecteurs pour *loss* et *gain* et une matrice pour *extra*). Alors il est facile de calculer en $O(pn)$ le meilleur couple (f_i, f_r) grâce à l'équation (3.19). Pour calculer efficacement les valeurs de ces structures de données une méthode est donnée par l'Algorithme 20. Ainsi nous pouvons réaliser une implémentation de la procédure de recherche locale en $O(n^2)$. En $O(pn)$ nous calculons ϕ_1 et ϕ_2 pour tous les sommets du graphe. En $O(pn)$, nous mettons à 0 les structures *gain*, *loss* et *extra*. Avec n appel à `updateStructures`, chacun étant réalisé en $O(n)$ nous déterminons leurs valeurs. Enfin en $O(pn)$ nous trouvons le meilleur échange réalisable.

Algorithme 19 rechercheLocale(J, ϕ_1, ϕ_2)

Require: J : ensemble des centres de la solution initiale
Require: ϕ_1 : vecteur des plus proches centres pour chaque sommet $u \in V$
Require: ϕ_2 : vecteur des seconds plus proches centres pour chaque sommet $u \in V$
Ensure: J : ensemble des centres de la nouvelle solution
1: $A \leftarrow V$ # A est l'ensemble des sommets modifiés
2: `resetStructures`($gain, loss, extra$)
3: **while true do**
4: **for all** $u \in A$ **do**
5: `updateStructures`($u, gain, loss, extra, \phi_1, \phi_2$)
6: **end for**
7: $(f_r, f_i, profit) \leftarrow$ `findBestNeighbor`($gain, loss, extra$)
8: **if**($profit \leq 0$) **then break**
9: $A \leftarrow \emptyset$
10: **for all** $u \in V$ **do**
11: **if** $\phi_1(u) = f_r$ **or** $\phi_2(u) = f_r$ **or** $d(u, f_i) < d(u, \phi_2(u)$ **then**
12: $A \leftarrow A \cup \{u\}$
13: **end if**
14: **end for**
15: **for all** $u \in A$ **do**
16: `undoUpdateStructures`($u, gain, loss, extra, \phi_1, \phi_2$)
17: **end for**
18: `insert`(J, f_i)
19: `remove`(J, f_r)
20: `updateClosest`($J, f_i, f_r, \phi_1, \phi_2$)
21: **end while**

Algorithme 20 updateStructures($u, gain, loss, extra, \phi_1, \phi_2$)

1: $d_1 \leftarrow d(u, \phi_1(u))$
2: $d_2 \leftarrow d(u, \phi_2(u))$
3: $f_r \leftarrow \phi_1(u)$
4: $loss(f_r) \leftarrow loss(f_r) + (d_2 - d_1)$
5: **for all** $f_i \notin J$ **do**
6: **if** $d(u, f_i) < d_2$ **then**
7: $gain(f_i) \leftarrow gain(f_i) + \max(0, d_1 - d(u, f_i))$
8: $extra(f_i) \leftarrow extra(f_i) + (d_2 - \max(d(u, f_i), d(u, f_r)))$
9: **end if**
10: **end for**

Parallélisation de la recherche locale

Nous nous sommes intéressé aussi à la parallélisation de la recherche locale, de manière à réduire encore les temps de calculs. Dans l'Algorithme 19 nous avons repéré trois portions de code pouvant être parallélisées. Les fonctions `updateStructures` et `undoUpdateStructures` semblent être indiquées pour être mise en parallèle. Il en est de même pour la fonction `findBestNeighbor` qui va, pour chaque sommet de J, trouver le meilleur sommet de $V \backslash J$ puis chercher le

meilleur couple (f_i, f_r).

La parallélisation des fonctions `updateStructures` et `undoUpdateStructures` ne peut être mise en place simplement. Supposons que l'on réorganise ces calculs de manière à ce que chaque sommet soit traité par un processus. Chaque processus va, pour un certain nombre de f_i (ligne 5 de l'Algorithme 20), mettre à jour les valeurs de $gain(f_i)$ et $extra(f_i)$ (lignes 7 et 8 de l'Algorithme 20). La technique de parallélisation que nous utilisons assure que deux processus ne pourront pas écrire ces donnée en même temps. Cependant rien ne garanti que deux processus ne pourront lire les données en même temps ce qui peut fausser les calculs.

Ces erreurs de calcul sont pourtant évitables. Une technique simple consiste à placer pour chacune des cases de ces tableaux un booléen qui indiquerait s'il est possible de lire la valeur de cette case. Ainsi nous pourrions nous assurer que tous les calculs soient exacts. Cependant le surcoût de cette solution est tel que nous perdrions tout l'intérêt de la parallélisation. Nous n'avons donc pas modifié ces fonctions.

Nous avons parallélisé la fonction `findBestNeighbor` qui pouvait l'être sans difficultés. Nous avons mis en place trois types de processus différents : un processus maître, un ensemble de processus calculateurs et un processus agrégateur.

Le processus maître coordonne les différents processus et implémente la majeure partie de l'Algorithme 19. Il se charge du lancement des autres processus ainsi que de leur bonne terminaison. Les processus calculateurs reçoivent un numéro de sommet, $f_i \in V \backslash J$. Pour ce sommet ils cherchent le meilleur sommet $f_r \in J$ pour faire un échange. Cette information, f_i, f_r, et le profit alors réalisé, sont envoyés au processus agrégateur. Ce processus récupère l'ensemble des couples (f_i, f_r) potentiels et sélectionne celui ayant le plus grand profit. Une fois le choix effectué, il en informe le processus maître qui peut alors continuer les calculs.

Le Tableau 16 présente une comparaison entre trois implémentations de la recherche locale. Les trois versions fournissent le même résultat. La version décrite dans [89] est bien plus efficace que l'algorithme de [105]. La parallélisation de cet algorithme, déjà efficace, n'améliore ses performances que dans peu de cas.

Nous nous sommes aussi intéressé à l'impact global de nos parallélisations. Pour ce faire nous avons lancé une résolution entièrement séquentielle d'une instance et une résolution entièrement parallèle. Nous avons choisi l'instance *bd*30 qui a 20000 sommets. Sur ces 20000 sommets nous souhaitons sélectionner 2000 centres. La résolution de la relaxation linéaire a duré 70753 secondes soit environ 19 heures et 40 minutes. L'heuristique de construction gloutonne a duré 220810 secondes soit environ 2 jours et 14 heures. Enfin la recherche locale implémentée selon [105] a durée 58713 secondes soit environ 16 heures et 20 minutes. Le temps d'éxécution pour la version parallèle est de 15005 secondes pour la résolution de la relaxation linéaire soit environ 4 heures et 10 minutes. L'heuristique gloutonne parallèle a duré 4713 secondes soit environ 1 heure et 20 minutes. Enfin la recherche locale implémentée selon [89] et de manière parallèle à mis 273 secondes (la même recherche locale implémentée de manière séquentielle a mis 468 secondes). Au total la version séquentielle de notre méthode de résolution a duré environ 81 heures et 10 minutes contre 5 heures et 35 minutes environ pour la version parallèle (en ne considérant que les implémentations rapides [89] pour la recherche locale).

Instance			Valeurs		Temps			
Label	n	p	RL	UB	RL	Whit	Res	Res //
bd21	3 999	500	20 265,83	20 379	423	90	4	10
bd21	3 999	750	18 025,16	18 066	412	85	4	10
bd21	3 999	1 000	16 035,88	16 068	390	48	4	7
bd21	3 999	1 250	14 270,23	14 289	389	41	3	6
bd21	3 999	1 500	12 526,66	12 541	393	40	3	6
bd21	3 999	2 000	9 480,35	9 494	354	15	1	6
bd22	4 000	500	32 538,83	32 615	405	67	4	8
bd22	4 000	750	29 299,56	29 365	449	70	4	8
bd22	4 000	1 000	26 359,90	26 394	409	53	3	8
bd22	4 000	1 250	23 608,11	23 642	367	22	2	6
bd22	4 000	1 500	20 922,96	20 973	425	42	3	6
bd22	4 000	2 000	15 954,62	15 971	490	5	1	5
bd26	5 999	500	26 391,36	26 602	969	367	13	21
bd26	5 999	750	24 029,01	24 183	1 045	474	13	18
bd26	5 999	1 000	22 131,47	22 176	982	296	26	20
bd26	5 999	1 250	20 360,50	20 425	1 005	156	9	15
bd26	5 999	1 500	18 807,30	18 853	997	356	18	21
bd26	5 999	2 000	15 795,41	15 859	1 002	197	11	16
bd27	8 000	500	40 241,23	40 702	2 538	1 159	14	39
bd27	8 000	750	37 322,87	37 593	2 236	1 896	23	30
bd27	8 000	1 000	34 939,24	35 116	1 980	1 131	30	28
bd27	8 000	1 250	32 835,64	32 977	2 144	3 416	41	39

Tableau 16: Comparaison entre trois versions de la recherche locale pour le problème du p-médian. Les trois premières colonnes présentent l'instance (label, nombre de sommets et nombre de centres à sélectionner). Les deux colonnes suivantes donnent respectivement la valeur des meilleures bornes inférieure et supérieure trouvées. Dans les trois cas la meilleure borne supérieure est identique. Les quatre dernières colonnes fournissent différents temps d'exécution : le temps mis pour résoudre la relaxation linéaire, le temps mis par l'algorithme de Whitaker [105], le temps mis par l'algorithme décrit dans [89] et le temps mis par ce même algorithme parallélisé.

Utilisation de la recherche locale

Nous avons maintenant une procédure de recherche locale. Pour pouvoir l'utiliser il faut une solution entière à notre problème. Naturellement nous utilisons la solution fournie par l'heuristique gloutonne pour le p-médian décrite à la section 3.4.4. Le point de départ de notre recherche locale est important car plus il est proche de l'optimum, meilleure sera la qualité de la solution de la recherche locale, et moins cette recherche fera d'itérations. Nous nous sommes donc intéressé aux résultats de la recherche locale couplée avec l'heuristique gloutonne de construction.

Les tableaux 17 et 18 montrent les résultats de l'exécution sur plusieurs instances de l'heuristique de construction gloutonne puis d'une recherche locale. Pour chaque instance, nous appliquons dans un premier temps l'heuristique de

Instance			
Label	p	S_{gLS}	S_{gRLLS}
rl1304	5	3 099 073	3 099 073
rl1304	10	2 135 581	2 141 048
rl1304	20	1 412 928	1 412 108
rl1304	50	799 482	795 012
rl1304	100	493 870	491 664
rl1304	200	269 919	268 661
rl1304	300	178 942	177 345
rl1304	400	129 982	128 336
rl1304	500	98 771	97 024
fl1400	5	174 877	174 877
fl1400	10	100 601	100 601
fl1400	20	57 238	57 191
fl1400	50	29 177	28 486
fl1400	100	16 011	15 970
fl1400	200	8 967	8 828
fl1400	300	6 216	6 155
fl1400	400	4 766	4 686
fl1400	500	3 823	3 773
u1432	5	1 210 126	1 210 126
u1432	10	857 929	849 759
u1432	20	593 195	588 875
u1432	50	366 303	362 499
u1432	100	245 539	243 866
u1432	200	162 612	160 707
u1432	300	126 914	123 771
u1432	400	106 439	104 389
u1432	500	93 200	93 200
vm1748	5	4 479 421	4 479 421
vm1748	10	2 985 928	2 989 031
vm1748	20	1 902 241	1 903 021
vm1748	50	1 008 973	1 005 297
vm1748	100	640 421	640 481

Tableau 17: Comparaison entre les deux versions de l'heuristique de construction gloutonne suivie d'une procédure de recherche locale. La colonne "Label" indique le nom de l'instance considérée. Le nombre après la série de lettre est la taille de l'instance. La seconde colonne indique le nombre de centres à sélectionner. La colonne S_{gLS} donne la valeur de la solution obtenue après application de l'heuristique de construction "classique" et une recherche locale. La dernière colonne, S_{gRLLS}, donne la valeur de la solution obtenue après résolution de la relaxation linéaire, application de l'heuristique gloutonne et de la recherche locale.

construction pour trouver une solution S_g. Puis nous effectuons une recherche locale à partir de cette solution. Dans un second temps nous appliquons l'algorithme du Volume pour résoudre la relaxation linéaire du problème. Nous ap-

Instance			
Label	p	S_{gLS}	S_{gRLLS}
vm1748	200	395 509	390 587
vm1748	300	288 259	286 416
vm1748	400	222 913	222 569
vm1748	500	178 481	177 691
d2103	5	1 005 370	1 005 139
d2103	10	691 183	687 747
d2103	20	483 086	482 963
d2103	50	304 291	303 228
d2103	100	195 922	195 343
d2103	200	119 702	117 978
d2103	300	91 836	90 536
d2103	400	76 178	76 101
d2103	500	65 526	64 394
pcb3038	5	1 779 506	1 778 273
pcb3038	10	1 211 704	1 211 704
pcb3038	20	845 137	840 027
pcb3038	50	508 521	506 509
pcb3038	100	357 461	351 681
pcb3038	150	282 619	280 209
pcb3038	200	240 502	237 531
pcb3038	300	188 480	186 896
pcb3038	400	157 895	156 335
pcb3038	500	136 252	134 822
fl3795	5	1 053 571	1 052 627
fl3795	10	525 202	520 940
fl3795	20	327 202	319 722
fl3795	50	152 194	150 942
fl3795	100	89 267	88 319
fl3795	150	66 509	65 891
fl3795	200	54 869	53 959
fl3795	300	39 949	39 616
fl3795	400	31 882	31 381
fl3795	500	26 548	26 003

Tableau 18: Suite du Tableau 17

pliquons ensuite l'heuristique gloutonne à partir de cette solution fractionnaire pour obtenir une solution entière S_{gRL}. Enfin nous effectuons une recherche locale à partir de cette solution.

Les résultats sont tous en faveur de la version modifiée de l'heuristique. Il arrive que les deux versions trouvent la solution optimale du problème, mais c'est toujours pour des valeurs de p petites ($p = 5$ et $p = 10$). Pour de telles valeurs de p la procédure de recherche locale est très efficace et permet aux deux versions d'arriver à l'optimum. Pour des valeurs de p plus importantes l'heuristique améliorée fournit une meilleure solution de base. Ainsi, après une recherche locale les solutions fournies par l'heuristique améliorée sont de meilleur

qualité.

Les tableaux 19 et 20 présentent une comparaison entre l'algorithme du Volume parallèle et les heuristiques décrites dans [90, 5]. Les comparaisons sont faites sur des instances de la TSPLIB. Nous remarquons que l'algorithme du Volume fournit des solutions comparables aux solutions trouvées par les deux heuristiques. Les valeurs des bornes inférieures données par Avella sont meilleures que celles données par l'algorithme du Volume. Ceci s'explique par le fait que l'algorihtme du Volume ne fournit pas une solution primale optimale, mais une estimation de cette solution.

La meilleure heuristique pour des problèmes de tailles raisonnables est celle décrite dans [5]. Le principe de cette heuristique est simple : résoudre la relaxation linéaire du problème, supprimer les variables nulles. Ensuite utiliser CPLEX pour résoudre ce problème réduit, en variables entières. Dans [5] les auteurs rapportent des résultats allant jusqu'à 6000 sommets environ. Cette technique est extrêmement gourmande en mémoire et en temps, à cause de l'utilisation d'un algorithme de coupes et de branchements. Cependant, cette heuristique produit de très bons résultats

L'heuristique Hybride de Resende fournit d'excellentes solutions pour le problème du p-médian. Cependant nous en obtenons toujours de très proches et parfois même de meilleures. Et nous pouvons résoudre des instances que l'heuristique hybride ne peut pas résoudre par manque de mémoire.

3.5 Résultats de calculs

Dans cette section nous présentons plusieurs résultats de l'application de notre méthode de résolution. Chaque résultat est obtenu par l'application de différentes étapes. Nous résolvons dans un premier temps la relaxation linéaire de notre problème grâce à l'algorithme du Volume. C'est la version parallèle de cet algorithme qui est utilisée dans ces tests. Ensuite nous appliquons l'heuristique de construction gloutonne. Cette heuristique utilise la solution de la relaxation linéaire pour construire la solution entière. Enfin nous appliquons une procédure de recherche locale pour améliorer encore cette solution. La procédure de recherche locale est implémentée suivant l'algorithme de Withaker comme décrit à la Section 3.4.5.

Les tableaux 21, 22 et 23 présentent des résolutions du problème du p-médian pour des instances de petites, moyennes et grandes tailles. Ces trois tableaux se présentent de manière quasi identique.

Mais les tableaux 21 et 22 contiennent trois colonnes supplémentaires. Nous souhaitons toujours comparer nos résultats à ceux de l'heuristique hybride de Resende et nous donnons donc ses résultats sur ces instances. Le Gap est calculé de manière à fournir toujours une valeur comparable et positive. Parmi les deux solutions, l'une est toujours supérieure (ou égale) à l'autre. Nous calculons notre Gap comme étant le rapport entre la différence entre les solutions et la plus grande des deux. Si l'on nomme BS la plus grande des deux solutions et LS la plus petite. Alors le Gap est calculé comme suit :

$$Gap = \frac{BS - LS}{BS}$$

Le Tableau 23 ne contient pas ces colonnes car l'heuristique hybride n'arrive pas à gérer ces tailles de problèmes. Aux alentours de 15000 sommets les besoins

en mémoire de cette heuristique ne peuvent être satisfaits par l'ordinateur sur lequel nous avons effectué ces tests. Nous avons cependant ajouté une colonne Gap au Tableau 23. Cette colonne représente la différence relative entre notre borne inférieure et notre borne supérieure et est calculée de la manière suivante :

$$Gap = \frac{UB - LB}{UB}$$

Pour tous ces résultats nous pouvons remarquer que les valeurs des Gap sont faibles, qu'il sagisse du Gap entre la solution de l'heuristique hybride et la notre ou du Gap entre notre borne supérieure et notre borne inférieure. Pour aucune des solutions comparées avec la solution de l'heuristique hybride le Gap ne monte au dessus de $0,5\%$. En termes de temps de calculs pour des instances de petites tailles l'heuristique hybride est plus rapide que notre méthode. Cependant pour des instances de tailles moyennes les durées sont comparables, et pour des instances de grandes tailles ces temps seront certainement à l'avantage de notre méthode. En effet, le principe même de l'heuristique hybride est d'effectuer de très nombreuse recherche locale ainsi que de nombreux chemins reliants alors que nous n'effectuons qu'une recherche locale et qu'une seule construction d'une solution après avori résolu la relaxation linéaire.

Instances		Resende	Avella		Volume + Greedy + LS		
Label	p	$UB_{Popstar}$	LB_{Avella}	UB_{avella}	LB_{Volume}	UB_{Volume}	Gap
rl1304	5	**3 099 073**	3 099 073	**3 099 073**	3 097 291,05	**3 099 073**	0,00 %
rl1304	10	2 134 295	2 131 788	2 134 295	2 131 065,12	2 141 048	0,32 %
rl1304	20	**1 412 108**	1 412 108	**1 412 108**	1 412 044,84	**1 412 108**	0,00 %
rl1304	50	**795 012**	795 012	**795 012**	794 941,00	**795 012**	0,00 %
rl1304	100	491 899	491 507	491 639	491 446,00	491 664	0,01 %
rl1304	200	268 737	268 573	**268 573**	268 483,00	268 661	0,03 %
rl1304	300	177 334	177 318	177 326	177 277,00	177 345	0,01 %
rl1304	400	128 338	128 332	**128 332**	128 238,00	128 336	0,00 %
rl1304	500	97 048	97 018	97 024	96 934,20	97 024	0,01 %
fl1400	5	**174 877**	174 877	**174 877**	174 874,76	**174 877**	0,00 %
fl1400	10	**100 601**	100 601	**100 601**	100 585,38	**100 601**	0,00 %
fl1400	20	**57 191**	**57 191**	57 191	57 171,98	**57 191**	0,00 %
fl1400	50	**28 486**	28 486	**28 486**	28 482,53	**28 486**	0,00 %
fl1400	100	15 970	15 954	15 962	15 950,12	15 970	0,05 %
fl1400	200	8 816	8 789	8 806	8 785,71	8 828	0,25 %
fl1400	300	6 131	6 070	6 111	6 085,50	6 155	0,71 %
fl1400	400	4 655	4 635	4 648	4 630,15	4 686	0,81 %
fl1400	500	3 765	3 755	3 765	3 752,64	3 773	0,21 %
u1432	5	**1 210 126**	1 210 126	**1 210 126**	1 210 016,66	**1 210 126**	0,00 %
u1432	10	**849 759**	849 759	**849 759**	849 737,30	**849 759**	0,00 %
u1432	20	588 766	588 720	588 766	588 649,29	588 875	0,02 %
u1432	50	362 089	361 699	362 089	361 316,06	362 499	0,11 %
u1432	100	243 827	243 756	243 831	243 667,27	243 866	0,01 %
u1432	200	159 929	159 852	159 887	159 786,97	160 707	0,51 %
u1432	300	13 835	123 663	123 689	123 581,41	123 771	0,07 %
u1432	400	104 102	103 401	103 979	103 353,95	104 389	0,39 %
u1432	500	93 200	93 199	93 200	93 027,11	93 200	0,00 %
vm1748	5	**4 479 421**	4 479 421	**4 479 421**	4 479 417,42	**4 479 421**	0,00 %
vm1748	10	2 983 645	2 982 964	2 983 645	2 980 694,17	2 989 031	0,18 %
vm1748	20	1 899 680	1 899 339	1 899 680	1 898 900,81	1 903 021	0,18 %
vm1748	50	1 004 337	1 004 324	1 004 331	1 003 763,52	1 005 297	0,10 %
vm1748	100	636 569	636 401	636 515	635 910,89	640 481	0,62 %
vm1748	200	390 353	390 350	**390 350**	390 180,58	390 587	0,06 %
vm1748	300	286 151	286 035	286 039	285 760,83	286 416	0,13 %
vm1748	400	221 562	221 523	221 526	221 355,21	222 569	0,47 %
vm1748	500	177 037	176 977	176 995	176 809,71	177 691	0,39 %

Tableau 19: Comparaison en valeur entre l'heuristique hybride [90], l'heuristique décrite dans [5] et l'algorithme du Volume parallèle suivit de l'heuristique gloutonne puis d'une recherche locale. La colonne "Label" indique le nom de l'instance considérée. Le nombre après la série de lettre est la taille de l'instance. La seconde colonne indique le nombre de centres à sélectionner. La colonne $UB_{popstar}$ présente la valeur de la meilleure solution fournie par l'heuristique hybride. Les colonnes LB_{Avella} et UB_{Avella} présentent respectivement la borne inférieure et la borne supérieure trouvée par l'heuristique décrite dans [5]. Les colonnes LB_{Volume} et UB_{Volume} présente la même chose pour l'algorithme du Volume. Enfin la colone Gap présente la différence relative entre UB_{Avella} et UB_{Volume}. Les valeurs en gras sont les valeur optimales. Un Gap négatif signifie que notre algorithme a trouvé une solution meilleure que celle trouvée par l'heuristique décrite dans [5].

Instances		Resende	Avella		Volume + Greedy + LS		
Label	p	$UB_{Popstar}$	LB_{Avella}	UB_{avella}	LB_{Volume}	UB_{Volume}	Gap
d2103	5	**1 005 136**	1 005 136	**1 005 136**	1 005 109,39	1 005 139	0,00 %
d2103	10	687 321	687 258	687 321	687 152,52	687 747	0,06 %
d2103	20	482 926	482 790	482 926	482 716,51	482 963	0,01 %
d2103	50	302 337	301 540	302 384	301 141,53	303 228	0,28 %
d2103	100	194 920	194 362	195 012	194 095,16	195 343	0,17 %
d2103	200	117 803	117 733	117 753	117 611,93	117 978	0,19 %
d2103	300	90 472	90 415	90 471	90 208,26	90 536	0,07 %
d2103	400	75 422	75 289	75 356	75 177,01	76 101	0,98 %
d2103	500	64 231	63 942	64 006	63 890,11	64 394	0,60 %
pcb3038	5	1 777 835	1 777 648	1 777 902	1 777 293,71	1 778 273	0,02 %
pcb3038	10	**1 211 704**	1 211 704	**1 211 704**	1 211 703,82	**1 211 704**	0,00 %
pcb3038	20	839 494	839 173	839 635	837 783,72	840 027	0,05 %
pcb3038	50	506 464	506 188	506 339	505 990,03	506 509	0,03 %
pcb3038	100	351 967	351 377	351 553	351 161,57	351 681	0,04 %
pcb3038	150	280 233	280 039	280 164	279 959,77	280 209	0,02 %
pcb3038	200	237 584	237 316	237 441	237 023,77	237 531	0,04 %
pcb3038	300	186 994	186 789	186 840	186 698,90	186 896	0,03 %
pcb3038	400	156 300	156 268	156 285	156 173,22	156 335	0,03 %
pcb3038	500	134 864	134 774	134 810	134 733,44	134 822	0,01 %
fl3795	5	**1 052 627**	1 052 627	**1 052 627**	1 052 626,93	**1 052 627**	0,00 %
fl3795	10	**520 940**	520 940	**520 940**	520 934,25	**520 940**	0,00 %
fl3795	20	**319 722**	319 722	**319 722**	319 704,37	**319 722**	0,00 %
fl3795	50	**150 940**	150 940	**150 940**	150 699,13	150 942	0,00 %
fl3795	100	88 378	88 299	**88 299**	88 115,78	88 319	0,02 %
fl3795	150	65 942	65 831	65 868	65 708,13	65 891	0,03 %
fl3795	200	53 959	53 904	53 928	53 857,79	53 959	0,06 %
fl3795	300	39 669	39 571	39 586	39 544,08	39 616	0,08 %
fl3795	400	31 463	31 340	31 354	31 295,40	31 381	0,09 %
fl3795	500	26 076	25 972	25 976	25 968,28	26 003	0,10 %
rl5934	10	9 792 218	9 786 340	9 795 785	9 771 081,35	9 792 956	-0,03 %
rl5934	20	6 716 215	6 712 891	6 718 043	6 710 264,27	6 716 478	-0,02 %
rl5934	50	4 030 302	4 026 613	4 030 518	4 022 492,61	4 032 050	0,04 %
rl5934	100	2 723 465	2 720 293	2 723 749	2 717 380,05	2 723 171	-0,02 %
rl5934	150	2 146 107	2 144 783	2 146 797	2 141 954,85	2 146 217	-0,03 %
rl5934	300	1 392 874	1 392 282	1 392 419	1 391 403,64	1 392 762	0,02 %
rl5934	400	1 145 300	1 143 632	1 143 962	1 142 950,97	1 144 229	0,02 %
rl5934	500	973 098	972 739	972 799	972 265,09	973 029	0,02 %
rl5934	600	847 725	847 229	847 301	846 952,67	847 340	0,01 %
rl5934	700	751 560	751 054	751 131	750 629,75	751 206	0,01 %
rl5934	800	676 218	675 882	675 963	675 538,66	676 114	0,02 %
rl5934	900	612 948	612 574	612 629	612 305,25	612 661	0,01 %
rl5934	1 100	511 357	511 137	511 247	510 916,30	511 256	0,01 %
rl5934	1 200	469 885	469 711	469 775	469 553,47	469 749	-0,01 %
rl5934	1 300	433 270	433 014	433 060	432 754,73	433 068	0,01 %
rl5934	1 400	401 450	401 355	401 370	401 160,92	401 388	0,01 %
rl5934	1 500	373 634	373 566	373 566	373 373,81	374 613	0,28 %

Tableau 20: Suite du Tableau 19.

Instance			Volume & Gr_{mod} & LS			Popstar			
Label	n	p	LB	UB	Tps	UB_{pop}	Tps_{pop}	Gap1	Gap2
bd04	2 389	200	4 328,24	4 378	275	4 362	54,50	1,13 %	0,77 %
bd04	2 389	400	3 083,03	3 099	172	3 094	56,23	0,51 %	0,35 %
bd04	2 389	600	2 380,78	2 385	169	2 386	48,14	0,17 %	0,21 %
bd04	2 389	800	1 909,46	1 923	178	1 923	49,34	0,70 %	0,70 %
bd04	2 389	1 000	1 518,85	1 523	189	1 523	64,26	0,27 %	0,27 %
bd03	2 742	200	5 016,39	5 097	314	5 094	56,91	1,58 %	1,52 %
bd03	2 742	400	3 715,93	3 744	307	3 734	87,25	0,74 %	0,48 %
bd03	2 742	600	2 938,17	2 950	263	2 950	84,42	0,40 %	0,40 %
bd03	2 742	800	2 429,17	2 436	285	2 436	97,79	0,28 %	0,28 %
bd03	2 742	1 000	2 031,38	2 036	271	2 037	100,33	0,22 %	0,27 %
bd02	2 758	200	5 097,73	5 181	301	5 162	80,83	1,60 %	1,24 %
bd02	2 758	400	3 749,13	3 773	387	3 768	86,01	0,63 %	0,50 %
bd02	2 758	600	2 999,03	3 009	288	3 009	111,48	0,33 %	0,33 %
bd02	2 758	800	2 466,82	2 479	281	2 479	87,37	0,49 %	0,49 %
bd02	2 758	1 000	2 068,02	2 079	309	2 079	98,57	0,52 %	0,52 %
bd05	2 766	200	5 170,23	5 251	316	5 233	90,50	1,53 %	1,19 %
bd05	2 766	400	3 814,45	3 848	288	3 835	97,55	0,87 %	0,53 %
bd05	2 766	600	3 022,41	3 031	303	3 034	99,52	0,28 %	0,38 %
bd05	2 766	800	2 468,81	2 478	289	2 478	82,26	0,37 %	0,37 %
bd05	2 766	1 000	2 071,65	2 078	269	2 078	86,20	0,30 %	0,30 %
bd06	3 491	200	7 528,10	7 660	506	7 628	144,88	1,72 %	1,30 %
bd06	3 491	400	5 974,44	6 013	440	6 002	142,12	0,64 %	0,45 %
bd06	3 491	600	5 023,97	5 047	347	5 038	149,58	0,45 %	0,27 %
bd06	3 491	800	4 305,04	4 319	332	4 320	207,43	0,32 %	0,34 %
bd06	3 491	1 000	3 712,96	3 720	309	3 722	180,80	0,18 %	0,24 %
bd21	3 999	500	20 265,83	20 381	443	20 360	176,41	0,56 %	0,46 %
bd21	3 999	750	18 025,16	18 068	423	18 062	223,89	0,23 %	0,20 %
bd21	3 999	1 000	16 035,88	16 079	413	16 064	234,44	0,26 %	0,17 %
bd21	3 999	1 250	14 270,23	14 288	378	14 290	233,96	0,12 %	0,13 %
bd21	3 999	1 500	12 526,66	12 540	374	12 541	320,84	0,10 %	0,11 %
bd21	3 999	2 000	9 480,35	9 494	369	9 495	209,91	0,14 %	0,15 %
bd22	4 000	500	32 538,83	32 618	432	32 606	160,06	0,24 %	0,20 %
bd22	4 000	750	29 299,56	29 366	436	29 356	176,29	0,22 %	0,19 %
bd22	4 000	1 000	26 359,90	26 407	416	26 392	207,05	0,17 %	0,12 %
bd22	4 000	1 250	23 608,11	23 641	368	23 641	215,86	0,13 %	0,13 %
bd22	4 000	1 500	20 922,96	20 973	390	20 973	260,55	0,23 %	0,23 %
bd22	4 000	2 000	15 954,62	15 974	453	15 970	236,36	0,12 %	0,09 %
bd24	4 000	500	50 139,92	50 252	405	50 244	171,80	0,22 %	0,20 %
bd24	4 000	750	45 554,15	45 618	391	45 606	258,94	0,13 %	0,11 %
bd24	4 000	1 000	41 275,36	41 306	355	41 303	264,95	0,07 %	0,06 %
bd24	4 000	1 250	37 181,16	37 232	351	37 228	312,03	0,13 %	0,12 %
bd24	4 000	1 500	33 195,24	33 240	342	33 235	347,14	0,13 %	0,11 %
bd24	4 000	2 000	25 679,29	25 720	392	25 720	285,92	0,15 %	0,15 %
bd25	4 000	500	74 139,04	74 284	389	74 271	193,62	0,19 %	0,17 %
bd25	4 000	750	67 749,78	67 813	378	67 811	147,88	0,09 %	0,09 %
bd25	4 000	1 000	61 656,96	61 723	335	61 721	185,11	0,10 %	0,10 %
bd25	4 000	1 250	55 807,02	55 841	358	55 839	246,91	0,06 %	0,05 %
bd25	4 000	1 500	50 013,74	50 095	382	50 089	225,40	0,16 %	0,15 %
bd25	4 000	2 000	39 060,49	39 071	346	39 072	169,83	0,02 %	0,02 %

Tableau 21: Résolution d'instances du p-médian de petites tailles par l'algorithme du Volume puis application d'une heuristique gloutonne de construction et d'une recherche locale. Les colonnes sont les même que celles du Tableau 22

Instance			Volume & Gr_{mod} & LS			Popstar			
Label	n	p	LB	UB	Tps	UB_{pop}	Tps_{pop}	Gap1	Gap2
bd26	5 999	500	26 391	26 609	909	26 575	605	0,81 %	0.69 %
bd26	5 999	750	24 029	24 189	880	24 153	1 028	0,66 %	0.51 %
bd26	5 999	1 000	22 131	22 178	837	22 166	1 009	0,21 %	0.15 %
bd26	5 999	1 250	20 361	20 425	859	20 416	1 016	0,31 %	0.26 %
bd26	5 999	1 500	18 807	18 853	847	18 849	1 138	0,24 %	0.22 %
bd26	5 999	2 000	15 795	15 855	880	15 850	1 174	0,37 %	0.34 %
bd27	8 000	500	40 241	40 722	2 363	40 662	1 447	1.18 %	1.03 %
bd27	8 000	750	37 323	37 591	1 821	37 547	1 548	0,71 %	0.59 %
bd27	8 000	1 000	34 939	35 118	2 037	35 068	1 912	0,50 %	0.36 %
bd27	8 000	1 250	32 836	32 974	1 726	32 946	1 938	0,41 %	0.33 %
bd27	8 000	1 500	30 906	30 986	1 673	30 975	2 160	0,25 %	0.22 %
bd27	8 000	2 000	27 410	27 499	1 710	27 465	2 154	0,32 %	0.20 %
bd29	8 990	500	36 928	37 445	2 548	37 429	1 230	1.38 %	1.33 %
bd29	8 990	750	34 402	34 647	2 443	34 603	1 690	0,70 %	0.58 %
bd29	8 990	1 000	32 235	32 437	2 522	32 391	2 003	0.62 %	0.48 %
bd29	8 990	1 250	30 367	30 520	2 782	30 493	2 573	0,50 %	0.41 %
bd29	8 990	1 500	28 660	28 770	2 583	28 745	2 571	0,38 %	0.29 %
bd29	8 990	2 000	25 640	25 694	2 298	25 684	3 040	0,21 %	0.17 %
bd28	9 999	500	50 572	51 279	3 333	51 196	2 686	1.37 %	1.21 %
bd28	9 999	750	47 066	47 527	3 097	47 484	2 671	0,96 %	0.88 %
bd28	9 999	1 000	44 304	44 648	2 942	44 600	2 639	0,77 %	0.66 %
bd28	9 999	1 250	41 937	42 244	2 948	42 190	2 802	0,72 %	0.59 %
bd28	9 999	1 500	39 932	40 113	3 349	40 072	3 178	0,45 %	0.34 %
bd28	9 999	2 000	36 269	36 376	2 737	36 344	3 933	0,29 %	0.20 %

Tableau 22: Résolution d'instances du p-médian de tailles moyennes par l'algorithme du Volume puis application de l'heuristique gloutonne modifiée et d'une recherche locale. La colonne "Label" donne le nom de l'instance. Les colonnes "n" et "p" présentent respectivement le nombre de sommets de l'instance et le nombre de centres à ouvrir. La colonne LB donne la valeur de la dernière résolution du sous problème Lagrangien. La colonne UB donne la valeur de la meilleure solution entière trouvée. La colonne "Temps" donne le temps mis par l'algorithme pour résoudre le problème. Les colonnes UB_{pop} et Tps_{pop} donnent la valeur de la meilleure solution et le temps d'exécution de l'heuristique hybride. La colonne Gap1 donne la différence relative entre notre solution entière et notre borne inférieure. La colonne Gap2 donne la différence relative entre la solution de l'heuristique hybride et notre borne inférieure.

Instance			Volume & Gr_{mod} & LS			
Label	n	p	LB	UB	Tps	Gap
bd33	14 960	1 000	49 582,85	50 131	7 218	1,09 %
bd33	14 960	1 500	45 276,35	45 684	6 572	0,89 %
bd33	14 960	2 000	41 937,05	42 186	6 731	0,59 %
bd33	14 960	2 500	38 996,67	39 147	8 242	0,38 %
bd34	14 997	1 000	71 695,78	72 421	7 134	1,00 %
bd34	14 997	1 500	66 268,27	66 718	6 956	0,67 %
bd34	14 997	2 000	61 833,46	62 146	7 099	0,50 %
bd34	14 997	2 500	57 903,97	58 145	7 183	0,41 %
bd23	15 000	1 000	261 521,15	262 378	6 097	0,32 %
bd23	15 000	1 500	248 829,83	249 226	5 768	0,16 %
bd23	15 000	2 000	236 702,47	237 270	5 929	0,24 %
bd32	15 000	1 000	76 103,74	76 998	7 670	1,16 %
bd32	15 000	1 500	70 367,14	70 868	7 839	0,70 %
bd32	15 000	2 000	65 671,86	65 969	7 643	0,45 %
bd32	15 000	2 500	61 514,72	61 746	6 615	0,37 %
bd35	15 994	1 000	70 798,70	71 560	7 886	1,06 %
bd35	15 994	2 000	61 159,78	61 570	8 004	0,66 %
bd35	15 994	2 500	57 377,51	57 573	9 011	0,34 %
bd30	20 000	1 500	164 199,21	165 902	10 895	1,03 %
bd30	20 000	2 000	155 329,32	156 465	12 482	0,72 %
bd30	20 000	2 500	147 469,12	148 589	13 430	0,75 %

Tableau 23: Résolution d'instances du p-médian de grandes tailles par l'algorithme du Volume puis application de l'heuristique gloutonne modifiée et d'une recherche locale. La colonne "Label" donne le nom de l'instance. Les colonnes "n" et "p" présentent respectivement le nombre de sommets de l'instance et le nombre de centres à ouvrir. La colonne LB donne la valeur de la dernière résolution du sous problème Lagrangien. La colonne UB donne la valeur de la meilleure solution entière trouvée. La colonne "Temps" donne le temps mis par l'algorithme pour résoudre le problème. Enfin la colonne Gap donne la différence relative entre la borne inférieure et la borne supérieure.

Chapitre 4

Modèles liés au problème du p-médian

4.1 Le dominant de la relaxation linéaire du problème de localisation de dépôts

Soit $P \subseteq \mathbb{R}^n$ un polyédre. Le *dominant* de P noté par $\mathrm{dom}(P)$ est défini comme suit :

$$\mathrm{dom}(P) = \{t \in \mathbb{R}^n : \ t = z + \alpha, \ z \in P, \alpha \geq 0\}.$$

Bien entendu $\alpha \in \mathbb{R}^n$ et par $\alpha \geq 0$ nous signifions que toutes les composantes de α sont non-négatives.

La motivation de décrire le dominant d'un polytope est la suivante. Supposons que nous voulons résoudre le problème $\{\min c^T x : \ x \in P\}$ où c est un vecteur à composantes non-négatives. Il n'est pas difficile de constater que ce problème revient à résoudre le problème $\{\min c^T x : \ x \in \mathrm{dom}(P)\}$. Il est en général plus facile d'optimiser sur le dominant d'un polytope que sur le polytope lui même. L'intuition derrière cette remarque est que $\mathrm{dom}(P)$ contient moins de sommets et moins de contraintes compliquées que P.

Quelques définitions et notations. Soit $G = (V, A)$ un graphe dirigé.

- Soit $S \subset V$. L'ensemble des arcs ayant l'extrémité initiale dans S et l'extrémité terminale dans $V \setminus S$ est appelé une *coupe* est noté par $\delta^+(S)$. Pour $v \in V$, on écrit $\delta^+(v)$ au lieu de $\delta^+(\{v\})$.
- Pour un sous-ensemble $F \subset A$, l'ensemble des extrémités terminales des arcs dans F sera noté par $H(F)$.
- Si F est un ensemble et x une fonction, $x : \ F \to \mathbb{R}$, alors pour chaque sous-ensemble $S \subseteq F$, on note $x(S) = \sum_{e \in S} x(e)$.

Rappelons le problème de localisation de dépôts sans capacités, LDSC. Soit $G = (V, A)$ un graphe dirigé où chaque arc (u, v) est associé avec un coût $c(u, v) \geq 0$ et chaque sommet v est associé avec un coût $f(v)$. Le problème

LDSC est de séléctionner un sous ensemble de sommets $S \subseteq V$ et d'affecter les sommets de $V \setminus S$ aux sommets S, tout en minimisant le coût de cette affectation plus la somme des coûts des sommets dans S. Appelons $P(G)$ le polytope de la relaxation linéaire classique et défini par l'ensemble des inégalités linéaires ci-dessous :

$$x(\delta^+(u)) + y(u) = 1, \quad \forall u \in v, \tag{4.1}$$
$$x(u,v) \leq y(v), \quad \forall (u,v) \in A, \tag{4.2}$$
$$y(u) \geq 0, \quad \forall u \in V, \tag{4.3}$$
$$x(u,v) \geq 0, \quad \forall (u,v) \in A. \tag{4.4}$$

Soit $D(G)$ le polyèdre suivant :

$$x(\delta^+(u) \setminus F) + y(u) + y(H(F)) \geq 1, \quad \forall u \in V \text{ et } F \subseteq \delta^+(u), \tag{4.5}$$
$$y(u) \geq 0, \quad \forall u \in V, \tag{4.6}$$
$$x(u,v) \geq 0, \quad \forall (u,v) \in A. \tag{4.7}$$

Théorème 4.1.1. *Pour tout graphe dirigé $G = (V, A)$, nous avons $dom(P(G)) = D(G)$.*

Preuve. Nous allons d'abord montrer que $\text{dom}(P(G)) \subseteq D(G)$. Soit $(x, y) \in \text{dom}(P(G))$. Par définition $(x, y) = (x', y') + (x'', y'')$ avec $(x', y') \in P(G)$ et $x'' \geq 0$ et $y'' \geq 0$. Il est clair que (x, y) vérifie les inégalités (4.6)-(4.7). Montrons que (4.5) est également vérifiée. Soit une telle inégalité par rapport à u et à $F \subseteq \delta^+(u)$. Donc nous voulons montrer l'inégalité suivante :

$$x(\delta^+(u) \setminus F) + y(u) + y(H(F)) \geq 1. \tag{4.8}$$

Puisque $(x', y') \in P(G)$, alors les inégalités suivantes sont vérifiées.

$$x'(\delta^+(u)) + y'(u) = 1, \tag{4.9}$$
$$x'(u,v) \leq y'(v) \quad \forall (u,v) \in F. \tag{4.10}$$

La somme des inégalités (4.9) et (4.10) nous donne

$$x'(\delta^+(u) \setminus F) + y'(u) + y'(H(F)) \geq 1. \tag{4.11}$$

Puisque x'' et y'' sont des vecteurs à composantes non-négatives l'inégalité (4.8) est obtenue en combinant $x'' \geq 0$, $y'' \geq 0$ et (4.11).

Dans ce qui suit nous allons se concentrer sur l'inclusion $D(G) \subseteq \text{dom}(P(G))$.

Soit $(x, y) \in D(G)$. Nous pouvons toujours supposer que $y(v) \leq 1$. Si $(x, y) \in P(G)$ il n y a rien à démontrer. Supposons donc que $(x, y) \notin P(G)$. Ceci est possible dans deux situations (a) une des égalités (4.1) est violée, ou (b) une des inégalités (4.2) est violée. Vérifons d'abord que dans tous les cas (a) est toujours vrai. Supposons le contraire c'est-à-dire toutes les égalités (4.1)

sont vérifiées et qu'il existe $(u', v') \in A$ tel que $x(u', v') > y(v')$, c'est-à-dire (b) est vrai. Dans ce cas, si nous posons $F = \{(u, v)\}$. Alors il est facile de voir que :

$$x(\delta^+(u') \setminus F) + y(u') + y(H(F)) < 1.$$

Ce qui contredit le fait que $x(, y) \in D(G)$. Désormais nous savons qu'il existe au moins un sommet $u \in V$ tel que

$$x(\delta^+(u)) + y(u) > 1 \tag{4.12}$$

Pour chaque sommet vérifiant (4.12), nous allons décroitre la valeur de $x(\delta^+(u)) + y(u)$ jusqu'à 1 en suivant la procédure suivante : Poser $x' = x$; $\beta = 0$. Ensuite appliquer la procédure ci-dessous pour tous les sommets de V successivement.

1. Si $x'(\delta^+(u)) + y(u) = 1$, Stop.
2. S'il existe un arc $(u, v) \in \delta^+(u)$ avec $x'(u, v) > y(v)$, faire
   (a) $\beta(u, v) = \min\{x'(\delta^+(u)) + y(u) - 1, x'(u, v) - y(v)\}$;
   (b) $x'(u, v) = x'(u, v) - \beta(u, v)$; aller à 1.
3. Choisir un arc avec $x'(u, v) > 0$ et faire
   (a) $\beta(u, v) = \min\{x'(\delta^+(u)) + y(u) - 1, x'(u, v)\}$
   (b) $x'(u, v) = x'(u, v) - \beta(u, v)$; aller à 1.

Evidemment, cette procédure s'arrête lorsque $x'(\delta^+(u)) + y(u) = 1$. Il est clair que $(x, y) = (x', y) + (x'', y)$ avec $x'' \geq 0$. Donc si nous voulons montrer que $(x, y) \in \text{dom}(P(G))$, il suffirait de montrer que $(x', y) \in P(G)$. Supposons le contraire. Par la procédure ci-dessus, toutes les égalités (4.1) sont vérifiées par (x', y). Donc il existe forcément $(u', v') \in A$ avec $x'(u', v') > y(v')$. Ce qui signifie que la procédure appliquée à u' n'a jamais atteint l'étape 3, et qu'en plus à une certaine itération k, $\beta(u', t)$ a pris la valeur $x'(\delta^+(u')) + y(u') - 1$ dans l'étape 2(a). C'est-à-dire que durant les étapes précédentes $\beta(u', w)$ a toujours pris la valeur $x'(u', w) - y(w)$ à l'étape 2(a). Donc à l'intération k nous avons une solution (x', y) obtenue à partir de (x, y) en remplaçant $x(u, v)$ par $y(v)$ pour un certain sous-ensemble d'arcs.

Formellement, appelons la solution (x', y) de l'étape k, (x^k, y). Et soit S le sous-ensemble d'arcs sortants de u' ayant la valeur modifiée à l'étape 2(b) durant les itérations précedentes. Donc nous avons :

$$x^k(\delta^+(u')) + y(u') > 1, \tag{4.13}$$

$$x^k(u', v) = y(v), \quad \forall (u', v) \in S. \tag{4.14}$$

Nous avons d'après l'étape 2(a) un arc $(u', t) \notin S$ avec :

$$x^k(u', t) - y(t) > x^k(\delta^+(u')) + y(u') - 1. \tag{4.15}$$

, cet arc peut éventuellement coincider avec (u', v') mais c'est sans importance. Rappelons que :

$$x^k(u', v) = x(u', v), \quad \forall (u', v) \notin S. \tag{4.16}$$

En combinant (4.14)-(4.16), nous obtenons :

$$x(\delta^+(u') \setminus S') + y(u') + y(H(S')) < 1, \tag{4.17}$$

où $S' = S \cup \{(u', t)\}$.

Ceci est une contradiction avec le fait $(x, y) \in D(G)$. □

Dans ce qui suit nous allons nous intéresser aux graphes G pour lesquels $D(G)$ est un polyhèdre entier, c'est-à-dire tous les sommets de $D(G)$ sont en 0-1.

Etant donné un graphe $G = (V, A)$, soit PLSC(G) l'enveloppe convexe des solutions en 0-1 qui vérifient les contraintes (4.1)-(4.4), généralement PLSC(G) est appelé le polytope du problème de localisations de dépôts sans capacités. Nous voulons étudier dom(PLSC(G)). D'après le résultat du Théorème 4.1.1, si PLSC(G)=$P(G)$ alors dom(PLSC(G))=D(G).

Une caractérisation des graphes pour lesquels PLSC(G)=$P(G)$ est donnée dans [10]. Il est clair que pour ces graphes nous avons aussi dom(PLSC(G))=D(G). Mais est-il vrai que ces graphes sont les seuls qui vérifient cette proprité ? c'est ce que nous allons voir dans ce qui suit. Mais avant, nous donnons des définitions nécessaires à l'étbalissement de notre résulat. Ces défnitions ont été introduites dans [7, 10].

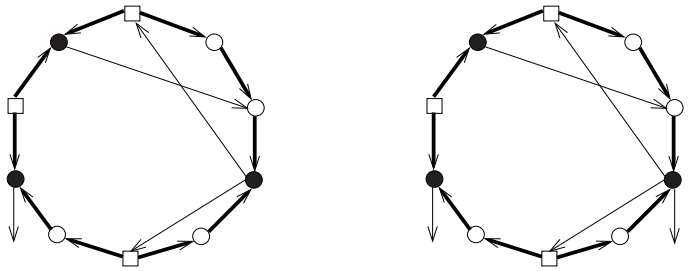
Un cycle dans $G = (V, A)$ est une séquence de sommets et d'arcs $v_0, a_1, v_1, a_2, \ldots, a_p, v_p = v_0$, telles que les extremités de chaque arc a_i sont v_{i-1} et v_i, pour $i = 1, \ldots, p$. Pour un cycle C, notons par $A(C)$ ses arcs et par $V(C)$ ses sommets. Distinguons les sommets d'un cycle C : $\tilde{C}$ ce sont les sommets qui sont à la fois une extrémité initiale et terminale d'un arc et d'un autre du cycle; $\dot{C}$ (resp. $\hat{C}$) sont ceux qui sont des extrémités initiales (resp. terminales) des arcs qui leur sont incidents dans le cycle, voir la Figure 17. Si le nombre de sommets dans $\dot{C}$ plus celui de $\tilde{C}$ est impair, alors le cycle est dit *g-impair*, dans le cas contraire il est *g-pair*. La lettre "g" est le diminutif du mot *généralisé*, car la notion classique des cycles impairs (ou pairs) fait uniquement tient compte du nombre de tous les sommets du cycle. Par exemple, les cycles dirigés sont des cas particuliers obtenus quand l'ensemble $\dot{C}$ (et donc $\hat{C}$) est vide.

Un Y-cycle C est un cycle où pour chaque sommet u dans $\hat{C}$ il existe un arc $(u, v) \in A$ avec v un sommet dans $V \setminus \dot{C}$.

Théorème 4.1.2. *[10] Soit $G = (V, A)$ un graphe dirigé. Nous avons PLSC(G) =$P(G)$ si et seulement si G ne contient pas un Y-cycle g-impair.*

Nous avons besoin de la caractérisation suivante d'un point extrême d'un polyèdre afin de démontrer le Théorème qui va suivre.

Caractérisation d'un point extrême. Soit $P = \{x \in I\!R^n : \ Ax \leq b\}$, un système linéaire. Un point $x^* \in P$ est un point extrême de P, s'il existe un sous-ensemble d'inégalités $A_0 x \leq b_0$ issues du système $Ax \leq b$ définissant P, tel que x^* est la soltuion unique du système $A_0 x = b_0$.



(a) (b)

Figure 17: Les carrés sont $\dot{C}$, en noir sont les sommets dans $\hat{C}$ et les blancs sont ceux de $\tilde{C}$. (a) désigne un cycle g-impair qui n'est pas un Y-cyle et (b) désigne un Y-cycle. Les arcs du cycle sont en gras.

Théorème 4.1.3. *Soit $G = (V, A)$ un graphe dirigé. Nous avons dom$(PLSC(G))$ $=D(G)$ si et seulement si G ne contient pas un Y-cycle g-impair.*

Preuve. Si G ne contient pas un Y-cycle g-impair alors, par les théorèmes 4.1.1 et 4.1.2, nous avons dom$(PLSC(G))$=$D(G)$. Maintenant supposons que G admet un Y-cycle g-impair apellé C. Pour qu'on puisse définir correctement un point extrême de $D(G)$, nous avons besoin de la définition suivante. Pour chaque $u \in \hat{C}$, définissons un arc particulier $(u, \hat{u}) \in \delta^+(u) \setminus A(C)$ et tel que $\hat{u} \notin \dot{C}$. Un tel arc existe toujours car C est un Y-cycle. Soit (x^*, y^*) défini comme suit, voir la Figure 18.

$$x^*(u,v) = \begin{cases} \frac{1}{2} & \text{si } u \in \hat{C} \text{ et } (u,v) = (u,\hat{u}), \\ \frac{1}{2} & \text{si } (u,v) \in A(C) \setminus \delta^+(\dot{C}), \\ 0 & \text{pour tout autre arc,} \end{cases} \qquad y^*(v) = \begin{cases} 1 & \text{if } v \notin V(C), \\ \frac{1}{2} & \text{if } v \in \tilde{C} \cup \hat{C}, \\ 0 & \text{if } v \in \dot{C}. \end{cases}$$

Nous affirmons que (x^*, y^*) est un point extrême de $D(G)$. Par définiton, les inégalités (4.5) quand $F = \emptyset$ sont vérifiées. Pour voir que la validité tient lorsque $F \neq \emptyset$, il suffit de noter que $x * (u, v) \leq y^*(v)$ pour tout arc (u, v). Il n'est pas difficile de vérifier que (x^*, y^*) est la solution unique du système ci-déssous:

$$\begin{cases} x(u,v) = 0, \text{ pour } (u,v) \notin A(C) \text{ et } (u,v) \neq (u,\hat{u}) \text{ quand } u \in \hat{C}, \\ x(\delta^+(u)) + y(u) = 1, \text{ pour tout } v \in V, \\ x\big(\delta^+(u) \setminus (u,v)\big) + y(u) + y(v) = 1, \text{ pour tout } u \in V(C) \setminus \hat{C}, \ (u,v) \in \delta^+(u) \cap A(C). \end{cases}$$

□

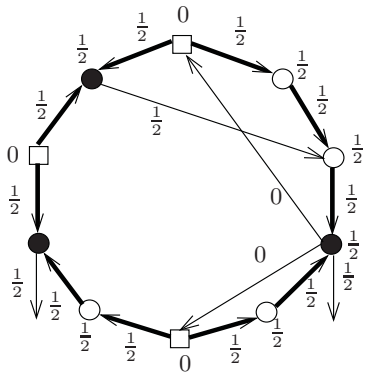


Figure 18: Le même cycle que celui de la Figure 17 (b). La valeur $y^*(v)$ est 1 quand v n'est pas dans le cycle. Les arcs qui ne sont pas représentés dans la figure prennent la valeur 0.

4.2 Résolution de la variante du problème de restructuration de bases de données semi-structurées

Nous avons déjà vu dans la Section 2.2.3 qu'il existait une variante au problème de restructuration de bases de données. Au lieu de simplement réduire ce problème à une instance du problème du p-médian nous voulons tenir compte du contexte. Il est possible de trouver des solutions de meilleure qualité en ajoutant des types fictifs bien choisis (voir l'exemple de la Section 2.2.3). Si ces types sont bien choisis ils peuvent réduire le coût des affectations et donc la perte de données.

Ce problème est donc un nouveau problème de restructuration de base de données. Il ne sagit plus seulement de trouver les meilleurs types existants. Il faut se poser la question de l'existance de types non présents dans la base initiale qui pourraient être de meilleurs types que ceux existants. Pour répondre à ce nouveau problème nous avons mis au point deux heuristiques.

4.2.1 Instances

Nous avons vu à la Section 2.3 comment obtenir des instances pour le problème de restructuration de données. Toutes ces instances ne peuvent pas être utilisées comme instance de ce nouveau problème. En effet, les deux heuristiques que nous avons développées pour y répondre se basent sur une idée simple : ayant connaissance des types du typage minimal parfait, nous aimerions en créer de nouveaux bien choisis. Et pour trouver ces nouveaux types, il nous faut la connaissance des types présents dans le typage minimal parfait. Cette simple condition ne nous permet pas d'appliquer ce problème à des instances de la TSPLIB par exemple.

Les seules instances dont nous disposons pour traiter ce problème sont les instances générées grâce à un typage minimal parfait. Comme nous l'avons expliqué dans la Section 2.3 nous avons peu d'instances de ce type, mais nous

avons une méthode de génération qui peut nous fournir une représentation de ces types, celle vue à la Section 2.3.3.

Pour les tests de nos heuristiques nous allons donc utiliser les instances générées aléatoirement en fonction de définitions de type. Les différents modes de génération de ces isntances peuvent être vu dans le Tableau 2.

4.2.2 Heuristiques

Notations et données

De manière à rendre les choses simples nous allons définir quelques notations et surtout une nouvelle manière de représenter les différents types présents dans notre typage parfait.

Chaque type τ_i présent dans le typage parfait est un ensemble de liens typés. Ces liens peuvent tous (pour l'ensemble de tous les types des objets de la base) être énumérés et numérotés. Nous pouvons alors représenter l'ensemble de nos types par des vecteurs contenant des 0 ou des 1. Les 1 signifient que le type considéré contient le lien typé correspondant (voir l'exemple de la Section 2.2.3 pour plus de détails).

Soit une base de données semi-structurée stockée sous la forme qui lui convient. Nous extrayons de cette base de données le typage parfait des données qui est stocké sous la forme de vecteurs de taille k, $\tau_i \in \{0,1\}^k$, pour $i = 1, \ldots, n$, où k est le nombre de liens typés. A ce vecteur est associé un poids, w_i qui représente le nombre d'objets ayant exactement ce type. Le nombre de types différents est n. Chaque composante du vecteur τ_i est noté a_i^j. Ainsi l'ensemble des types peut être écrit comme :

$$\tau_i = (a_i^1, a_i^2, \ldots, a_i^k) \text{ pour } i = 1, \ldots, n.$$

Une première heuristique

Le principe de l'heuristique est repris dans l'algorithme 21. L'idée est de créer de nouveaux types qui respectent la plupart des types présents. Nous allons simplement estimer la présence d'un lien typé dans l'ensemble des types de la base, avec ces informations nous allons créer des nouveaux types adaptés.

Nous commençons par calculer la valeur de la représentation d'un lien typé dans la base. Cette représentation s'exprime simplement par la formule :

$$\sigma[j] = \sum_i w_i a_i^j. \tag{4.18}$$

Cette valeur est ensuite ramenée à un pourcentage de la totalité des liens typés. Nous avons donc pour chaque lien typé j une valeur chiffrée (en pourcentage), $\sigma_{\%}[j]$, qui nous renseigne sur la présence du lien typé dans la base. Nous allons ensuite créer une nouvelle famille de type, les τ_ρ, pour $\rho = 1, \ldots, q$, $q \leq p$. Chacun de ces nouveaux types va obligatoirement contenir des liens typés, ceux pour lesquels $\sigma_{\%}[j]$ est supérieur à un certain seuil, s_1. Chacun de ces types va aussi obligatoirement ne pas contenir certains liens typés, ceux pour lesquels $\sigma_{\%}[j]$ est inférieur à un autre seuil, s_2. Enfin pour tous les liens typés incertains nous ne pouvons générer tous les types correspondant (rapidement un calcul de dénombrement montre que si 1/3 des liens typés ne sont pas décidés par les

Algorithme 21 Heur1(w, A, s_1, s_2, n, k, q)

Require: n: Nombre de types du typage parfait
Require: k: Nombre de liens typés différents
Require: q: Nombre de nouveau types à créer
Require: w: Tableau de taille n contenant les poids associés aux types
Require: A: Matrice de taille nxk contenant les vecteurs des types
Require: s_1, s_2: seuils d'inclusion ou d'exclusion automatique d'un lien typé
Ensure: B: Matrice de taille qxk contenant les q vecteurs des nouveaux types.

```
1: α ← Σ_{i=1}^{n} w_i
2: for j de 1 à k do
3:    σ[j] ← 0
4:    for i de 1 à n do
5:       σ[j] ← σ[j] + w_i A_i^j
6:    end for
7: end for
8: for j de 1 à n do
9:    σ_%[j] ← σ[j]/α
10: end for
11: for t de 1 à q do
12:    for j de 1 à k do
13:       if σ_%[j] > s_1 then
14:          B_t^j ← 1
15:       else
16:          if σ_%[j] < s_2 then
17:             B_t^j ← 0
18:          else
19:             B_t^j ← decider()
20:          end if
21:       end if
22:    end for
23: end for
```

seuils alors il faudrait générer $2^{n/3}$ types différents. Ce qui nous donne pour une base comprenant 30 liens typés différents quelques $2^{10} = 1024$ types à ajouter). Il faudra donc générer un sous ensemble de tous ces types, mais comment ? Pour le moment ce choix est contenu dans la fonction *decider*() qui va renvoyer un 0 ou un 1.

La fonction *decider*() est implémentée de manière simple. Nous considérons que pour un lien typé j que la valeur $\sigma_{\%}[j]$ correspont à une probabilité de présence ou non du lien typé dans le type. Ainsi plus $\sigma_{\%}[j]$ s'approche de s_1 plus il y aura de chances que ce liens typé se retrouve dans les types générés. Cette technique semble bien fonctionner. Les valeurs utilisées pour les seuils sont 60% (s_1) et 40% (s_2).

Une seconde heuristique

Le principe de cette heuristique est repris dans l'algorithme 22. L'idée ici est de créer de nouveaux types en fonction des clusters qui peuvent être formés à partir des distances. Si des types sont proches les uns des autres, alors un nouveau

type proche de cet ensemble doit pouvoir être créé, de manière a réduire le coût total des affectations.

Algorithme 22 Heur(G, n, p, A, k)

Require: G: le graphe originel
Require: n: la taille du graphe
Require: k: le nombre de liens typé dans la base
Require: p: le nombre de type à conserver
Require: A: Matrice de taille nxk contenant les vecteurs des types
Ensure: G': un nouveau graphe prenant en compte les nouveaux sommets

```
1: x ← resoudre(G, n, p)
2: for C_i cluster de x do
3:    #Il faut générer un nouveau type T_i
4:    oldCost ← affectationCost(C_i)
5:    for j de 1 à k do
6:       for l de 1 à q do
7:          d_0^j ← d_0^j + w_l A_l^j
8:          d_1^j ← d_1^j + w_l (1 − A_l^j)
9:       end for
10:      if d_0^j < d_1^j then
11:         T_i^j ← 0
12:      else
13:         T_i^j ← 1
14:      end if
15:   end for
16:   #Comparaison entre le nouveau type et les types existants
17:   newCost ← affectCost(C_i, T_i)
18:   if newCost < oldCost then
19:      reAffect(x, C_i, T_i)
20:      T ∪ T_i
21:   end if
22: end for
23: G' ← generateGraph(G, T)
```

La première étape est bien évidement de trouver ces clusters. Ceci peut être facilement réalisé grâce à l'application d'un p-médian sur le graphe des types. En effet le résultat de cette exécution sera non seulement les types choisis comme centres, mais aussi l'ensemble des types affectés à un centre. Considérons un centre et l'ensemble des sommets qui lui sont affectés comme un cluster C_i, nous avons donc p clusters C_i. Chaque cluster est constitué d'un certain nombre de sommets et du centre, nous notons alors q_i le nombre total de sommets d'un cluster C_i.

Pour chacun de ces clusters nous créons un nouveau type T_i, et pour chacune des composantes j de T_i nous calculons deux distances :

- la distance $d_0 = \sum_{k \in C_i} w_k A_k^j$,
- la distance $d_1 = \sum_{k \in C_i} w_k (1 - A_k^j)$.

Une fois ces distances calculées nous pouvons les comparer, nous choisissons alors de fixer la composante j de T_i. Si la distance d_0 est plus faible que la distance d_1 alors la composante j est mise à 0, ou sinon elle est fixée à 1. Cette opération est appliquée à toutes les composantes de T_i. Ce type peut alors être inséré ou non au problème en fonction de la modification sur la solution qu'il implique. Il est possible de calculer l'impact local d'un tel sommet sur les affectations des sommets du cluster.

En effet, étant donnée un cluster C_i, il est facile de calculer le coût qu'il induit. Ce coût est calculé grâce à la formule suivante :

$$cost_i = \sum_{u \in C_i \setminus \{v\}} c(u, v),$$

où v est le centre du cluster C_i. De la même façon si nous notons par t_i le sommet qui vient d'être ajouté, le coût de l'affectation de tous les sommets de C_i à ce nouveau centre se calculera de la manière suivante :

$$newCost_i = \sum_{u \in C_i} c(u, t_i).$$

Il suffit alors de comparer $cost_i$ et $newCost_i$ pour savoir s'il faut intégrer ce sommet ou non.

Pour gérer leur insertion nous allons commencer par rajouter un sommet puit à notre graphe. Ce sommet sera choisis comme centre, pour nous en assurer il suffit de s'assurer que ce sommet ne possède aucune arête sortante. Il ne pourra donc jamais être affecté à un autre sommet. Pour éviter aussi que l'un des sommets originels, u, ne soit affecté à ce sommet *puit* nous nous assurons qu'il n'existe aucune arête entre u et *puit*. Enfin nous devons relier les nouveaux sommets au reste du graphe, ce qui est fait comme suit. Soit u le sommet que nous souhaitons ajouter et v un autre sommet du graphe, différent du sommet puit. Nous nous assurons qu'il n'existe aucune arête (u, v) et l'on pose

$$c(v, u) = w_v * d(v, u),$$

où w_v est le poids du sommet (ou le nombre d'objets de la base ayant le type associé, 1 pour les nouveaux sommets) et $d(v, u)$ la distance de manhattan entre les types réprésentés par les sommets u et v.

Une fois tous les nouveaux types calculés et insérés ou non, nous pouvons aussi relancer une résolution du problème du p-médian si l'on souhaite. Cette étape peut fournir de meilleurs résultats que ceux donnés par l'étape de recalcul des valeurs car il est possible que certains sommets soient réaffectés hors de leur cluster d'origine.

4.2.3 Résultats de calcul

Le Tableau 24 présente différents résultats de l'application de l'heuristique décrite par l'Algorithme 21 sur des instances générées aléatoirement. Nous n'utilisons que ces instances car nous avons besoin de connaitre les différents types composant le typage minimal parfait.

Le Tableau 24 se présente de la manière suivante. Les quatre premières colonnes décrivent l'instance considérée. Une instance se compose d'un label (le

Instance				Heuristique		Initial	
Label	n	p	# Nouv Type	LB_{atf}	UB_{atf}	LB_{stf}	UB_{stf}
bd02	2 758	200	200	4 040,20	4 150	5 097,73	5 181
bd02	2 758	400	200	3 015,00	3 086	3 749,13	3 773
bd02	2 758	600	200	2 466,03	2 485	2 999,03	3 009
bd02	2 758	800	200	2 065,91	2 083	2 466,82	2 479
bd02	2 758	1 000	200	1 753,39	1 759	2 068,02	2 079
bd03	2 742	200	200	3 883,10	3 980	5 016,39	5 097
bd03	2 742	400	200	2 893,77	2 956	3 715,93	3 744
bd03	2 742	600	200	2 388,84	2 407	2 938,17	2 950
bd03	2 742	800	200	1 988,23	2 012	2 429,17	2 436
bd03	2 742	1 000	200	1 735,68	1 742	2 031,38	2 036
bd04	2 389	200	200	3 126,55	3 224	4 328,24	4 378
bd04	2 389	400	200	2 297,98	2 350	3 083,03	3 099
bd04	2 389	600	200	1 861,62	1 888	2 380,78	2 385
bd04	2 389	800	200	1 586,41	1 589	1 909,46	1 923
bd04	2 389	1 000	200	1 387,15	1 389	1 518,85	1 523
bd05	2 766	200	200	4 068,22	4 180	5 170,23	5 251
bd05	2 766	400	200	3 017,35	3 093	3 814,45	3 848
bd05	2 766	600	200	2 472,99	2 491	3 022,41	3 031
bd05	2 766	800	200	2 066,34	2 097	2 468,81	2 478
bd05	2 766	1 000	200	1 760,82	1 766	2 071,65	2 078
bd06	3 491	200	200	7 001,04	7 123	7 528,10	7 660
bd06	3 491	400	200	5 573,25	5 635	5 974,44	6 013
bd06	3 491	600	200	4 691,54	4 724	5 023,97	5 047
bd06	3 491	800	200	4 044,30	4 072	4 305,04	4 319
bd06	3 491	1 000	200	3 451,73	3 471	3 712,96	3 720

Tableau 24: Différents résultats de l'applications de la première heuristique décrite par l'Algorithme 21 sur des instances simulant des bases de données semi-structurées. La colonne "Label" donne le nom de l'instance. Les colonnes "n" et "p" présentent respectivement le nombre de sommets de l'instance et le nombre de centres à ouvrir. La colonne "# New Type" indique combien de nouveaux types peuvent être créés. Les colonnes LB_{atf} et UB_{atf} donnent les valeurs de la borne inférieure et de la borne supérieure du problème avec les types fictifs. Les colonnes LB_{stf} et UB_{stf} donnent les même valeurs mais pour le problème sans les types fictifs.

nom de l'instance), d'un nombre de sommets dans le graphe (n), de la taille du sous-ensemble des types que l'on souhaite conserver (p) ainsi que du nombre de types fictifs que l'on a ajouté. Ici pour chacun des tests nous avons arbitrairement choisis d'ajouter au maximum 200 types. Pour chacune des instances envisagées 200 types ont effectivement été ajoutés.

Les quatre dernières colonnes présentent les résultats de l'application d'un problème de p-médian à deux graphes. Les deux graphes considérés sont le graphe avec ajout des types fictifs (atf) et sans cet ajout (stf). Pour chaque graphe nous proposons deux résultats. Le premier (colonne LB) présente la valeur de la relaxation linéaire du problème fournie par l'algorithme du Volume.

Le second (colonne UB) expose la valeur de la meilleur solution entière trouvée.

Pour chacune des expériences nous trouvons des solutions entières pour le problème avec ajout de types fictifs ayant une valeur inférieure à la valeur de la relaxation linéaire du problème sans ces ajouts. Ainsi, partant de la même base de donnée, nous arrivons à obtenir une réaffectation des types des objets de cette base à un coût plus faible. Ceci est possible grâce aux types que nous ajoutons.

Instance				Heuristique			
Label	n	p	# Nouv Type	LB_{Init}	UB_{Init}	UB_{Heur}	UB_{Heurv2}
bd02	2758	200	200	5 097,73	5 181	5 181	5 181
bd02	2758	400	400	3 749,13	3 773	3 773	3 772
bd02	2758	600	600	2 999,03	3 009	3 009	3 008
bd02	2758	800	800	2 466,82	2 479	2 479	2 476
bd02	2758	1 000	1 000	2 068,02	2 079	2 079	2 079
bd03	2742	200	200	5 016,39	5 097	5 097	5 097
bd03	2742	400	400	3 715,93	3 744	3 744	3 743
bd03	2742	600	600	2 938,17	2 950	2 950	2 950
bd03	2742	800	800	2 429,17	2 436	2 436	2 434
bd03	2742	1 000	1 000	2 031,38	2 036	2 035	2 035
bd04	2389	200	200	4 328,27	4 378	4 378	4 378
bd04	2389	400	400	3 083,03	3 099	3 099	3 099
bd04	2389	600	600	2 380,78	2 385	2 385	2 385
bd04	2389	800	800	1 909,46	1 923	1 922	1 922
bd04	2389	1 000	1 000	1 518,85	1 523	1 523	1 523
bd05	2766	200	200	5 170,23	5 251	5 251	5 251
bd05	2766	400	400	3 814,45	3 848	3 848	3 848
bd05	2766	600	600	3 022,41	3 031	3 031	3 031
bd05	2766	800	800	2 468,81	2 478	2 478	2 478
bd05	2766	1 000	1 000	2 071,65	2 078	2 077	2 077
bd06	3491	200	200	7 528,10	7 660	7 660	7 656
bd06	3491	400	400	5 974,44	6 013	6 008	6 008
bd06	3491	600	600	5 023,97	5 047	5 042	5 042
bd06	3491	800	800	4 305,04	4 319	4 314	4 314
bd06	3491	1 000	1 000	3 712,96	3 720	3 720	3 716

Tableau 25: Différents résultats de l'applications de la seconde heuristique décrite par l'Algorithme 22 sur des instances simulant des bases de données semi-structurées. La colonne "Label" donne le nom de l'instance. Les colonnes "n" et "p" présentent respectivement le nombre de sommets de l'instance et le nombre de centres à ouvrir. La colonne "# New Type" indique combien de nouveaux types peuvent être créés. Les colonnes LB_{Init} et UB_{Init} donnent les valeurs de la borne inférieure et de la borne supérieure pour le problème initial. La colonne UB_{Heur} donne la valeur de la meilleure solution entière trouvée par l'heuristique, sans relancer la résolution d'un problème de p-médian. La colonne UB_{Heurv2} donne la même chose mais après ajout de tous les types calculés et re-résolution d'un problème de p-médian modifié.

Comme le Tableau 24, le Tableau 25 montre différents résultats de l'application de l'heuristique décrite par l'Algorithme 22. Les quatre premières colonnes sont identiques à celles du Tableau 24. Cependant pour cette heuristique nous tentons d'ajouter autant de sommets fictifs que de centres à selectionner.

Les quatre dernières colonnes présentent le résultat de cette heuristique. Dans la colonne LB_{Init} nous donnons la valeur de la relaxation linéaire du problème initial. La colonne suivante (UB_{Init}) fournit la valeur de la meilleure solution entière trouvée pour le problème original. La colonne suivante (UB_{Heur}) montre la valeur de la solution fournie par l'heuristique. Cette solution est la solution ne tenant pas compte d'une possible réaffectation des sommets hors de leur cluster. Enfin la dernière colonne présente la valeur de la solution après la seconde résolution du problème, sur le graphe modifié cette fois. Pour cette valeur nous avons considéré que tous les sommets calculés puissent être ajouté, qu'ils aient fournit ou non une amélioration directe.

Nous constatons que cette heuristique n'est pas efficace du tout. A aucun moment elle ne permet de trouver une solution (que ce soit par amélioration directe ou par re-résolution du problème) qui soit en dessous de la borne inférieure du problème initial. Cette heuristique n'est donc pas utilisable pour résoudre cette version du problème.

Conclusion

Dans cette thèse nous nous sommes intéressés à la résolution du problème du p-médian pour des instances de très grandes tailles. Notre motivation pour résoudre ces instances est la résolution d'un problème de base de donnée, mais notre technique de résolution peut être appliquée à d'autres problèmes.

Dans le premier chapitre nous avons rappelé des applications du problème du p-médian. Ces applications peuvent être très variées, allant de la localisation des centres de commutation dans un réseau de télécommunication à la localisation des lieux d'ouverture de comptes bancaires pour un société souhaitant maximiser ses liquidités. Suite au descriptif de ces applications nous avons rappelé les différentes méthodes existante pour résoudre ce problème. Dans la littérature personne ne s'intéresse à la résolution d'instances aussi grandes que les notres. La meilleure heuristique, qui peut résoudre des instances de grandes tailles, est l'heuristique hybride [90].

Dans le Chapitre 2 nous expliquons de manière précise le problème de restructuration de bases de données semi-structurées. Dans ce chapitre nous introduisons aussi une variante de ce problème qui est plus difficile à modéliser et à résoudre. Cette variante du problème, que l'on nomme variante difficile, n'est pas étudiée dans la littérature. Enfin nous détaillons les différentes instances du problème du p-médian qui seront utilisées pour effectuer nos expériences.

Dans le Chapitre 3 nous nous intéressons en détails à la résolution du problème du p-médian. Notre technique de résolution se base sur l'utilisation de l'algorithme du Volume pour résoudre la relaxation linéaire du problème. L'algorithme du Volume est une variante de la méthode du sous-gradient permettant d'obtenir une solution primale en plus de la solution duale.

De manière à obtenir des solutions rapidement nous avons souhaité avoir une résolution de la relaxation linéaire rapide. L'algorithme du Volume permet d'obtenir rapidement des solutions pour des problèmes de petites tailles. Cependant pour des problèmes de très grandes tailles cette algorithme était trop long. Nous avons donc mis à profit le fait qu'il existe au sein d'un ordinateur plusieurs processeurs permettant de réaliser des calculs de manière parallèle. Nous avons donc réalisé une implémentation parallèle de l'algorithme du Volume pour le p-médian.

Une fois calculée, la solution de la relaxation linéaire est alors transformée en une solution entière par des techniques heuristiques. Nous décrivons quatre heuristiques ainsi qu'une méta-heuristique que nous utilisons.

Au cours de nos travaux nous avons toujours souhaité nous comparer à l'heuristique hybride. Le principe de l'heuristique hybride est de générer aléatoirement une solution, puis d'effectuer une recherche locale à partir de cette solution, et enfin de combiner cette solution avec une précédement trouvée grâce à

un algorithme de *Path Relinking*. Une fois un certain nombre de ces itérations effectués un ensemble de 10 solutions est trouvé. L'heuristique effectue alors un croisement entre chacune de ces solutions grâce à un *Path Relinking*, et si une nouvelle solution est découverte alors une recherche locale est effectuée. Nous nous contentons de resoudre la relaxation linéaire, d'utiliser cette solution dans une heuristique de construction gloutonne puis de faire une recherche locale.

Les efforts que nous effectuons pour transformer une solution fractionnaire en une solution entière sont minimes devant les efforts effectués par l'heuristique hybride pour trouver une solution. Et pourtant, dans nos expériences finales, nous arrivons de temps en temps à trouver des solution meilleures que celles trouvées par l'heuristique hybride et dans le pire des cas nous ne sommes pas à plus de $0,5\%$ de la solution de l'heuristique hybride.

Dans le Chapitre 4 nous décrivons deux modèles liés à nos travaux. Le premier est le dominant du problème de localisation de dépôts sans capacité. Ensuite nous nous sommes intéressés à la résolution de la variante du problème de restructuration de bases de données. Nous décrivons ici deux heuristiques permettant la résolution de ce problème.

Cette thèse ouvre des perspectives intéressantes. Le premier point intéressant est la résolution de la relaxation linéaire du problème du p-médian en des temps de calculs très faibles comparés aux temps de calculs des solveurs commerciaux. Nous avons utilisé l'algorithme du Volume pour résoudre la relaxation linéaire de notre problème. L'utilisation de cet algorithme a permis de réduire nos temps de calculs. Pour les réduire encore nous avons réaliser une parallélisation de l'algorithme du Volume pour le p-médian. L'étape de transformation de la solution fractionnaire en une solution entière compte parmi les techniques les plus rapide compte tenu de la qualité des solutions trouvées.

Les deux heuristiques développées à la fin de ce document permettent d'apporter une meilleure réponse au problème de restructuration de bases de données qu'une simple résolution du problème du p-médian associé. Les résultats montrent que pour certains problèmes nous obtenons des solutions dont la valeur est largement inférieure à l'optimum atteignable par la première technique.

Bibliography

[1] S. Abiteboul, "Querying semi-structured data" In *Proceedings of ICDT*, Delphi, Grece, January 1997, 1-18.

[2] R. Anbil and F. Barahona, "The volume algorithm: producing primal solutions with a subgradient method," *Mathematical Programming* 87 (2000) 385-399.

[3] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala and V. Pandit, "Local Search Heuristics for k-Median and Facility Location Problems." In Proceedings of the 30th Annual ACM Symposium on Theory of Computing, 2001.

[4] P. Avella and A. Sassano, "On the median polytope," *Mathematical Programming* 89 (2001) 395-411.

[5] P. Avella, A. Sassano and I. Vasil'ev, "Computational study of large-scale p-Median problems," *Mathematical Programming* 109 (2007) 89-114.

[6] M. Baïou and F. Barahona, "On the the p-median polytope of a Y-free graphs," *Discrete Optimization* 5 (2008) 205-219.

[7] M. Baïou and F. Barahona, "On the integrality of some facility location polytopes", *SIAM Journal on Discrete Mathematics* 23 (2009) 665-679.

[8] M. Baïou and F. Barahona, "On the linear relaxation of the p-median problem," *Discrete Optimization* 8 (2011) 344-375.

[9] M. Baïou, F. Barahona and J. R. Correa, "On the p-median polytope and the intersection property: polyhedra and algorithms," *SIAM Journal on Discrete Mathematics* 25 (2011) 1-20.

[10] M. Baïou and F. Barahona, "On a connection between facility location and perfect graphs," paper under review.

[11] M. L. Balinski, "Integer Programming: Methods, Uses, Computations," *Management Science* 12 (1965) 253-313.

[12] F. Barahona and F. Chudak, "Solving large scale uncapacitated facility location problems," NONCONVEX OPTIMIZATION AND ITS APPLICATION, *Approximation and Complexity in Numerical Optimization*, Continuos and Discrete Problems, In Panos M. Pardalos (Ed.), (2000) 48-62.

[13] F. Barahona and L. Ladanyi, "An implementation of the Volume Algorithm," *Coin-or Documentation* (2006) available at URL https://projects.coin-or.org/svn/Vol/stable/1.1/Vol/doc/volDoc.pdf.

[14] Y. Bartal, "Probabilistic approximation of metric spaces and its algorithmic applications." In Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science, 184-193, 1996.

[15] Y. Bartal, "On approximating arbitrary metrics by tree metrics." In Proceedings of the 30th Annual ACM Symposium on Theory of Computing, 161-168, 1998.

[16] W. J. Baumol and P. Wolfe, "A Warehouse Location Problem," *Operations Research* 6 (1958) 252-263.

[17] J. E. Beasley, "A note on solving large scale p-median problems," *European Journal of Operational Research* 21 (1985) 270-273.

[18] J. E. Beasley, "Lagrangean heuristics for location problems," *European Journal of Operational Research* 65 (1993) 383-399.

[19] J. E. Beasley, "OR-library: distributing test problems by electronic mail," Technical Report, The Management School, Imperial College (available at URL http://people.brunel.ac.uk/ mastjjb/jeb/info.html).

[20] P. Belotti, M. Labbé, F. Maffioli and M. Ndiaye, "A branch-and-cut method for the obnoxious p-median problem," *4OR* 5 299-314.

[21] O. Berman, Z. Drezner, A. Tamir and G. O. Wesolowsky, "Optimal location with equitable loads," *Annal of Operations Research* 1(2009) 307-325.

[22] O. Briant and D. Naddef, "The optimal diversity management problem," *Operations Research* 52 (2004) 515-526.

[23] P. Bunemane, "Semistructured data: a tutorial" In *Proceedings of PODS*, Tucson, Arizona, May 1997.

[24] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu, "A query language and optimization techniques for unstructured data," In *Proceedings of the ACM SIGMOD International Conference* Montreal, Canada, June 1996 505-516

[25] L. Cánovas, S. García, M. Labbé and A. Marín, "A strengthened formulation for the simple plant location problem with order," *Operations Research Letters* 35 (2007) 141-150.

[26] L. Cánovas, M. Landete and A. Marín, "On the facets of the simple plant location packing polytope," *Discrete Applied Mathematics* 124 (2002)27-53.

[27] M. Charikar and S. Guha, "Improved Combinatorial Algorithms for the Facility Location and k-Median problems." Proceedings of the 40th Annual IEEE Conference on Foundations of Computer Science, 1999.

[28] M. Charikar, S. Guha, E. Tardos and D. B. Shmoys, "A constant-factor approximation algorithm for the k-median problem," STOC'99 Proceedings of the thirty-first annual ACM symposium on Theory of computing, 1-10, 1999.

[29] M. Charikar, C. Chekuri, A. Goel and S. Guha, "Rounding via trees: deterministic approximation algorithms for group Steiner trees and k-median." In Proceedings of the 30th Annual ACM Symposium on Theory of Computing, 114-123, 1998.

[30] M. Charikar, S. Guha, E. Tardos and D. B. Shmoys, "A constant-factor approximation algorithm for the k-median problem," *Journal of Computer and System Sciences* 65 (2002) 129-149.

[31] D. C. Cho, E. L. Johnson, M. Padberg and M. Rao, "On the uncapacitated facility location problem I: Valid inequailities and facets," *Mathematics of Operations Research* 8 (1983) 579-589.

[32] D. C. Cho, M. Padberg and M. Rao, "On the uncapacitated facility location problem II: facets and lifting theorems," *Mathematics of Operations Research* 8 (1983) 590-612.

[33] M. Chrobak, L. L. Larmore and W. Rytter, "The k-Median Problem for Directed Trees," J. Sgall, A. Pultr and P.L Kolman (Eds.): MFCS 2001, *Lecture Notes on Computer Science* 2136 (2001) 260-271.

[34] R.L. Chuch, "COBRA: a new formulation of the classic p-median location problem, *Annals of Operations Research* 122 (2003) 103-120.

[35] R. L. Church and J. L. Cohon, "Multiobjective location analysis of regional energy facility siting problems," Report preparated for the U. S. Energy Research and Development Administration (BNL 50567).

[36] R. L. Church and K. L. Roberts, "Generalized coverage models and public facility location," *Papers in Regional Science* 53 (1983) 117-135.

[37] COmputational INfrastructure for Operatinos Research, URL : http://coin-or.org/.

[38] G. Cornuéjols, M. L. Fisher and G. L. Nemhauser, "Location of bank accounts to optimize float : an analytic study of exact and approximate algorithms," *Mangement Science* 23 (1977) 789-810.

[39] G. Cornuéjols, G. L. Nemhauser, L. Wolsey, "A canonical representation of simple plant location problems and its applications," *SIMAX* 1 (1980) 261-272.

[40] G. Cornuéjols, J.-M. Thizy, "Some facets of the simple plant location polytope," *Mathematical programming* 23 (1982) 50-74.

[41] N. Christofides and J. E. Beasley, "A tree search algorithm for the p-median problem," *European Journal of Operational Research* 10 (1982) 196-204.

[42] I. R. de Farias, "A family of facets for the p-median polytope," *Operations Research Letters* 28 (2001) 161-167.

[43] S. de Vries, M. E. Posner, R. V. Vohra, "Polyhedral properties of the K-median problems on a tree," *Mathematical Programming* 110 (2003) 261-285.

[44] G. Dobson and U. S. Karmarkar, "Competitive location on a network", *Operations Research* 35 (1987) 565-574.

[45] M. Efroymson and T. Ray, "A branch-and-bound algorithm for plant location," *Operation Research* 14 (1966) 361-368.

[46] S. Elloumi, "A tighter formulation of the p-median problem," *Journal of Combinatorial Optimization* 19 (2010) 69-83.

[47] R. Z. Farahani and M. Hekmatfar (Eds.), "Facility Location, Concepts, Models, Algorithms and Case Studies," Physica-Verlag A Springer Company, 2009.

[48] F. García-López, B. Melián-Batista, J. A. Moreno-Pérez and J. M. Moreno-Vega, "The Parallel variable neighborhood search for the p-median problem," *Journal of Heuristics* 8 (2002) 375-388.

[49] M. R. Garey and D. S. Johnson, "Computers and intractability: A Guide to the theory pf
$mathcal NP$-completeness", W. H. Freeman and Co., San Francisco, CA, 1979.

[50] A. M. Geoffrion, "Lagrangian Relaxation for Integer Programming," *Mathematical Study* 2 (1974) 82-114.

[51] R. A. Gerrard and R. L. Church, "Closest assignment constraints and location model: properties and structure," *Location Science* 4 (1996) 251-270.

[52] M. Grötschel, L. Lovász and A. Schrijver,"The ellipsoid method and its consequences in combinatorial optimization," *Combinatorica* 1 (1981) 70-89.

[53] M. Guinard, "Fractional vertices, cuts and facets of the simple plant location problem," *Mathematical Programming Study* (1980) 150-162.

[54] S. L. Hakimi, "Optimum Locations of Switching Centers and the Absolute Centers and Medians of a Graph," *Operations Research* 12 (1964) 450-459.

[55] S. L. Hakimi, "Optimum Distribution of Switching Centers in a Communication Network and Some Related Graph Theoretic Problems," *Operations Research* 13 (1965) 462-475.

[56] P. Hanjoul and D. Peeters, "A facility location problem with client' preference orderings," *Regional Science and Urban Economics* 17 (1987) 451-473.

[57] P. Hanjoul, P. Hansen, D. Peeters and J. F. Thisse, "Uncapacitated plant location under alternative spatial price policies, *Management Science* 36 (1990) 41-57.

[58] P. Hansen and N. Mladenović, "Variable neighborhood search for the p-median," *Location Science* 5 (1997) 207-226.

[59] P. Hansen, N. Mladenović and D. Perez-Brito, "Variable neighborhood decomposition search," *Journal of Heuristics* 7 (2001) 335-350.

[60] R. Hassin and A. Tamir, "Improved complexity bounds for location problems on the real line," *Operations Research Letters* 10 (1991) 395-402.

[61] R. W. Hause and T. Rado, "On a Computer Program for Obtaining Irreducible Representations for Two-Level Multiple Input-Output Logical Systems," *Journal of the Association Computing Machinery* 48-77 (1963).

[62] M. Held, R.M. Karp, "The travelling salesman problem and minimum spanning trees," *Operations Research* 18 (1970) 1138-1162.

[63] M. Held, R.M. Karp, "The travelling salesman problem and minimum spanning trees: Part II," *Mathematical Programming* 1 (1971) 6-25.

[64] M. Held, P. Wolfe, H.P. Crowder, "Validation of subgradient optimization," *Mathematical Programming* 6 (1974) 62-88.

[65] M. J. Hodgson, "Toward more realistic allocation in location-allocation models: An interaction approach," *Environment and Planning A* 10 (1978) 1273-1285.

[66] W. L. Hsu, "The distance-domination numbers of trees," *Operations Research Letters* 1 (1982) 96-100.

[67] K. Jain and V. Vazirani, "Primal-dual approximation algorithms for metric facility location and k-median problems." Proceedings of the 40th Annual Symposium on Foundation of Computer Science, 1999.

[68] K. Jain and V. Vazirani, "Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation," *Journal of the ACM* 48 (2001) 274-296.

[69] P. Jarvinen , J. Rajala and J. Sinerro, "A Branch-and-Bound Algorithm for Seeking the p-Median," *Operations Research* 20 (1972) 173-178.

[70] O. Kariv and S. L. Hakimi, "An algorithmic approach to network location problems, Part II: p-medians," *SIAM Journal on Applied Mathematics* 37 (1979) 539-560.

[71] R. Karp and C. Papadimitriou, "On linear characterization of combinatorial optimization problems," Proceedings of the Twenty-first annual Symposium on the Foundation of Computer Science, IEEE (1980) 1-9.

[72] L. G. Khachiyan, "A polynomial algorithm in linear programming," *Soviet Mathematics Doklady* 20 (1979) 191-194.

[73] A. A. Kuehn and M. J. Hamburger, "A Heuristic Program for Locating Warehouse," *Management Science* 9 (1963) 643-666.

[74] C. Lemaréchal, "An extension of Davidson methods to non differentiable problems," *Mathematical Programming Study* 3 (1975) 95-109.

[75] B. Li, X. Deng, M. J. Golin, K. Sohraby, "On the optimal placement of Web proxies on the Internet," in: Proc. 8th IFIP Conference on High Performance Nteworking (HPN'98), 1998.

[76] J.-H. Lin and J. S. Vitter, "ϵ-approximation with minimum packing constraint violation." in proceedings of the 24th Annual ACM Symposium on Theory of Computing, 771-782, 1992.

[77] , A. S. Manne, "Plant Location under Economics-of-Scale-Decentralization and Computation," *Management Science* 11 (1964) 213-235.

[78] , M. P. Marcus, "Switching Circuits for Engineers," Prentice-Hall, Englewood Cliffs, N. J., 1962.

[79] A. Marín, "The discrete facility location problem with balanced allocation of customers," *European Journal of Operations Research* 210 (2011) 27-38.

[80] N. Megiddo and K. J. Supowit, "On the Complexity of Some Common Geometric Location Problems," *SIAM Journal on Computing* 13 (1984) 182-196.

[81] G. Melançon and I. Dutour and M. Bousquet-Mélou, "Random Generation of Directed Acyclic Graphs" *Electronic Notes in Discrete Mathematics* 10 (2001) 202-207.

[82] P. B. Mirchandani and R. L. Francis, eds., "Discrete location theory," Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons Inc., New York, 1990.

[83] J. G. Morris, "On the extent to which certain fixed-charge depot location problems can be solved by LP," *Journal of the Operational Research Society* 29 (1978) 71-76.

[84] G.L. Nemhauser, L.A. Wolsey, "Interger and Combinatorial Optimization," Wiley, New York, 1988.

[85] S. Nestorov and S. Abiteboul and R. Motawani, "Extracting Schema from Semistructured Data" *SIGMOD Rec.* 27 (1998) 295-306.

[86] M. W. Padberg and MR. Rao, "The Russian method for linear inequalities III: Bounded integer programs," GBA Working paper 81-39 New York University, New York (1981).

[87] C. H. Papadimitriou, "Worst-case and Probabilistic Analysis of a Geometric Location Problem," *SIAM Journal on Computing* 10 (1981) 542-557.

[88] M. G. C. Resende and R. F. Werneck, "A GRASP with path-relinking for the p-median problem," Technical Report. AT&T Labs Research, Florham Park, NJ 07932 (2002).

[89] M. G. C. Resende and R. F. Werneck, "On the implementation of a swap-based local search procedure for the p-median problem", In R. E. Ladner, editor, *Proceedings of the Fifth Workshop on Algorithm Engineering and Experi(ALENEX'03)* (2003) 119-127.

[90] M. G. C. Resende and R. F. Werneck, "A Hybrid Heuristic for the p-Median Problem", *Journal of Heuristics* 10 (2004) 59–88.

[91] G. Reinelt, "TSPLIB: A travelling salesman problem library" *ORSA Journal on Computing* (1991) 3 376-384 *http://www.iwr.uni-heildelberg.de/groups/comopt/software/TSPLIB95/.*

[92] P. Rojeski and C. S. ReVelle, "Central facilities location under an investment constraint," *Geographical Analysis* 2 343-360.

[93] K.E. Rosing, C.S. ReVelle, and H. Rosing-Vogelaar, "The p-median and its linear programming relaxation: An approach to large problems," *Journal of the Operational Research Society* 30 (1979) 815-823.

[94] M. P. Scaparra and R. L. Church, "A bilevel mixed-integer program for critical infrastructure protection planning," *Computers and Operations Research* 35 (2008) 1905-1923.

[95] L. Schrage, "Implicit representation of variable upper bounds in linear programming," *Mathematical Programming Study* 4 (1975) 118-132.

[96] K. Spielberg, "Algorithms for the simple Plant-Location Problem with Some Side Conditions," *Operations Research* 17 (1969) 85-111.

[97] G. Stauffer, "The p-median polytope of Y-free graphs: an application of the matching theory," *Operations Research* 36 (2008) 351-354.

[98] A. Tamir, "An $O(pn^2)$ algorithm for the p-median and related problems on tree graphs," *Operations Research Letters* 19 (1996) 59-64.

[99] M. B. Teitz and P. Bart, "Heuristic methods for estimating the generalized vertex median of a weighted graph," *Operations Research* 16 (1968) 955-961.

[100] J. C. Texeira and A. P. Antunes, "A hierarchical location model for public facility planning," *European Journal of Operational Research* 185 (2008) 92-104.

[101] M. J. Todd, "Linear programming with variable upper bounds," *Mathematical Programming* 23 (1982) 34-49.

[102] J.D. Ullman, "Principles of Database and Knowledge-Base Systems, Volume I,II" Computer Science Press, Rockville, Maryland 1989

[103] A. Vigneron, Lixin Gao, M. J. Golin, G. F. Italiano and B. Li, "An algorithm for finding a k-median in a directed tree," *Information Processing Letters* 74 (2000) 81-88.

[104] J. L. Wagner and L. M. Falkson, "The optimal nodal location of public facilities with price-sensitive demand," *Geographical Analysis* 7 (1975) 69-83.

[105] R. Whitaker, "A fast algorithm for the greedy interchange of large-scale clustering and median location problems," *INFOR* 21 (1983) 95-108.

[106] P. Wolfe, "A method of conjugate subgradients for minimizing nondifferentiable functions," *Mathematical Programming Study* 3 (1975) 145-173.

[107] W. Zhao and M. E. Posner, "A large class of facets for the K-median polytope", *Mathematical Programming* 128 (2011) 171-203.