École Polytechnique Fédérale de Lausanne
Section de Mathématiques
ROSO

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Travail Pratique de Master

Résolution de problèmes de Steiner de grande taille pour la détection de fuites dans un réseau d'eau

Louis Pellaux

Professeurs : Thomas Liebling
Alain Prodon
Assistant : Christophe Weibel

hiver 2005-2006

Table des matières

Introduction

Dans les réseaux d'eau, il est absolument nécessaire de détecter toutes les fuites possibles ou du moins, le plus grand nombre, pour des raisons sécuritaires, pratiques et économiques. Le système utilisé pour repérer des fuites est le système de stations *Lorno*. Ces stations comprennent en particulier un hydrophone mesurant les ondes sonores, un système électronique enregistrant les données et une radio pour transmettre les données ou des alarmes. Malheureusement, le coût d'un tel système peut se révéler assez élevé et il n'est pas envisageable de munir tous les hydrantes de station *Lorno*. Ainsi, nous devons choisir les emplacements de bornes *Lorno* de manière à couvrir le plus efficacement le réseau d'eau. Cela nous conduit à résoudre un problème de recherche d'arborescence de Steiner. En particulier, nous souhaitons déterminer des arborescences de Steiner optimales pour les réseaux d'eau de la Ville de Lausanne. Nous utiliserons une formulation en Programme Linéaire que nous résoudrons à l'aide d'un algorithme *Branch & Cut*.

Chapitre 1

Problème de Steiner

Dans ce chapitre, nous donnons la définition du problème de Steiner et nous exposons brièvement le problème du *Prize-Collecting Steiner Tree (PCST)*.

1.1 Définition

Le problème d'arbre de Steiner de poids minimum se formule ainsi : soit $G = (V, E, c)$ un graphe non-orienté avec $c : E \to \mathbb{R}$ une fonction représentant les poids sur les arêtes. Nous voulons déterminer un arbre de poids minimum couvrant un sous-ensemble de sommets $U \subset V$. Hormis les cas $|U| = 2$ et $|U| = |V|$, ce problème est NP-complet. Dans le cas d'un graphe orienté, il s'agit de chercher une arborescence de poids minimum couvrant un sous-ensemble de sommets.

1.2 Prize-Collecting Steiner Tree Problem

Nous nous intéressons maintenant à un cas particulier de problème de Steiner : le *Prize-Collecting Steiner Tree Problem (PCST)*. Soit $G = (V, E, c, p)$ un graphe non-orienté avec une fonction de coût sur les arêtes, $c : E \to \mathbb{R}^+$, et une fonction de profits $p : V \to \mathbb{R}^+$ sur les sommets. Nous souhaitons trouver un sous-graphe connexe $T = (V_T, E_T)$ de G tel qu'il maximise le profit total :

$$profit(T) = \sum_{v \in V_T} p(v) - \sum_{e \in E_T} c(e) \tag{1.1}$$

La solution optimale T sera un arbre. En effet, si T contient un cycle C, nous ôtons l'arête la plus coûteuse du cycle e_C et considérons le sous-graphe

$T' = (V_{T'}, E_{T'})$ avec $V_{T'} = V_T$ et $E_{T'} = E_T - e_C$. Par construction, T' est connexe et nous avons profit(T') >profit(T) ! !

Une forme équivalente du (PCST) est celle de *Goemans et Williamson* : il s'agit de trouver un sous-graphe connexe $T = (V_T, E_T)$ de G qui minimise la fonction suivante :

$$GW(T) = \sum_{v \notin V_T} p(v) + \sum_{e \in E_T} c(e) \tag{1.2}$$

La fonction $p(v)$ représente dans ce cas une pénalité pour ne pas appartenir à la solution T. Cette formulation (1.2) est préférable à (1.1), car nous savons qu'il existe plusieurs algorithmes d'approximation de l'optimum, contrairement à la formulation (1.1) dont l'approximation de l'optimum est un problème NP-dur (cf. [2]).

Dans le cadre de ce projet, nous aurons à résoudre un*Rooted-Prize-Collecting Steiner Tree Problem (RPCST)*, une version particulière de (PCST), où nous imposons qu'un sommet source $r \in V$ fasse partie de chaque solution admissible T. De plus, nous traiterons du cas orienté : le sommet $r \in V$ sera la racine de notre arborescence.

1.3 Programme linéaire

Afin de résoudre le (RPCST), nous formulons ce problème avec un Programme Linéaire. Il existe plusieurs formulations en PL (cf.[2]). Celle que nous retenons dans le cadre de ce projet, est la *Formulation par Coupes*, se concentrant sur la connexité de la solution : chaque coupe séparant un sommet appartenant à la solution de la racine r doit avoir l'un de ses arcs dans la solution. Nous présentons ci-dessous la formulation en question :

$$min \sum_{(i,j) \in E} c_{ij} x_{ij} + \sum_{i \in V} p_i \tag{1.3}$$

$$s.c \sum_{(j,i) \in E} x_{ji} = y_i \qquad \forall i \in V - \{r\} \tag{1.4}$$

$$x(\delta^-(S)) \geq y_k \qquad k \in S, r \notin S, \forall S \subset V \tag{1.5}$$

$$\sum_{(r,i) \in E} x_{ri} = 1 \tag{1.6}$$

$$x_{ij},\ y_i \in \{0,1\} \qquad \forall i \in V - \{r\},\ \forall (i,j) \in E \tag{1.7}$$

Nous avons introduit ici une notation concernant les coupes : un sous-ensemble de sommets $S \subset V$ et son complément $\overline{S} = V - S$ induisent deux coupes : $\delta^+(S) = \{(i,j)|i \in S, j \in \overline{S}\}$ et $\delta^-(S) = \{(i,j)|i \in \overline{S}, j \in S\}$. Nous écrivons $x(A) = \sum_{(i,j)\in A} x_{ij}$ pour $A \subset E$ un sous-ensemble d'arcs. Les variables du PL sont : $x_{ij} = \begin{cases} 0 & (i,j) \notin E_T \\ 1 & (i,j) \in E_T \end{cases}$ représentant les arcs de la solution T et $y_i = \begin{cases} 0 & i \notin V_T \\ 1 & i \in V_T \end{cases}$ représentant les sommets. Nous notons dans la fonction objective que les variables y_i n'interviennent pas et que la fonction $p : V \to \mathbb{R}$ n'est qu'une constante. En effet, dans le (RPCST), nous avons reporté les profits des sommets sur le coût des arcs c_{ij}. Les contraintes (1.4) nous assure que pour chaque sommet, différent de la racine, appartenant à la solution, il y a un arc entrant exactement. La contrainte (1.6) nous dit qu'il y a un arc sortant exactement de la racine r. Les contraintes (1.7) sont les *contraintes d'intégralité* des variables et enfin, les *contraintes de coupe* (1.5), dont nous avons déjà parlé plus haut. Après avoir formulé un programme linéaire (PL), nous pouvons passer à sa résolution effective, ce sera l'objet du chapitre suivant.

Chapitre 2

Modélisation

Dans ce chapitre, nous détaillerons, entre autres, l'algorithme *Branch & Cut* qui est la clé de voûte de tout notre programme. Auparavant, nous exposons la modélisation d'un réseau d'eau et décrivons le système de détection de fuites : le système LORNO.

2.1 Qu'est-ce que le système LORNO ?

Le système LORNO est un système de contrôle pour la détection de fuites dans un réseau d'eau. Il est composé de stations installées sur des hydrantes. Chaque station est équipée d'un *hydrophone*, appareil mesurant une fois par heure les ondes sonores dans les tuyaux. Ces données sont enregistrées et comparées avec un spectre de référence. Ainsi la station peut suivre l'évolution du bruit dans les tuyaux et détecter, après un certain temps, une fuite. Dans ce cas, la station transmettra par radio aux stations voisines un message d'alarme, qui sera transmis jusqu'à une centrale de contrôle. Il est aussi possible de n'installer qu'un relais radio sur un hydrante, qui n'aura pour but que de transmettre les données des stations voisines.

2.2 Modélisation d'un réseau d'eau

Dans cette section, nous définissons les graphes qui modélisent un réseau d'eau. En voici un exemple :

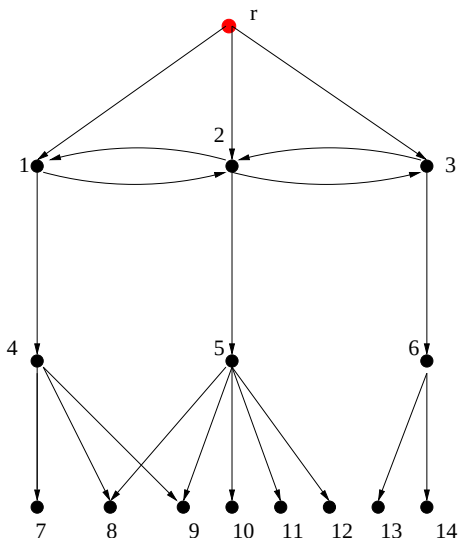

FIG. 2.1 – Exemple de graphe d'un réseau d'eau

Le graphe $G = (V, E, c)$ est un graphe orienté avec r comme racine . Elle désigne la centrale récoltant les données transmises par les hydrantes. Les sommets $1, 2, 3$ représentent les hydrantes "en surface", nous noterons cet ensemble $\overline{H}$. Chaque hydrante $\overline{h_i}$ est relié á la centrale, symbolisé par un arc $(r, \overline{h_i})$. Si l'un de ces sommets fait partie de la solution, cela signifie que l'hydrante sera muni d'un relais radio (au minimum). Les sommets $4, 5, 6$, quant à eux, sont les hydrantes "en sous-sol". L'ensemble de ces sommets sera noté $\underline{H}$. Si l'un de ces sommets appartient à la solution, cela voudra dire que l'hydrante possédera une station complète. Enfin, les sommets "terminaux" (ici $7, 8, ..., 14$) décrivent les régions écoutables par les hydrantes. Nous noterons cet ensemble R.
Nous classons les arcs en quatre catégories :

- les arcs $(r, \overline{h_i})$ avec $\overline{h_i} \in \overline{H}$
- les arcs $(\overline{h_i}, \overline{h_j})$, avec $\overline{h_i}, \overline{h_j} \in \overline{H}$ représentent les liaisons radios entre les hydrantes.
- les arcs $(\overline{h_i}, \underline{h_i})$, avec $\overline{h_i} \in \overline{H}$, $\underline{h_i} \in \underline{H}$
- les arcs $(\underline{h_i}, r_k)$ avec $\underline{h_i} \in \underline{H}$, $r_k \in R$ signifient que l'hydrante $\underline{h_i}$ peut écouter la région r_k.

Les arcs $(r, \overline{h_i})$ sont de coût nul. Dans notre problème, nous autorisons qu'un arc de première catégorie à intégrer la solution, de sorte que nous obtenons bien une arborescence si nous retranchons la racine r de la solution. Le coût d'un arc de la deuxième catégorie est le coût d'installation d'une radio sur

un hydrante, tandis que le coût d'un arc de la troisième catégorie décrit tous les frais inhérents à l'installation complète d'une station LORNO. Nous supposons que le coût d'installation partielle ou complète d'une station est le même pour chaque hydrante. Enfin, les coûts des arcs de la dernière catégorie sont les bénéfices engendrés par la région écoutée : chaque arc entrant dans un même sommet région a le même bénéfice. Nous notons que le coût d'un tel arc est négatif.

2.3 Algorithme *Branch & Cut*

Dans le chapitre précédent, nous avons donné la formulation du PL du (RPCST). Pour résoudre ce Programme Linéaire, nous utilisons un algorithme *Branch & Cut* :

$$min \sum_{(i,j)\in E} c_{ij}x_{ij} + \sum_{i\in V} p_i \tag{2.1}$$

$$s.c \sum_{(j,i)\in E} x_{ji} = y_i \qquad \forall i \in V - \{r\} \tag{2.2}$$

$$\sum_{(r,i)\in E} x_{ri} = 1 \tag{2.3}$$

$$\sum_{(j,i)\in E} x_{ji} \leq \sum_{(i,j)\in E} x_{ij} \qquad \forall i \notin R \cup \{r\} \tag{2.4}$$

$$y_i \leq 1 - x_{rj} \qquad \forall i < j, \{i,j\} \subset \overline{H} \tag{2.5}$$

$$0 \leq x_{ij},\ y_i \leq 1 \qquad \forall i \in V - \{r\},\ \forall (i,j) \in E \tag{2.6}$$

Nous commençons par résoudre une relaxation du PL de la formulation par coupes sans les contraintes (1.5). Nous ajoutons à la relaxation du PL les contraintes (2.4) et (2.5) pour renforcer le problème. A cause de la struture d'arborescence de la solution, il est évident que pour chaque sommet non-terminal, le *in-degré*, i.e. le nombre d'arcs entrants, est plus petit ou égal au *out-degré*, i.e. le nombre d'arcs sortants. Ainsi, la contrainte (2.4) exprime cette condition. Quant à la contrainte (2.5), la *contrainte d'asymétrie*, elle assure que l'arc sortant de la racine r entre dans le sommet de plus petit index (appartenant à la solution). Cela a pour but d'éliminer les symétries du PL. Pour plus de précisions, nous renvoyons le lecteur à [1] et [2].

La solution obtenue de ce PL ne sera évidemment pas connexe. Nous utilisons donc un algorithme de flots maximum de Goldberg pour déterminer les contraintes de coupes à ajouter :

ALGORITHME 1
Donnée : le graphe de la solution courante $G_s = (V, E, x)$
Résultat : un ensemble de contraintes de coupes à ajouter dans le PL.

1. Pour $i \in R$, $y_i > 0$ faire :
   (a) Poser $x^{'} = x + EPS$.
   (b) Répéter :
      i. Poser $f = MaxFlow(G, x^{'}, r, i, S_r, S_i)$.
      ii. Trouver $\delta^+(S_r)$ t.q. $x^{'}(\delta^+(S_r)) = f$, $r \in S_r$.
      iii. Si $f < y_i$ alors :
         A. Insérer la contrainte $x(\delta^+(S_r)) \geq y_i$ dans le PL.
         B. Poser $x^{'}_{ij} = 1$, $\forall (i,j) \in \delta^+(S_r)$.
      iv. Trouver $\delta^-(S_i)$ t.q. $x^{'}(\delta^-(S_i)) = f$, $i \in S_i$.
      v. Si $S_i \neq V - S_r$ alors :
         A. Insérer la contrainte $x(\delta^-(S_i)) \geq y_i$ dans le PL.
         B. Poser $x^{'}_{ij} = 1$, $\forall (i,j) \in \delta^-(S_i)$.
   vi. Arrêter lorsque $f \geq y_i$ ou que $MAXCUTS$ est atteint

Pour chaque paire de sommets (r, i) avec $i \in R$, $y_i > 0$, l'algorithme retourne la valeur maximale du flot f et deux ensembles :

$S_r \subset V$, avec $r \in S_r$ et tel que $x(\delta^+(S_r)) = f$.

$S_i \subset V$, avec $i \in S_i$ et tel que $x(\delta^-(S_i)) = f$

Si $f < y_i$, nous insérons la coupe violée $x(\delta^+(S_r)) \geq y_i$ dans le PL relaxé. Ensuite, nous ajoutons itérativement les contraintes violées induites par la coupe min. (r, i) du graphe dans lequel on pose que chaque arc $(u, v) \in \delta^+(S_r)$ a une capacité de 1, i.e. $x_{ij} = 1$. Nous répetons ce processus aussi longtemps qu'il y a de telles contraintes ou que le nombre maximal $MAXCUTS$ n'est pas atteint. Après avoir ajouté ces contraintes, nous résolvons de nouveau le PL. Dès que l'algorithme de flots n'ajoute plus aucune nouvelle coupe, nous avons obtenu une solution optimale. Il reste à voir si la solution est entière ou non. Dans le cas où la solution est fractionnaire : nous faisons du Branch & Bound pour avoir une solution entière. Ainsi, nous avons décrit dans le détail notre algorithme *Branch & Cut*.

Chapitre 3

Méthodes utilisées

Après avoir exposé l'algorithme *Branch & Cut*, nous donnons dans ce chapitre les méthodes utilisées pour améliorer l'efficacité de cet algorithme.

3.1 Heuristique utilisée pour le calcul de bornes

Avant de donner nos méthodes permettant d'améliorer la rapidité de l'algorithme *Branch & Cut* et afin d'avoir une borne supérieure pour chaque graphe, nous présentons une heuristique simple pour trouver une arborescence remplissant les conditions de notre problème :

ALGORITHME 2
Donnée : le graphe de départ $G = (V, E, c)$
Résultat : une arborescence non-optimale

1. Pour $i \in \underline{H}$ faire :
   (a) Poser $s = 0$.
   (b) Pour chaque arc $(i, j) \in E$, $j \in R$: poser $s = s + c_{ij}$.
   (c) Poser $\phi_i = s$.
2. Chercher $m \in \underline{H}$ tel que $\phi_m = \min_{i \in \underline{H}} \{\phi_i\}$.
3. Poser $k = 0$, $k' = 0$, Liste[k]$= m'$, où $m' \in overline{H}$, image de m.
4. Tant que Liste[k]$\neq 0$, faire :
5. poser $s =$Liste[k].
6. si $(s, t) \in E$ avec t non-pris dans Liste :
7. poser Liste[k']=t.

8. retenir le prédecesseur de t.

9. retenir le rang topologique de t.

10. poser $k^{'} = k^{'} + 1$.

11. poser $k = k + 1$.

12. Trier les sommets de la liste selon ϕ.

13. Parcourir Liste :

14. i =sommet actuel.

15. si $\phi_i + c_r + c_l < 0$; faire :

16. s'assurer que son prédecesseur fait partie de l'arborescence.

17. insérer s et ses sommets terminaux non-pris dans l'arbo.

18. mettre à jour ϕ.

Pour chaque hydrante en surface $i \in \underline{H}$, nous considérons tous les sommets terminaux adjacents et nous construisons la fonction ϕ où, $\phi_i = \sum_{(i,j)\in E} c_{ij}$ avec $i \in \underline{H}$, $j \in R$. Nous cherchons ensuite l'hydrante en surface i' tel que son "image" en sous-sol i, avec ϕ_i minimum. Nous construisons la composante connexe des hydrantes en surface . Nous trions ensuite les hydrantes d'après leurs valeurs de ϕ, dans l'ordre croissant. Dès lors, nous n'avons plus qu'à construire l'arborescence, en prenant les hydrantes et les sommets terminaux adjacents non-pris dans l'ordre défini précédemment, en nous assurant du gain apporté par les sommets terminaux.

3.2 Qu'est-ce qu'une contrainte de coupe ?

Dans le chapitre précédent, nous avons cité à plusieurs reprises les notions de contraintes de coupe, qui sont celles qui garantissent la connexité de la solution. Par la suite, nous discuterons des méthodes qui ajoutent au PL initial des contraintes de coupe. Donc, dans un souci de clarté, nous illustrons ici avec un exemple la notion de *contrainte de coupe*. Nous reprenons le graphe (2.1) :

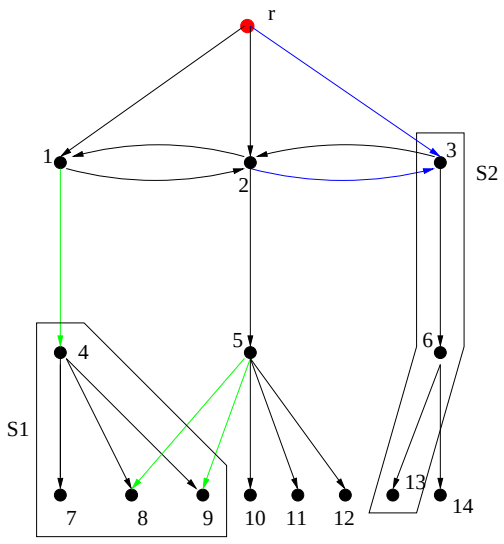

FIG. 3.1 – Exemple de graphe et de coupes

Soit deux sous-ensembles de sommets $S_1, S_2 \subset V$, où $S_1 = \{4, 7, 8, 9\}$ et $S_2 = \{3, 6, 13\}$. Les arcs entrant en S_1 sont $(1, 4), (5, 8)$ et $(5, 9)$. Les arcs entrant en S_2 sont $(r, 3), (2, 3)$. Ainsi, les *contraintes de coupe* définies par S_1 sont :

$$- y_4 + x_{14} + x_{58} + x_{59} \geq 0 \tag{3.1}$$
$$- y_7 + x_{14} + x_{58} + x_{59} \geq 0 \tag{3.2}$$
$$- y_8 + x_{14} + x_{58} + x_{59} \geq 0 \tag{3.3}$$
$$- y_9 + x_{14} + x_{58} + x_{59} \geq 0 \tag{3.4}$$
$$\tag{3.5}$$

Quant aux contraintes de coupe engendrées par S_2, nous avons :

$$- y_3 + x_{r3} + x_{23} \geq 0 \tag{3.6}$$
$$- y_6 + x_{r3} + x_{23} \geq 0 \tag{3.7}$$
$$- y_{13} + x_{r3} + x_{23} \geq 0 \tag{3.8}$$
$$\tag{3.9}$$

Le nombre de contraintes ajoutées au PL est donc $|S_1| + |S_2|$. Nous remarquons, pour terminer, qu'il n'y a pas lieu de considérer le cas des sous-ensembles $S = \{t\}$, où $t \in R$ un sommet terminal, car les contraintes de type (1.4) répondent déjà à ces cas-là.

3.3 Première méthode d'ajout de contraintes de coupe

Dans le PL relaxé initial, nous définissons les contraintes de coupe à ajouter de la façon suivante : pour chaque sommet $\underline{h_i} \in \underline{H}$, nous associons un sommet terminal adjacent r_i. Nous construisons ensuite le sous-ensemble $S_i = \{\underline{h_i}, r_i\} \subset V$, $\forall i = 1, ..., |H|$. Le choix du sommet-région est fixé par l'ordre dans lequel est indicé le graphe de départ.

3.4 Deuxième méthode

Nous proposons ici une deuxième façon d'ajouter des contraintes de coupes. Elle ressemble à la première méthode, mais cette fois nous considérons tous les sommets terminaux comme un seul bloc : pour chaque sommet-hydrante $\underline{h_i} \in \underline{H}$, nous associons le sous-ensemble de sommets $S_i \subset V$ défini par $S_i = \{\underline{h_i}, r_{i_1}, r_{i_2}, \cdots, r_{i_k}\}$, où $(\underline{h_i}, r_{i_j}) \in E$, $\forall r_{i_j} \in R$, $\forall j = 1, ..., k$.

3.5 Troisième méthode

Une autre manière d'ajouter des contraintes de coupe, est décrite ainsi : pour chaque sommet-région $r_i \in R$, $\forall i = 1, \cdots, |R|$, nous lui associons tous les sommets $\underline{h_i} \in \underline{H}$ adjacents. Nous construisons ainsi les sous-ensembles $S_i = \{r_i, \underline{h_{i_1}}, \underline{h_{i_2}}, ..., \underline{h_{i_k}}\}$, $\forall i = 1, ..., |R|$

3.6 Quatrième méthode

La quatrième méthode reprend les sous-ensembles de sommets décrits par la méthode 3 avec en supplément, pour chaque sommet terminal t, le sous-ensemble contenant tous les sommets hydrantes adjacents et leurs "prédécesseurs" dans $\overline{H}$, autrement dit : $S_t = \{t, \underline{h_{i_1}}, ..., \underline{h_{i_k}}, \overline{h_{i_1}}, ..., \overline{h_{i_k}}\}$.

L'avantage de ces contraintes de coupe, pour les méthodes 3 et 4 est qu'aucun arc de type $(\underline{h}, t)$, où $\underline{h} \in \underline{H}$, $t \in R$ ne constitue un arc entrant

pour les sous-ensembles S_t. De cette manière, nous espérons connecter plus rapidement le graphe.

3.7 Intérêt de mettre des contraintes de coupe

Nous avons présenté des méthodes pour ajouter dans le PL initial des *contraintes de coupe*. Toutefois, nous n'avons pas encore expliqué les raisons de ces ajouts. Nous voulons simplement améliorer l'efficacité de l'algorithme *Branch & Cut*. Pour illustrer nos propos, nous avons reproduit les solutions trouvées aux diverses étapes par le programme, dans un premier temps, sans aucune *contrainte de coupe* et, dans un second temps, avec des *contraintes de coupe*. Sur ce petit graphe, nous constatons déjà une diminution du nombre d'étapes nécessaires à l'obtention de la solution optimale :

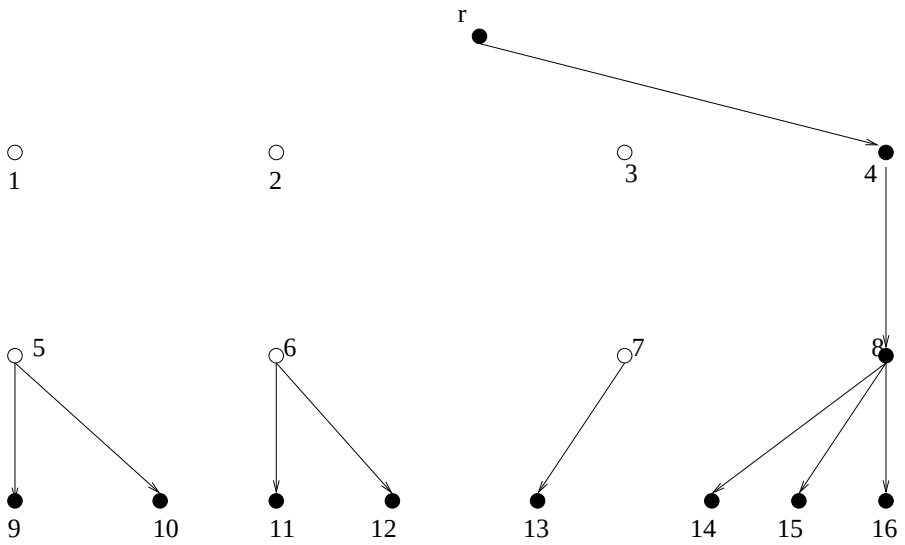

FIG. 3.2 – Etape 1

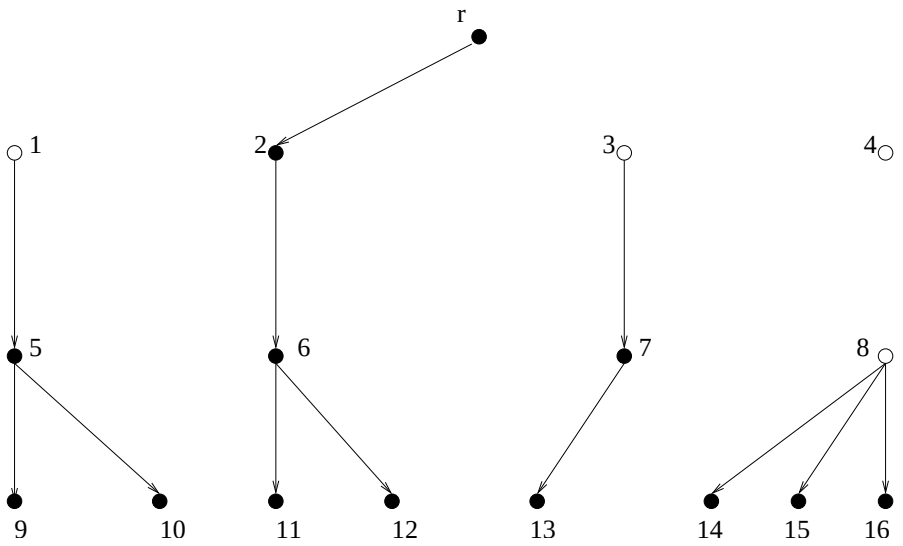

FIG. 3.3 – Etape 2

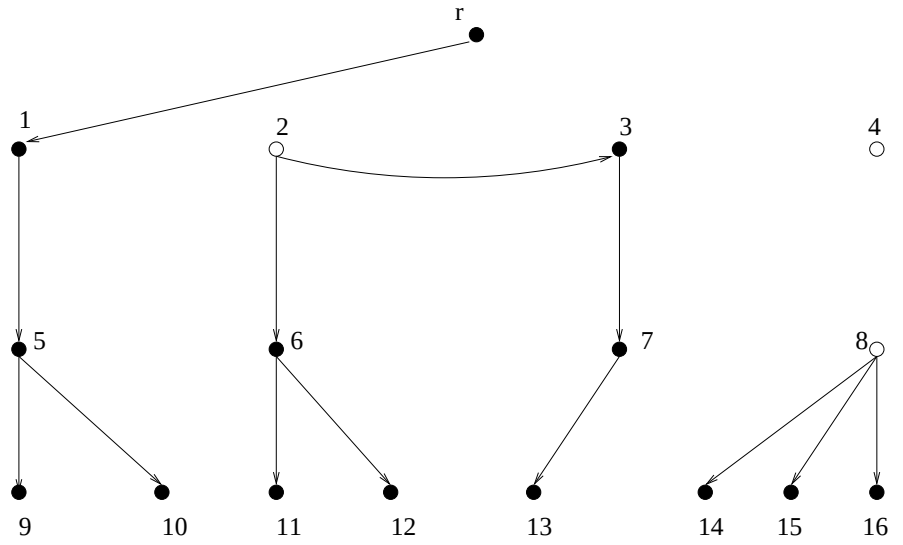

FIG. 3.4 – Etape 3

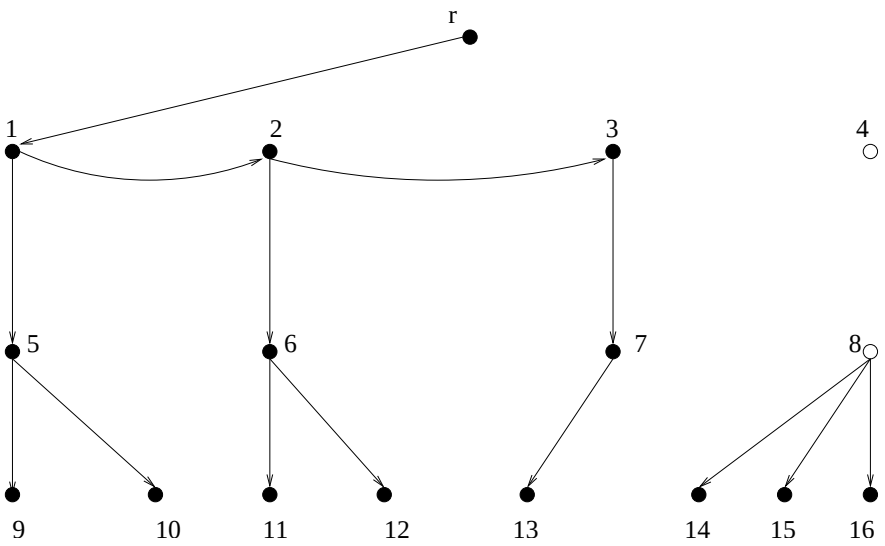

FIG. 3.5 – Etape 4

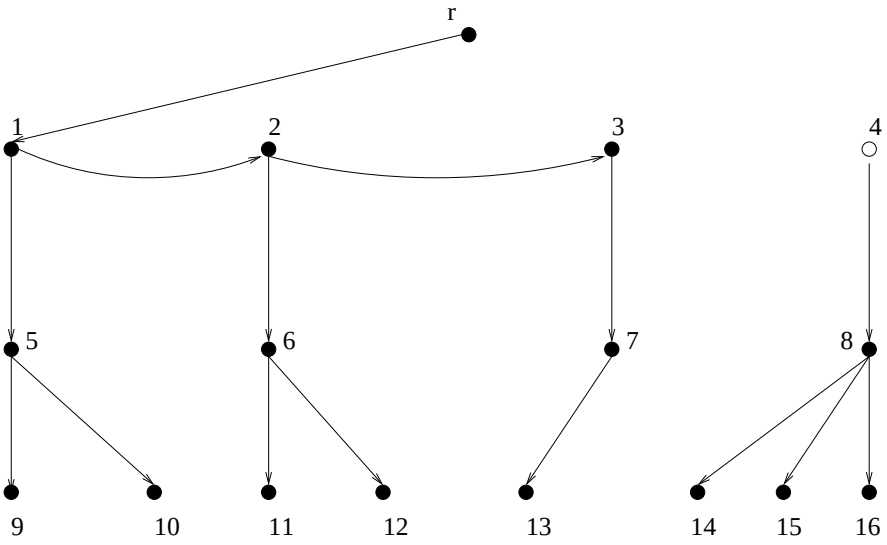

FIG. 3.6 – Etape 5

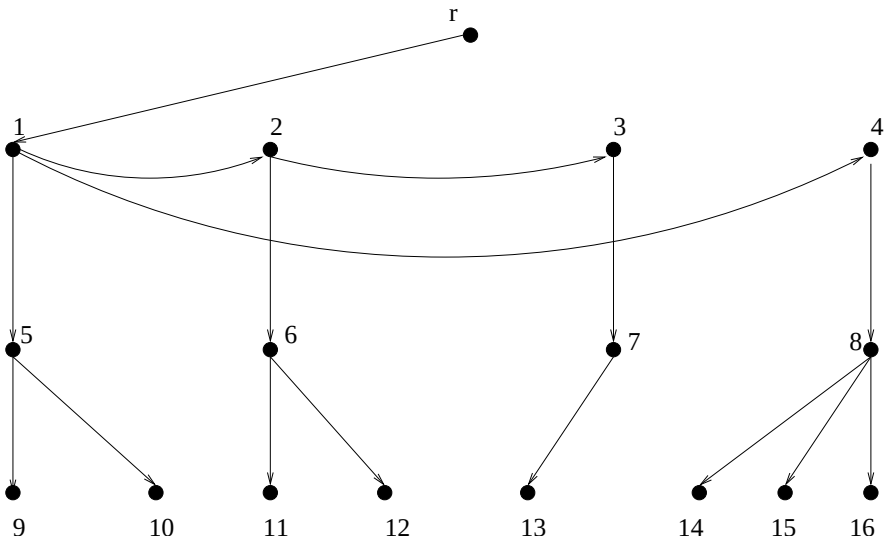

FIG. 3.7 – Etape 6

Pour le programme sans les *contraintes de coupe* ; six étapes auront été nécessaires. Voici maintenant les étapes de la construction de la solution optimale en ayant ajouté des contraintes de coupe dans le PL initial selon la méthode 3.

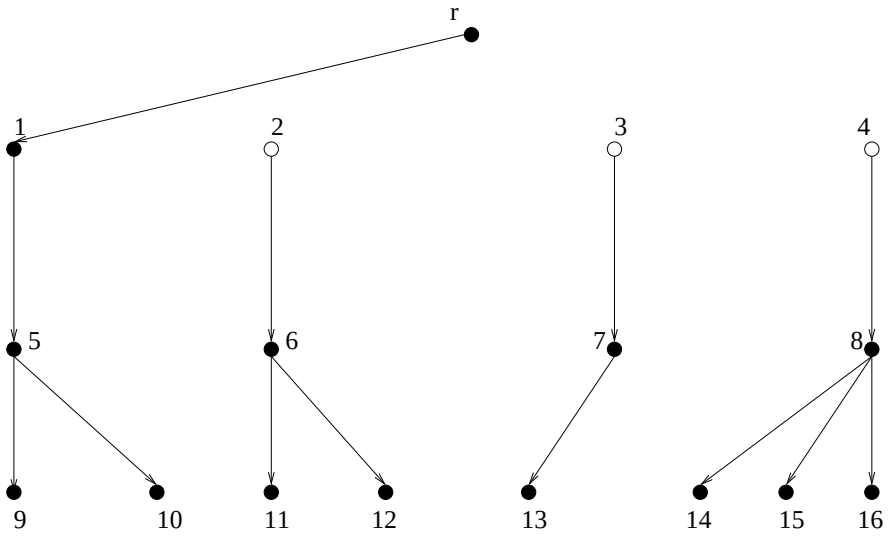

FIG. 3.8 – Etape 1

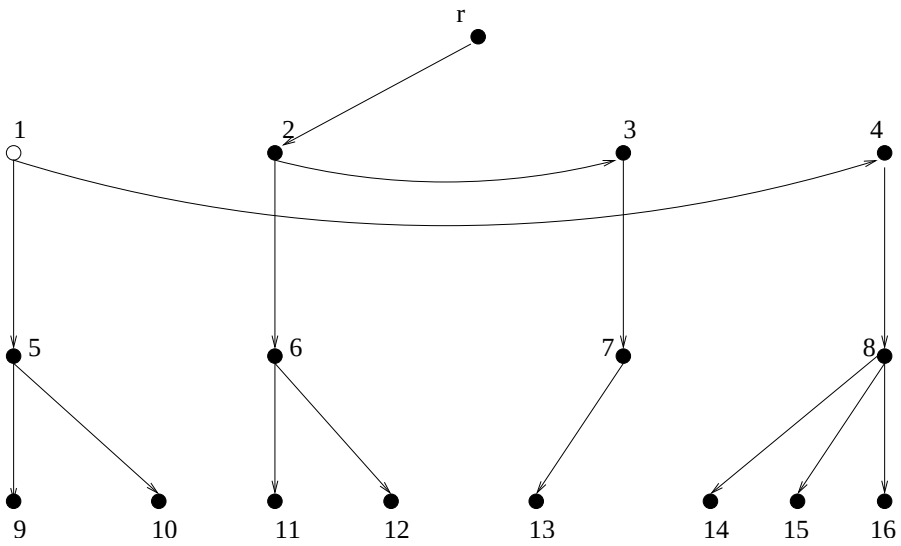

FIG. 3.9 – Etape 2

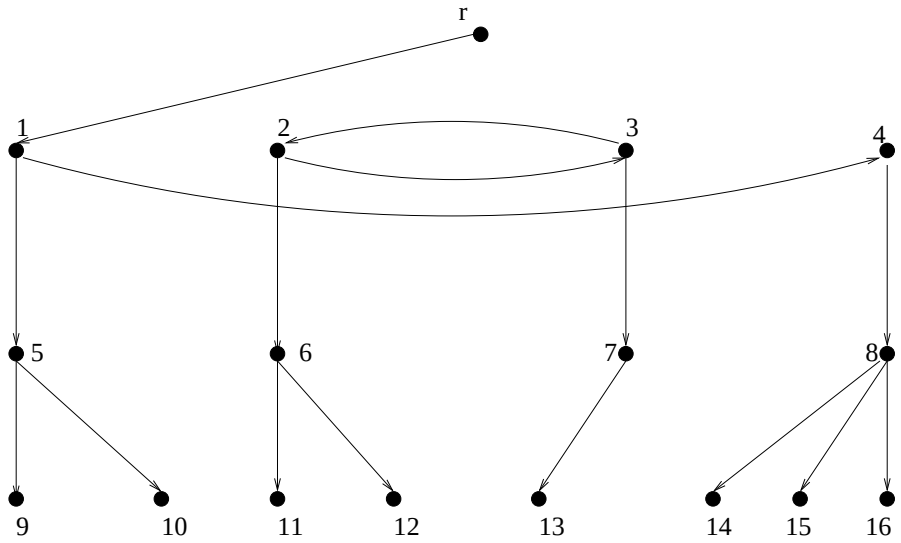

FIG. 3.10 – Etape 3

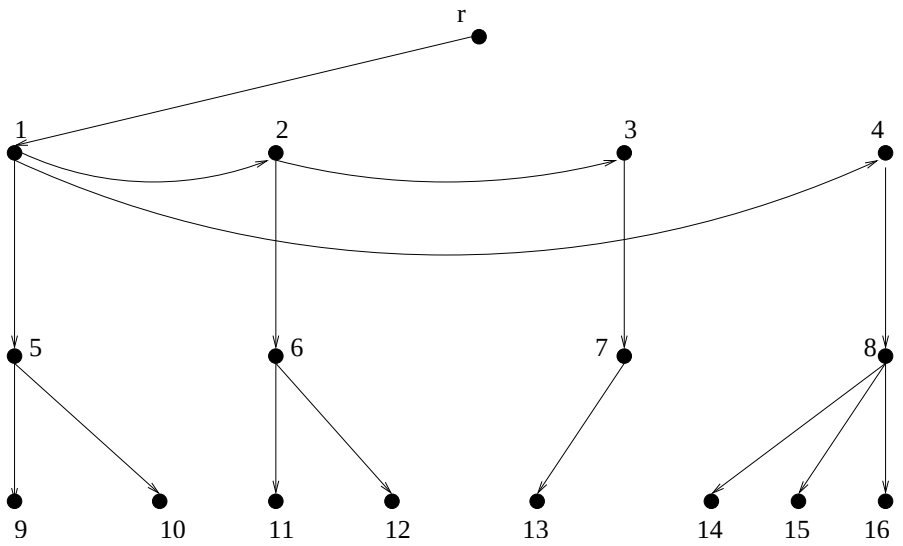

FIG. 3.11 – Etape 4

Dans le deuxième cas, quatre étapes suffisent. Nous notons que la solution trouvée est la même que dans le premier cas. Toutefois, comme les coûts des arcs entrant dans un même sommet terminal sont identiques, il est tout à fait possible d'avoir, pour un graphe, des solutions optimales différentes.

3.8 D'autres méthodes

Nous citons dans ce paragraphe d'autres méthodes que nous avons essayées au cours de ce projet. Elles avaient pour but de d'améliorer la rapidité de résolution, sans garantir pour autant l'optimalité de la solution trouvée :

- méthode 1 : poser dans le PL initial $y_i = 1$, pour certains $i \in R$.
- méthode 2 : poser dans le PL initial $y_i = 0$, pour certains $i \in R$.

La premiére méthode s'apparente à la recherche "standard" d'une arborescence. La deuxième méthode force la recherche de la solution dans une partie du graphe donné. En effet, en posant la condition $y_i = 0$, $i \in R$ et, avec la contrainte (2.2), nous obligeons les variables x_{ji} à être nulles. Ensuite, à cause de la contrainte (2.4), cela se répercute sur les variables x_{kj}.

Dans les deux cas, les résultats n'ont guère été satisfaisants. Nous avons certes amélioré la rapidité de la résolution du PL pour les petits graphes. Cependant, ces méthodes n'ont eu aucun effet sur les grands graphes.

3.9 Graphes utilisés

Dans ce projet, nous nous sommes intéressés en particulier à trois graphes, qui représentent chacun un réseau d'eau de la Ville de Lausanne. Nous donnons ci-dessous les caractéristiques de ces réseaux :

Réseau	Nombre de sommets	Nombre d'arcs	Nombre d'hydrantes
Vernand	116	389	34
Dailles	623	3186	173
Haute-Pierre	2257	10392	606

TAB. 3.1 – Réseaux d'eau de la Ville de Lausanne

Pour une meilleure visualisation de ces réseaux : nous avons dessiné les trois réseaux à l'aide du programme élaboré en Matlab dans [1] :

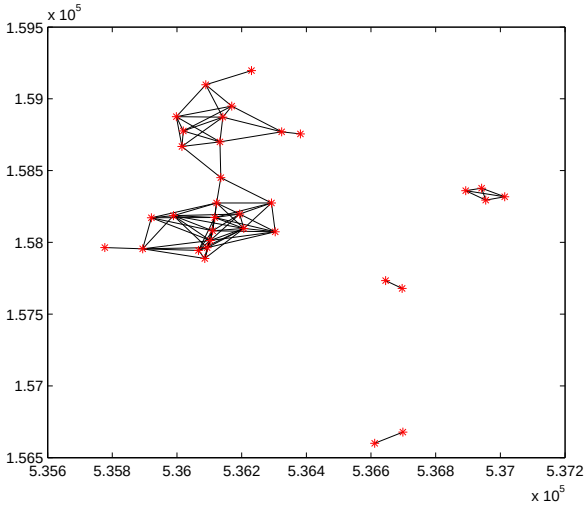

FIG. 3.12 – Réseau de communication en surface de Vernand

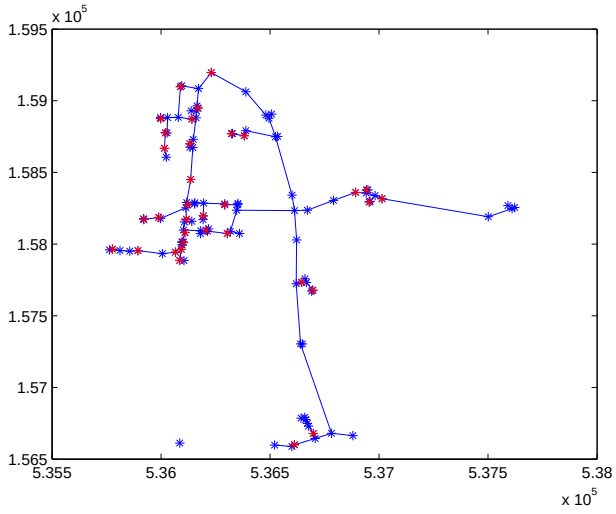

FIG. 3.13 – Réseau de Vernand en sous-sol

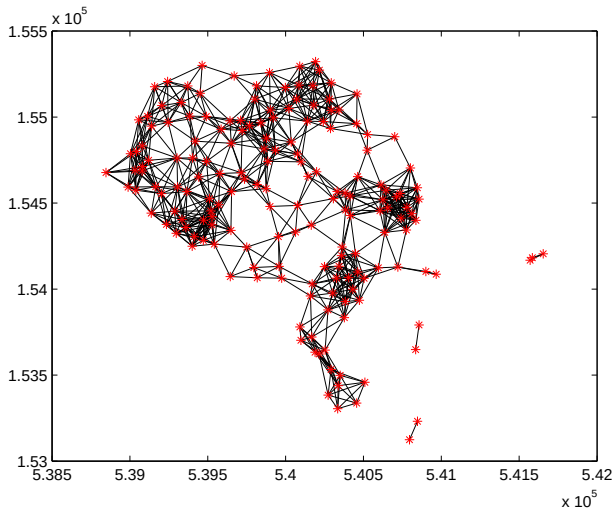

FIG. 3.14 – Réseau de communication en surface de Dailles

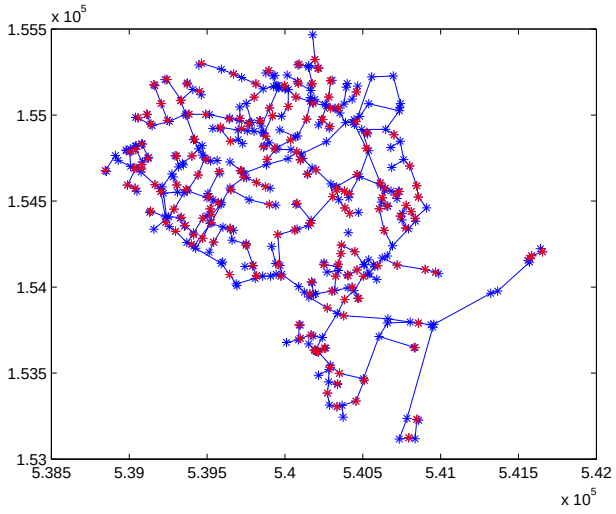

FIG. 3.15 – Réseau de Dailles en sous-sol

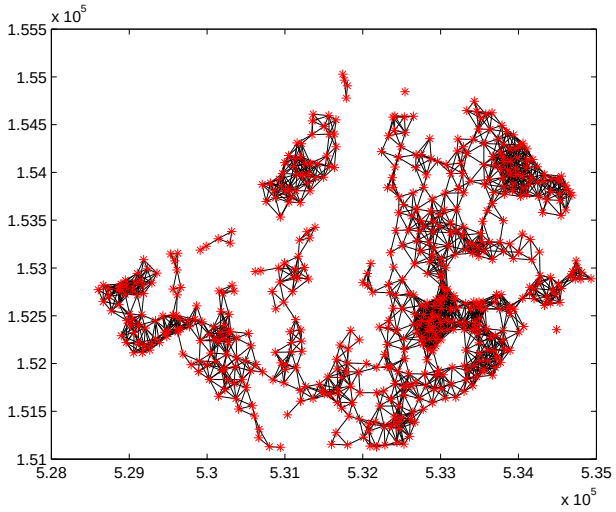

FIG. 3.16 – Réseau de communication en surface de Haute-Pierre

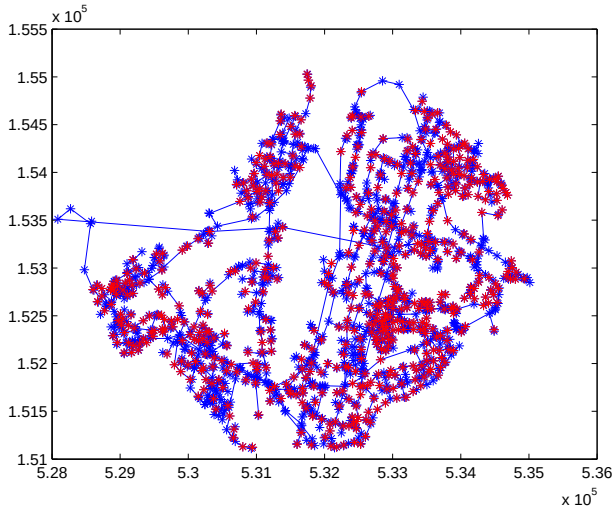

FIG. 3.17 – Réseau de Haute-Pierre en sous-sol

Les régions écoutées sont représentées sur les figures (3.13), (3.15) et (3.17) par des arêtes : il s'agit en fait de tuyaux écoutés. Pour plus de détails, nous renvoyons le lecteur à [1]. A la vue de ces schémas, nous constatons que l'ensemble des hydrantes en surface $\overline{H}$ n'engendre pas un graphe connexe. C'est la raison pour laquelle, nous avons aussi construit, à partir de ces réseaux, les graphes suivants que nous utiliserons pour des comparaisons :

Graphe	Nombre de sommets	Nombre d'arcs	Nombre d'hydrantes
$\text{vern}_{splitte}$	94	345	26
$\text{dail}_{splitte}$	604	3154	166
$\text{hpie}_{splitte}$	2049	9588	551

TAB. 3.2 – Partie connexe des réseaux d'eau

Les graphes $vern_{splitte}$, $dail_{splitte}$, $hpie_{splitte}$ reprsésentent respectivement la plus grande partie connexe des réseaux de Vernand, Dailles et Haute-Pierre. Dans le chapitre suivant, nous testons ces graphes et nous tenterons d'obtenir les solutions optimales pour les réseaux de la Ville de Lausanne.

Chapitre 4

Résultats

Dans ce chapitre, nous exposons les résultats trouvés par les méthodes vues précédemment pour les réseaux d'eau de la Ville de Lausanne. Nous commenterons aussi l'efficacité de ces méthodes.

4.1 Récapitulatif

Tous les tests ont été faits sur un CPU 3,7Ghz. Nous employons la librairie COIN-OR, en particulier les solveurs Clp et Cpx, pour la résolution. Nous donnons maintenant les résultats pour le réseau de Vernand avec les solveurs Clp et Cpx (CPLEX) :

Méthode	Val. opt.	Temps en [s]	Nb d'étape	Nb total.contr.
Sans contraintes (avec Clp)	-328108	$83,75$	108	4420
Sans contraintes (avec CPLEX)	-328108	$23,98$	391	5365
Méthode 1 (avec Clp)	-328108	$516,34$	176	7004
Méthode 1 (avec CPLEX)	-328108	$14,22$	234	3690
Méthode 2 (avec Clp)	-328108	$415,49$	195	8269
Méthode 2 (avec CPLEX)	-328108	$14,25$	223	3295
Méthode 3 (avec Clp)	-328108	$208,32$	144	5739
Méthode 3 (avec CPLEX)	-328108	$5,86$	131	2803
Méthode 4 (avec Clp)	-328108	$111,5$	73	4157
Méthode 4 (avec CPLEX)	-328108	$5,88$	159	2848

TAB. 4.1 – Résultats pour le réseau de Vernand

Nous présentons maintenant les résultats pour le graphe $\text{vern}_{splitte}$:

Méthode	Val. opt.	Temps CPU	Nb d'étape	Nb total.contr.
Sans contraintes (Clp)	−328108	3	56	1917
Sans contraintes (Cpx)	−328108	1,02	79	1282
Méthode 1 (Clp)	−328108	4,02	62	2081
Méthode 1 (Cpx)	−328108	1,06	90	1257
Méthode 2 (Clp)	−328108	2,87	53	1993
Méthode 2 (Cpx)	−328108	0,99	76	1277
Méthode 3 (Clp)	−328108	2,1	45	1841
Méthode 3 (Cpx)	−328108	0,82	78	1145
Méthode 4 (Clp)	−328108	0,84	16	1393
Méthode 4 (Cpx)	−328108	0,35	22	1205

Nous n'avons pas cité les résultats obtenus pour les réseaux de Dailles et Haute-Pierre : mise à part la méthode 4 avec CPLEX, aucune méthode n'est arrivée à terme, même après plusieurs jours de calculs.

4.2 Utilisation de graphes complets

Faisant face à de multiples difficultés pour obtenir des résultats convaincants pour les grands graphes, nous avons regardé plus attentivement ce qu'il se passe dans des graphes où tous les hydrantes en surface sont tous reliés entre eux, i.e le sous-graphe $G_{\overline{H}}$ est complet. Donc, nous avons "complété" les graphes Vernand, Dailles et Haute-Pierre et nous avons testé avec la méthode 4 : les contraintes de coupe que nous ajoutons au PL initial nous garantissent la valeur de la solution optimale dès la première étape. De plus, la solution obtenue après chaque étape doit être entière. En effet, étant donné que nous avons $G_{\overline{H}}$ complet, nous devons juste trouver les sommets hydrantes $\underline{h_i}$ qui rapportent le plus et insérer leurs images $\overline{h_i}$.

Nous avons ensuite testé la méthode 4 pour le graphe $complet_{vern}$. Les résultats obtenus par le solveur Clp ont surpris : contrairement à nos prévisions, durant la résolution, il a obtenu des solutions fractionnaires de valeurs optimales. Elles étaient la combinaison convexe de deux solutions optimales. Nous avons aussi testé ce graphe avec CPLEX (solveur Cpx), qui n'a pas retourné de solutions fractionnaires.

Graphe	Nombre de sommets	Nombre d'arcs	Nombre d'hydrantes
complet_{vern}	116	1329	34
complet_{dail}	623	31082	173
complet_{hpie}	2257	371054	606

TAB. 4.3 – graphes "complets"

4.3 Bornes supérieures pour les réseaux

Dans cette section, nous donnons les résultats trouvés par notre heuristique pour les trois réseaux de la Ville de Lausanne :

Graphe	Poids de l'arborescence
Vernand	−319948
Dailles	−2237386
Haute-Pierre	−8209024

TAB. 4.4 – Résultats de l'heuristique

Nous illustrons à présent les arborescences obtenues par l'heuristique :

les sommets hydrantes munis d'une station complète sont marqués par un point et les régions écoutées sont symbolisées par les arêtes en rouge :

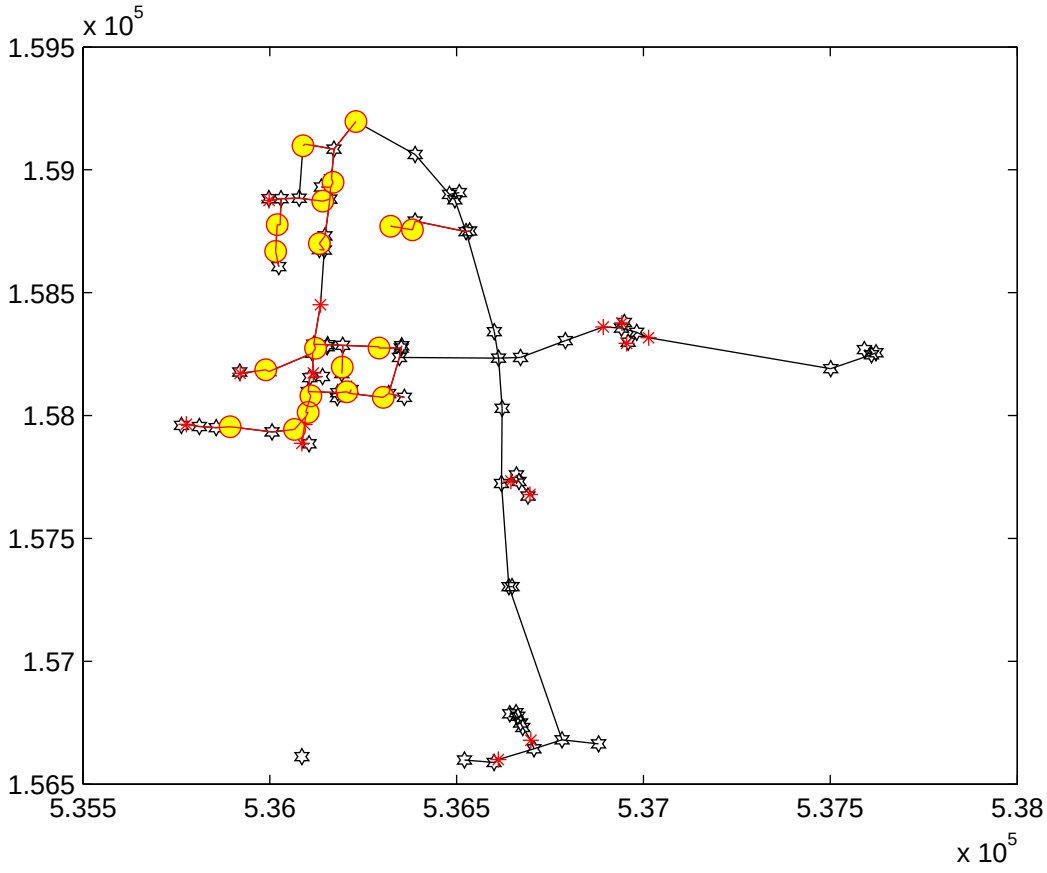

FIG. 4.1 – Résultat de l'heuristique pour Vernand

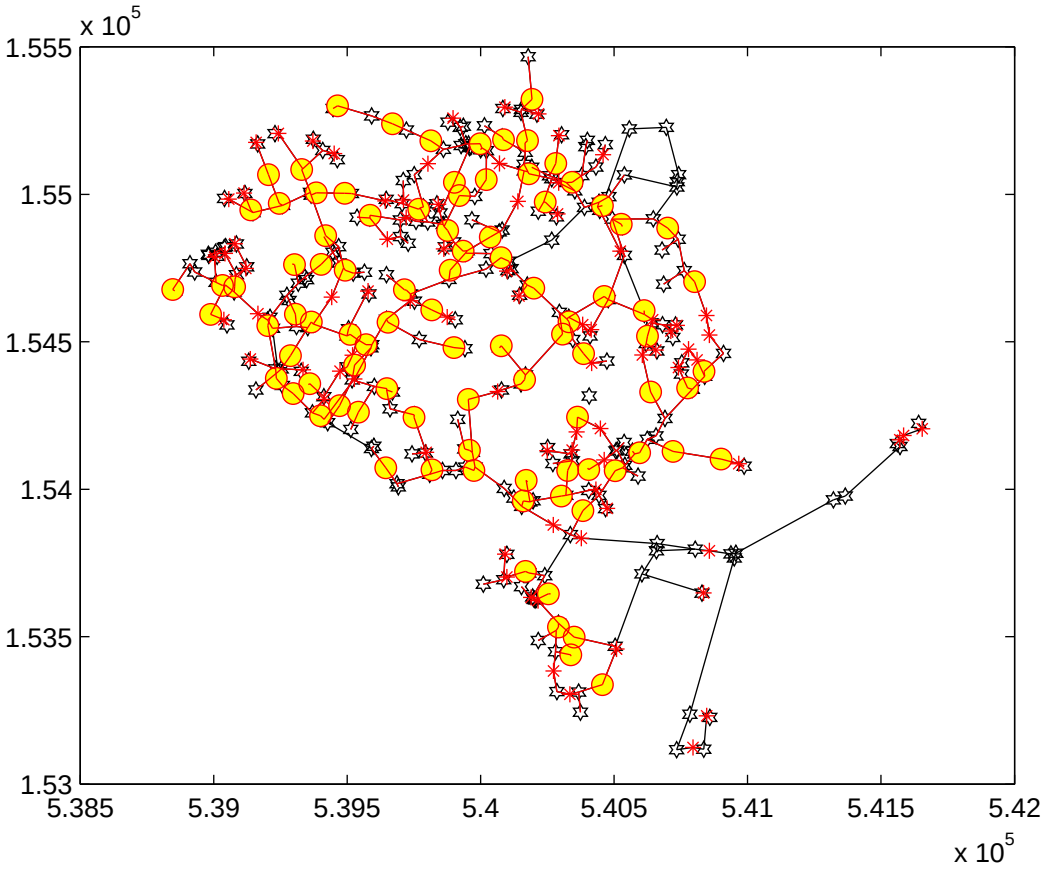

FIG. 4.2 – Résultat de l'heuristique pour Dailles

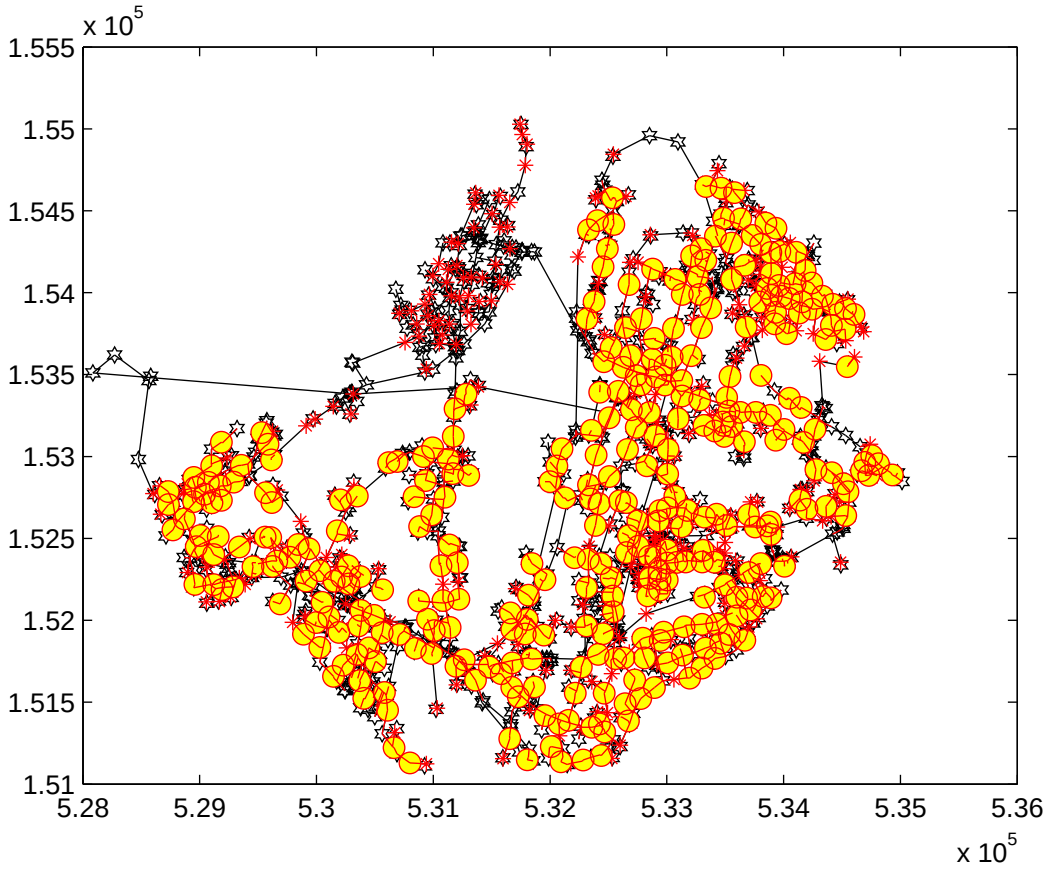

FIG. 4.3 – Résultat de l'heuristique pour Haute-Pierre

4.4 Interprétations des résultats

Le premier constat à faire est la plus grande rapidité de CPLEX. De plus, il apparaît que l'ajout de quelques *contraintes de coupe* bien choisies améliore la vitesse de résolution, notamment avec les méthodes 3 et 4.D'ailleurs , pour la recherche d'une solution pour le réseau de Dailles et de Haute-Pierre, nous utilisons la méthode 4 avec le solveur CPLEX. En ce qui concerne la résolution du PL avec le solveur Clp, il semble que l'ajout de contraintes supplémentaires au PL initial ralentisse passablement la recherche de la solution optimale. En outre, nous avons pu constater à quelques reprises des "bizarreries" de résolution (combinaison convexe de deux solutions...) que CPLEX ne fait pas.

Le deuxième constat est le gain de temps conséquent en ne prenant que la plus “grande partie connexe” du réseau : quelle que soit la méthode, la résolution ne prend que quelques secondes, voire dixièmes de secondes, contraiment à la résolution du réseau original, qui nécessitait parfois plusieurs minutes (cf. Méthode 1 et 2). Cependant, suite aux divers problèmes déjà évoqués, nous n'avons pas obtenu de résultats pour les graphes $\text{dail}_{splitte}, hpie_{splitte}$.

Nous illustrons ici la solution optimale trouvée par la méthode 4 avec CPLEX pour les graphes de Vernand et de Dailles.

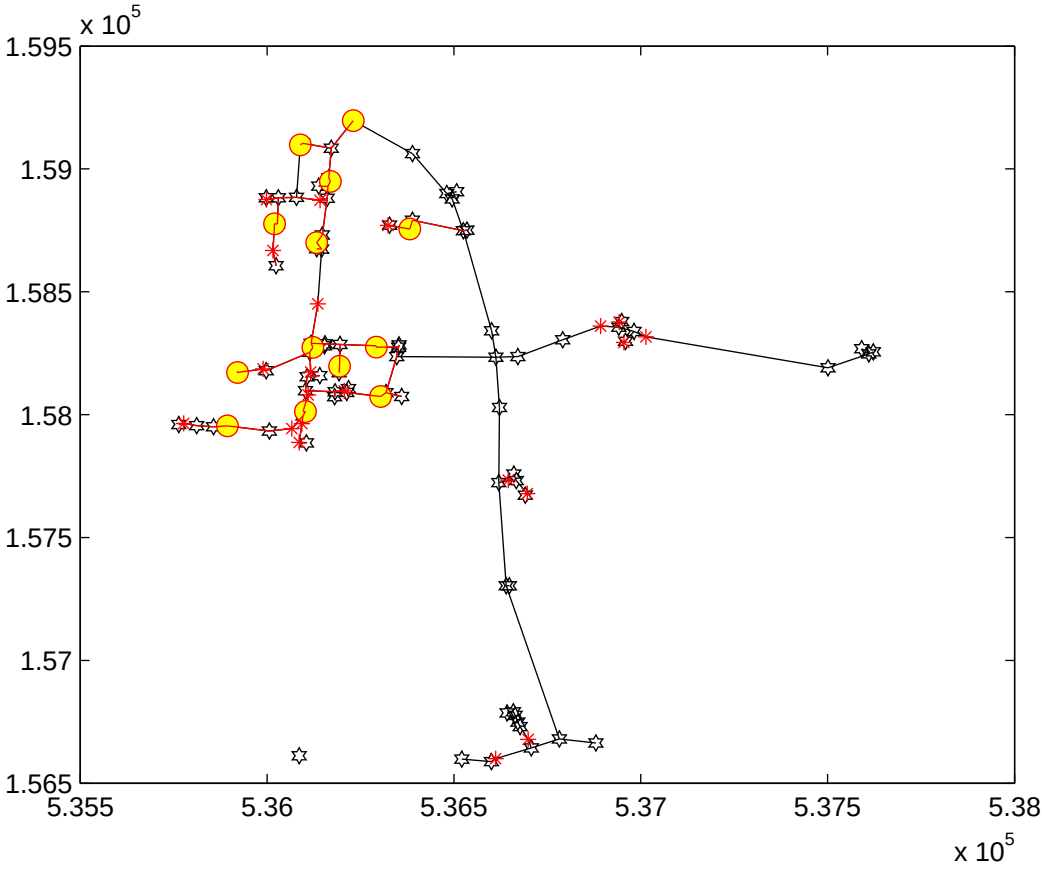

FIG. 4.4 – Solution pour Vernand

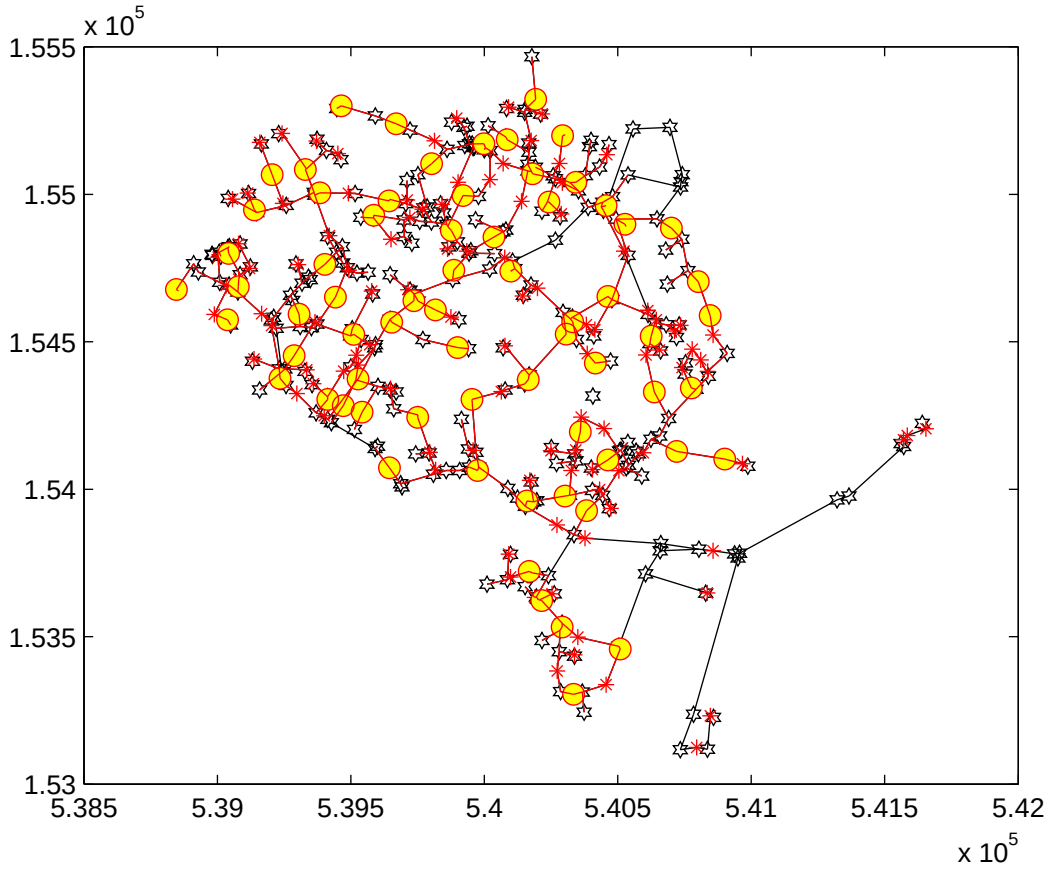

FIG. 4.5 – Solution pour Dailles

A la vue de ces solutions, nous ne sommes pas surpris de retrouver pour le graphe "vern-splitte" la même valeur optimale que le réseau de Vernand ! Les caractéristiques de la solution pour le réseau de Dailles sont les suivantes :

Méthode	Val. opt.	Temps CPU	Nb d'étape	Nb total.contr.
Méthode 4 (CPLEX)	$-2,28764.10^6$	$2103,8$	4206	29455

TAB. 4.5 – Solution optimale pour Dailles

Pour la solution optimale du réseau de Haute-Pierre, le programme employant la méthode 4 avec CPLEX n'a pas pu se terminer à cause d'un

problème de mémoire.

En comparant les résultats de la méthode 4 avec CPLEX et les résultats obtenus par l'heuristique pour les réseaux de Vernand et Dailles, nous voyons, dans les deux cas, que les solutions ont le même ensemble de régions. ce n'est pas étonnant, car de manière générale, dans notre modélisation, le gain apporté par l'écoute d'une région est très supérieure au coût d'installation d'une station. Donc, la solution optimale contiendra presque toujours tous les sommets terminaux. La différence est dans le nombre nécessaire de stations pour écouter toutes les régions. Dans notre heuristique, nous considérons les hydrantes d'après le gain potentiel que peuvent amener tous leurs sommets teminaux adjacents. Nous construisons ainsi notre arborescence en mettant toujours le plus grand nombre de sommets terminaux possible pour un sommet. Or, nous avons pu voir dans ce projet, par exemple, qu'un sommet hydrante avec un grand gain potentiel n'avait pas tous ses "arcs-régions" sortants dans la solution optimale. Néanmoins, les différences des valeurs des solutions optimales trouvées par notre heuristique et la méthode 4 pour les deux réseaux sont assez raisonnables.

Chapitre 5

Conclusion et suggestions d'amélioration

Dans ce projet, nous avons pour but principal la recherche d'arborescences de Steiner optimales pour les réseaux d'eau de la Ville de Lausanne. Nous connaissons déjà une solution optimale pour le réseau de Vernand [1] : cela nous a d'ailleurs aidé pour vérifier la validité de nos diverses méthodes d'ajout de contraintes de coupes. Il ne restait donc qu'à découvrir des solutions optimales pour les deux autres réseaux. Suite aux divers problèmes rencontrés et à l'efficacité relative de certaines méthodes, nous n'avons pu résoudre que le cas du réseau de Dailles avec la méthode 4 et le solveur CPLEX. Pour le réseau de Haute-Pierre, la grande taille de ce graphe n'a pas permis à cette méthode de trouver une solution optimale, faute de mémoire. Nous nous contenterons donc de la solution fournie par l'heuristique.

Toutefois, la résolution du réseau de Haute-Pierre n'est pas sans espoir : nous avons d'ores et déjá une borne inférieure et les résultats obtenus en ne prenant que la partie connexe de réseau en surface laisse penser que nous pouvons trouver une solution dans un temps raisonnable. En reprenant l'idée de ne considérer que la partie connexe du réseau en surface, il serait intéressant de casser le réseau en surface en plusieurs parties connexes (de la taille du réseau de Dailles) et d'appliquer la méthode 4 à chaque partie. Nous obtiendrions ainsi plusieurs arborescences que nous recollerions de la manière la plus économique. L'inconvénient d'une telle méthode est de trouver le meilleur moyen de casser le réseau pour obtenir une solution optimale. Nous avons aussi envisagé, lorsque l'algorithme *Branch & Cut* emploie l'algorithme de flots de Goldberg pour déterminer les coupes à ajouter, de regarder à chaque étape si la solution actuelle est entière. Si tel est le cas, nous regardons les variables non-nulles représentant les hydrantes en surface et nous

arrêtons l'algorithme si tous les sommets hydrantes en surface appartiennent à la même composante connexe. Il serait aussi intéressant de considérer des réseaux d'eau avec des coûts non-tous fixes.

Bibliographie

[1] Julia Dias, *Détection de fuites dans un réseau d'eau*, Projet de Master, Eté 2005.

[2] I. Ljubik, R. Weiskircherl, U. Pferschy, G.Klau, P. Mutzel, M. Fischetti, *An Algorithmic Framework for the Exact Solution of the Prize-Collecting Steiner Tree Problem*, to appear in Mathematical Programming B.

[3] A. Lucena, M.G.C. Resende, *Strong lower bounds for the prize collecting Steiner problem in graphs*, Discrete Applied Mathematics, vol. 141, pp. 277-294, 2004.

[4] D.S. Johnson, M. Minkoff, S. Phillips, *The Prize Collecting Steiner Tree Problem : Theory and Practice*, Proc. SODA 2000.