Approximation de l'arborescence de Steiner

Dimitri Watel

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN EN YVELINES
École Doctorale Sciences et Technologies de Versailles – STV

Laboratoire E3S
SUPELEC, Département Informatique

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES
Spécialité : Informatique

Présentée par :
Dimitri Watel

Pour obtenir le grade de Docteur de l'Université de Versailles Saint-Quentin-en-Yvelines

Approximation de l'arborescence de Steiner

soutenue le 26 novembre 2014

Directeur de thèse :
Dominique Barth : Professeur, Prism, UVSQ

Encadrants de thèse :
Cédric Bentz : Maître de conférence, CEDRIC, CNAM
Marc-Antoine Weisser : Enseignant-Chercheur, E3S, Supélec

Rapporteurs :
Cristina Bazgan : Professeur, LAMSADE, Université Paris Dauphine
Bruno Escoffier : Professeur, LIP6, Université Pierre et Marie Curie

Examinateurs :
Jean-Claude König : Professeur, LIRMM, Université de Montpellier
Yannis Manoussakis : Professeur, LRI, Université Paris Sud

Numéro national d'enregistrement :

Version du 3 décembre 2014

Table des matières

Introduction

Le sujet de cette thèse porte sur le *problème de l'arborescence de Steiner* [1].

Problème : Problème de l'arborescence de Steiner (DST)

Instance :

- Un graphe orienté $G = (V, A)$ contenant n nœuds et m arcs.
- Un nœud $r \in V$ appelé *racine*.
- Un ensemble de k nœuds $X \subset V$ appelés *terminaux*.
- Une fonction de pondération $\omega : A \to \mathbb{R}^+$.

Solution réalisable : Une arborescence T enracinée en r couvrant chaque terminal de X.

Optimisation : Minimiser le poids $\omega(T) = \sum_{a \in T} \omega(a)$.

Résoudre DST consiste à mutualiser des arcs pour relier la racine aux terminaux. Un exemple est donné Figure 1. On y observe en particulier qu'il ne suffit pas de relier la racine aux terminaux par des plus courts chemins pour trouver une solution optimale.

La dénomination *problème de Steiner*, dans cette thèse, désignera toujours le problème de l'arborescence de Steiner, ou DST, sauf si cela provoque une ambigüité. Le problème de Steiner peut être vu comme la généralisation de plusieurs problèmes de la littérature sur lesquels nous reviendrons plus en détail dans la partie I :

- le problème de recherche d'un chemin de poids minimum (quand $k = 1$) ;
- le problème d'arborescence couvrante de poids minimum (quand $k = n - 1$ ou $k = n$) ;
- le problème de couverture par ensembles [2] ;
- le problème de l'arbre de Steiner [3] : relier un sous-ensembles de nœuds d'un graphe non orienté par un arbre de poids minimum ;
- le problème de couverture de groupes par arbre de Steiner [4] : relier des groupes de nœuds d'un graphe non orienté par un arbre de poids minimum, l'arbre devant couvrir au moins un nœud de chaque groupe ;
- le problème de couverture de groupes par arborescence de Steiner [5], la version orientée du problème précédent.

1. *Directed Steiner Tree*, en anglais
2. *Set Cover*
3. *Undirected Steiner Tree* (UST)
4. *Undirected Group Steiner Tree*
5. *Directed Group Steiner Tree*

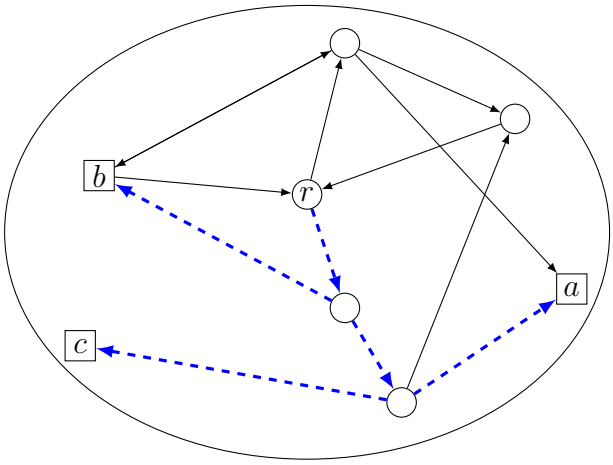


FIGURE 1 – *Un exemple d'instance de DST où les trois terminaux de X sont a, b et c. Les poids des arcs sont tous unitaires. Il faut au minimum 5 arcs pour relier r aux terminaux. Une solution optimale est représentée avec les arcs en tirets bleus. On observe en particulier que r n'est pas relié à a avec un plus court chemin.*

La première version du problème de Steiner est celle du problème de l'arbre de Steiner dans le plan euclidien, où un ensemble de points du plan, nommés *terminaux*, doivent être reliés par une série de segments pouvant faire intervenir des points intermédiaires et dont la longueur totale minimum. Premièrement, étudiée par Fermat avec 3 terminaux, cette version fut généralisée à n terminaux par Gauss [CD01]. Si le mathématicien Jakob Steiner a donné son nom au problème, c'est probablement dû au succès du livre de Richard Courant et Herbert Robbins [CR41], qui furent les premiers à le nommer ainsi [GT93, DH08, (Préfaces)].

Les problèmes de l'arbre de Steiner, de couverture par ensembles et de de couverture de groupes par arbre de Steiner sont NP-difficiles [Kar72, GKR98]. Le problème de l'arborescence de Steiner appartenant à la classe NP, il est donc également NP-difficile. Rechercher des algorithmes exacts ou d'approximation pour DST permet d'en trouver pour ces problèmes. Inversement, des résultats d'inapproximabilité pour ces problèmes s'appliquent au problème de Steiner.

On s'intéresse aujourd'hui aux problèmes de Steiner dans les graphes orientés ou non orientés principalement pour leurs applications dans la conception de circuit (problèmes d'intégration à très grande échelle ou VLSI) [CD01], mais aussi de diffusion multicast dans un réseau [Voß06, Win87]. Une machine, appelée *racine*, fait la demande d'une requête *multicast* vers un ensemble X de k terminaux dans un réseau. Le but est de satisfaire la requête tout en minimisant l'utilisation de la bande passante. Pour cela on modélise le réseau par un graphe orienté, où le poids de chaque connexion est égal au poids lié à la bande passante consommée si le paquet traverse cet arc. On calcule ensuite une arborescence optimale de Steiner, que le paquet va parcourir de la racine jusqu'aux terminaux.

En fonction du réseau ou de l'objectif, on utilisera plutôt le problème de l'arborescence de Steiner ou un des problèmes connexes. Le problème de l'arbre de Steiner, par exemple, servira quand les connexions dans le réseau sont symétriques.

Ce manuscrit se penche sur l'approximabilité (polynomiale ou non) du problème de l'arborescence de Steiner. Du fait de leur proximité, il semble naturel d'étendre les résultats obtenus dans des graphes non orientés sur le problème de l'arbre de Steiner aux graphes orientés et au problème de l'arborescence de Steiner. Nous rappelons la définition du problème de l'arbre de Steiner ci-après.

Problème : Problème de l'arbre de Steiner (UST, non orienté)

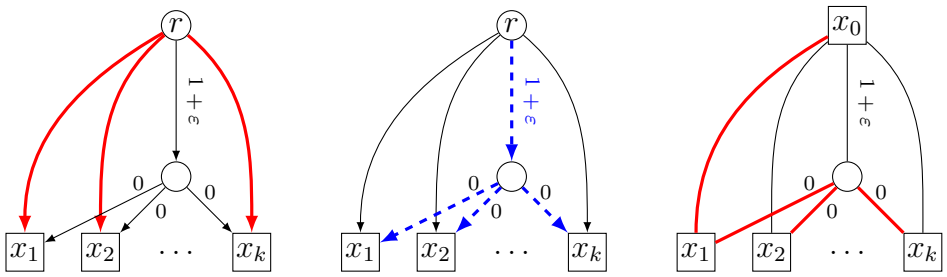
Instance :
- Un graphe non orienté $G = (V, E)$.
- Un ensemble de k nœuds $X \subset V$ appelés *terminaux*.
- Une fonction de pondération $\omega : E \to \mathbb{R}^+$.

Solution réalisable : Un arbre $T \subset G$ couvrant tous les terminaux de X.

Optimisation : Minimiser le poids $\omega(T) = \sum_{e \in T} \omega(e)$.

Le problème UST peut être approché avec un rapport constant. Une 2-approximation connue pour ce problème consiste à chercher, parmi les couples de terminaux non reliés, celui dont la distance séparant les deux terminaux est minimum, et à recommencer jusqu'à avoir relié tous les terminaux entre eux [Cho78, KMB81].

Une généralisation de cet algorithme au cas de DST est l'algorithme dit *des plus courts chemins* qui consiste à relier la racine à chaque terminal avec un plus court chemin. Cependant cet algorithme est une k-approximation (nous le démontrerons dans le chapitre 2). La figure 2 montre une instance où la solution renvoyée est k fois plus coûteuse que la solution optimale. Elle montre aussi pourquoi, une fois supprimée la dissymétrie due à l'orientation des arcs mais aussi à la racine, la version non orientée de l'algorithme n'est plus affectée par le piège que pose cette instance.



Poids : k Poids : $1 + \varepsilon$ Poids : 1

FIGURE 2 – *Instance où le poids de la solution renvoyée par l'algorithme des plus courts chemins est k fois plus grand que celui d'une solution optimale. Les arcs dont le poids n'est pas précisé ont un poids unitaire. Cette figure représente également, à gauche, une solution renvoyée par l'algorithme des plus courts chemins (en traits gras rouges), au centre, l'arborescence optimale (en traits pointillés bleus), et à droite, la solution renvoyée par l'algorithme si on supprime l'orientation (en traits gras rouges).*

De manière générale, les algorithmes d'approximation développés pour le problème de l'arbre de Steiner ne sont pas généralisables au problème de l'arborescence de Steiner.

À moins que $P = NP$, il n'existe pas d'approximation polynomiale de rapport $\log^{2-\varepsilon}(k)$ pour DST [HK03]. Trouver une approximation de rapport polylogarithmique en k pour ce problème est une question ouverte puisque le plus petit rapport d'approximation polynomiale connu pour le problème de Steiner est $O(k^{\varepsilon})$ quel que soit $\varepsilon > 0$ [CCCD98].

Résultats présentés dans ce manuscrit

L'objet de cette thèse consiste à apporter de nouveaux résultats portant sur le problème de Steiner et plus particulièrement son approximabilité polynomiale.

La partie I décrit le contexte de l'étude relatif au problème de Steiner, à ses résultats d'approximabilité connus, aux différents algorithmes exacts existants pour ce problème. Nous nous pencherons également sur les problèmes connexes.

La partie II est consacrée à la recherche d'algorithmes d'approximations pour DST dans le cas général. Trois résultats sont décrits. Les algorithmes du chapitre 4 sont des algorithmes gloutons dont l'efficacité en pratique est démontrée en fin de chapitre. Le chapitre 5 présente une approximation du problème dans le cas particulier d'un graphe découpé en paliers de nœuds, ceux-ci n'étant reliés qu'aux nœuds du palier suivant. Le rapport de cette approximation est en dessous du plus haut rapport d'inapproximabilité connu pour DST dans le cas général. Enfin, dans le chapitre 6, nous développons un schémas approximation exponentielle qui mêle approximation gloutonne et algorithme exact.

Ces algorithmes ne permettent pas de diminuer le rapport d'approximation du problème en dessous de $O(k^{\varepsilon})$. C'est pourquoi la partie III de la thèse explore une autre piste. Le problème est contraint de sorte à réduire le nombre de solutions réalisables que l'on peut renvoyer. Sachant que la solution optimale du problème contraint est elle-même une solution réalisable du problème non contraint, peut-on s'en servir pour approcher sa solution optimale ? La partie étudie les classes de complexité paramétrée et l'approximabilité des problèmes contraints. Ces contraintes, issues des problématiques de réseaux tout-optiques, sont présentées dans le chapitre 7. On se limite premièrement aux solutions qui n'utilisent qu'un nombre limité de *nœuds de branchement*. Cette étude est faite dans les chapitres 8 et 10. Secondement, on se restreint aux solutions qui n'utilisent qu'un nombre limité de *nœuds diffusants*[6]. Ce problème est étudié dans les chapitres 9 et 10.

La dernière partie de ce manuscrit est dédiée aux annexes. La première annexe contient un ensemble de démonstrations non présentes dans le chapitre 4, du fait de leur longueur. Les deux annexes suivantes se focalisent sur les expériences du chapitre 4. L'annexe B présente l'implémentation de ces expériences dans le but de pouvoir les reproduire. Elles utilisent une bibliothèque d'instances disponibles en ligne. Ces instances sont utilisées globalement. L'annexe C détaille plus finement les résultats de ces expériences. Enfin l'annexe D présente quelques notions fondamentales de la théorie de la complexité paramétrée nécessaires à la compréhension de ce manuscrit.

6. *diffusing nodes* ou *multicast-capable* en anglais. Un nœud non diffusant ne peut transmettre un paquet entrant que vers un seul destinataire, alors qu'un nœud diffusant peut le retransmettre autant de fois qu'il le souhaite vers chacun de ses destinataires.

Première partie

Contexte de l'étude

Cette partie présente les résultats existants concernant le problème de l'arborescence de Steiner au travers de trois chapitres. Le premier est dédié à la résolution exacte du problème de l'arborescence de Steiner. Il nous permet de présenter quelques unes des propriétés des solutions optimales d'une instance du problème de Steiner. Le deuxième développe les principaux résultats d'approximabilité et d'inapproximabilité du problème. Il explique en particulier les principales techniques d'approximation qui sont utilisées dans la pratique et dans ce manuscrit. Le dernier chapitre exhibe une liste de problèmes connexes, principalement constituée de sous-problèmes du problème de Steiner, et explicite les ressemblances et différences entre les résultats d'approximation de ces problèmes et ceux du chapitre précédent.

Chapitre 1.

Algorithmes exacts pour le problème de l'arborescence de Steiner

Les algorithmes exacts présentés dans ce chapitre ont tous été développés pour le problème de l'arbre de Steiner, mais sont facilement adaptables au problème de l'arborescence de Steiner. Il existe deux catégories d'algorithmes : ceux dont la complexité en temps est Fixed-Parameter Tractable (FPT) [1] vis-à-vis du nombre de terminaux k et ceux utilisant les propriétés de programmes linéaires associés au problème.

Dans toute suite du chapitre, on considère comme donnée une instance $\mathcal{I} = (G, r, X, \omega)$ de DST, vérifiant $|X| = k$.

Nous utilisons dans ce manuscrit, et en particulier dans ce chapitre, la notation O^* qui ne conserve que les membres exponentiels d'une expression à l'instar de la notation O qui ne conserve que les membres non constants.

1.1 Algorithmes exacts paramétrés par k

Le premier algorithme FPT en k conçu pour les problèmes de Steiner est celui de Dreyfus et Wagner. Cet algorithme a posé les bases d'une série d'algorithmes cherchant à réduire sa complexité en temps et/ou sa complexité en espace.

Ces algorithmes reposent tous, comme c'est le cas pour l'algorithme de Dijkstra de calcul d'un plus court chemin entre deux nœuds, sur le fait qu'une solution optimale est composées de sous-solutions optimales.

Algorithme de Dreyfus-Wagner [DW71]. Le premier algorithme FPT en k conçu pour les problèmes de Steiner est celui de Dreyfus et Wagner. Il a été redécrit dans [DYWQ07] et [KS06] pour le problème de couverture de groupes par arbre de Steiner et pour DST. Cet algorithme est une extension de l'algorithme de Dijkstra pour les plus courts chemins. Il s'agit d'un algorithme de programmation dynamique qui, pour tout nœud v et tout sous-ensemble de terminaux $X' \subset X$, construit l'arborescence de Steiner de poids minimum enracinée en v et couvrant X'. Lorsque $v = r$ et $X' = X$, alors l'algorithme s'arrête et renvoie la solution.

1. Des notions de complexité paramétrée utiles à ce chapitre sont données en annexe D.

La programmation dynamique est basée sur deux faits, illustrés par la figure 1.1. Si, premièrement la racine r d'une solution optimale T^* n'a qu'un seul fils u alors, nécessairement, le sous-arbre enraciné en u est une solution optimale à l'instance $\mathcal{I} = (G, u, X, \omega)$. Dans le cas contraire, la racine a au moins deux fils et T^* peut être décrit comme l'union de deux arborescences T_1 et T_2 disjointes par les sommets sauf en r. Ainsi, les sous-ensembles X_1 et X_2 de terminaux couverts respectivement par T_1 et T_2 forment une partition de X.

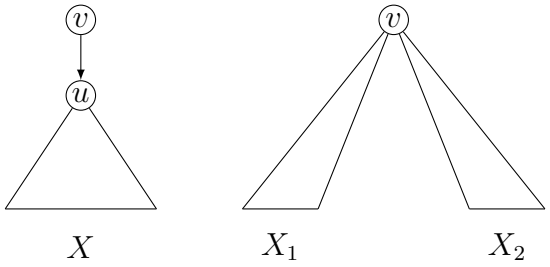


Figure 1.1

On obtient donc l'équation ci-après, où $T^*(v, X')$ est l'arborescence de poids minimum enracinée en v couvrant X'.

$$\omega(T^*(v,X')) = \min \begin{cases} \omega(v,u) + \omega(T^*(u,X')) & \text{pour } (v,u) \in A \\ \omega(T^*(v,X_1)) + \omega(T^*(v,X_2)) & \text{pour } X_1 \cup X_2 = X' \end{cases}$$

L'algorithme de Dreyfus-Wagner calcule, à l'aide de cette équation, toutes les solutions optimales des instances (G, v, X', ω) pour $v \in V$ et $X' \subset X$. Un exemple d'exécution est donné par la figure 1.2.

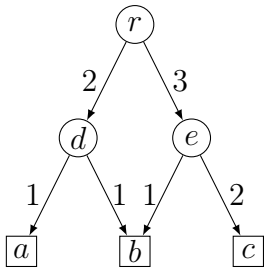
Il s'initialise avec les arbres $T^*(x, \{x\})$, de poids 0 et ne contenant que le nœud x, pour tout x, et les insère tous dans une liste $\mathcal{L}$, associés à leur poids. Au début de chaque itération, le premier élément $T(u, X_1)$ de $\mathcal{L}$ est extrait. Pour tout arc (v, u), $T(v, X_1)$ est inséré dans $\mathcal{L}$ avec le poids $\omega(v, u) + \omega(T(u, X_1))$, ou s'il était déjà présent, il remplace l'ancien $T(v, X_1)$ s'il a un poids plus faible. De même, pour tout $X_2 \subset X$ disjoint de X_1 tel que $T(u, X_2)$ soit dans $\mathcal{L}$ ou en ait été extrait, l'arborescence $T(u, X_1 \cup X_2)$ est insérée dans $\mathcal{L}$ avec le poids $\omega(T(u, X_1)) + \omega(T(u, X_2))$, ou remplace l'ancien $T(u, X_1 \cup X_2)$ s'il était déjà présent et s'il a un poids plus faible. La liste $\mathcal{L}$ est ensuite retriée, si nécessaire, par ordre de poids.

La propriété fondamentale de cet algorithme est que tout arbre $T(u, X_1)$ extrait de $\mathcal{L}$ a un poids optimal : $T(u, X_1) = T^*(u, X_1)$. Quand $T(r, X)$ est extrait de $\mathcal{L}$ c'est une solution optimale de $\mathcal{I}$.

Cet algorithme a une complexité en temps égale à $O(3^k n + 2^k(k + \log(n)n + m)) = O^*(3^k)$ si la liste $\mathcal{L}$ est implémentée sous forme d'un tas de Fibonnacci. Sa complexité spatiale est $O(2^k n)$.

Les questions que posent l'existence de cet algorithme sont de savoir s'il est possible :

1. de trouver pour le problème de Steiner un algorithme exact en temps FPT en k mais polynomial en espace ; autrement dit, peut-on construire une solution optimale sans stocker toutes les sous-solutions optimales ;
2. de réduire la complexité en temps en dessous de $O^*(3^k)$; autrement dit, peut-on calculer une solution optimale couvrant X' sans tester l'union des solutions optimales couvrant les partitions de X'.



Itération	$\mathcal{L}$
0	$(a,\{a\},0),(b,\{b\},0),(c,\{c\},0)$
1	$(b,\{b\},0),(c,\{c\},0),(d,\{a\},1)$
2	$(c,\{c\},0),(d,\{a\},1),(d,\{b\},1),(e,\{b\},1)$
3	$(d,\{a\},1),(d,\{b\},1),(e,\{b\},1),(e,\{c\},2)$
4	$(d,\{b\},1),(e,\{b\},1),(e,\{c\},2),(d,\{a,b\},2),(r,\{a\},3)$
5	$(e,\{b\},1),(e,\{c\},2),(d,\{a,b\},2),(r,\{a\},3),(r,\{b\},3)$
6	$(e,\{c\},2),(d,\{a,b\},2),(r,\{a\},3),(r,\{b\},3),(e,\{b,c\},3)$
7	$(d,\{a,b\},2),(r,\{a\},3),(r,\{b\},3),(e,\{b,c\},3),(r,\{c\},5)$
8	$(r,\{a\},3),(r,\{b\},3),(e,\{b,c\},3),(r,\{a,b\},4),(r,\{c\},5)$
9	$(r,\{b\},3),(e,\{b,c\},3),(r,\{a,b\},4),(r,\{c\},5),(r,\{a,c\},8)$
10	$(e,\{b,c\},3),(r,\{a,b\},4),(r,\{c\},5),(r,\{a,c\},8),(r,\{b,c\},8)$
11	$((r,\{a,b\},4),(r,\{c\},5),(r,\{b,c\},6),(r,\{a,c\},8)$
12	$(r,\{c\},5),(r,\{b,c\},6),(r,\{a,c\},8),(r,\{a,b,c\},9)$
13	$(r,\{b,c\},6),(r,\{a,c\},8),(r,\{a,b,c\},9)$
14	$(r,\{a,c\},8),(r,\{a,b,c\},9)$
15	$(r,\{a,b,c\},9)$

FIGURE 1.2 – *Exemple d'application de l'algorithme de Dreyfus-Wagner.*

.

Les deux algorithmes suivants ont permis de répondre successivement par l'affirmative à ces deux questions.

Algorithme de Bjorklund *et al.* [BHKK07]. L'algorithme de Bjorklund est le premier à atteindre une complexité en temps égale à $O^*(2^k)$. Pour cela, il utilise la transformée de Möbius.

On peut modifier l'équation de programmation dynamique de l'algorithme de Dreyfus-Wagner comme suit, en posant $\omega(v,v)=0$:

$$\omega(T^*(v,X)) = \min_{u\in V} \omega(v,u) + g_u(X)$$
$$g_u(X) = \min_{X'\subset X} \omega(T^*(u,X')) + \omega(T^*(u,X\backslash X'))$$

L'idée est de voir l'expression $g_u(X)$ comme un produit de convolution dans le semi-anneau $(\mathbb{N}, min, +)$. La transformée de Möbius [2] permet de réduire le temps de calcul de toutes les convolutions $g_u(X')$, pour $u \in V$ et $X' \subset X$, de $O^*(3^k)$ à $O^*(2^k)$. La complexité spatiale reste $O^*(2^k)$ car l'algorithme stocke toujours, pour tout couple (v, X'), la solution optimale $T^*(v, X')$.

Cette implémentation a deux défauts. Premièrement, contrairement à l'algorithme de Dreyfus, la complexité n'est ici pas une borne supérieure du nombre d'opérations, elle indique exactement ce nombre. Secondement, la transformée de Möbius manipule de très grands entiers, dont la taille dépend du poids des arcs de l'instance. Ainsi, la complexité en temps et en espace dépendent tous les deux linéairement du poids le plus élevé dans l'instance. Si ce poids n'est pas maîtrisé, ces complexités sont donc pseudo-polynomiales.

Algorithme de Nederlof [Ned09]. Enfin, l'algorithme de Nederlof réussit à réduire la complexité en temps à $O^*(2^k)$ tout en conservant un espace polynomial. L'idée est d'utiliser le principe d'inclusion-exclusion. Par le biais d'une formule de programmation dynamique, on peut ainsi décider, dans le cas où les poids sont unitaires, s'il existe une arborescence de poids inférieur à un entier N fixé, en temps égal à $0^*(2^k)$ et en espace polynomial. L'auteur de cet algorithme précise que, dans le cas où les poids ne sont pas unitaires, une variante utilisant la transformée de Möbius donne le même résultat.

Cet algorithme possède les mêmes défauts que l'algorithme de Bjorklund *et al.* : la complexité en temps est exacte et dépend linéairement du poids le plus élevé des arcs de l'instance. On peut ajouter qu'à la différence de tous les autres algorithmes, celui-ci **ne construit pas l'arborescence optimale** mais ne peut que calculer son poids. En effet, ce poids est la résultante d'une somme dont il n'est pas possible d'extraire de l'information.

La valeur d'une solution optimale seule ne suffit pas à construire une solution optimale. Cependant, cet algorithme nous donne la valeur d'une solution optimale de toute instance du problème de Steiner. En utilisant l'algorithme 1, nous somme en mesure de construire une solution optimale. Cet algorithme n'utilise qu'un nombre linéaire de fois l'algorithme de Nederlof et est donc également FPT en k et polynomial en espace.

Il n'existe pas aujourd'hui d'algorithme ayant une complexité en temps meilleure que $O^*(2^k)$.

Nous introduirons dans le chapitre 10 un algorithme d'une toute autre nature qui est également FPT en k et utilise un espace polynomial pour construire une solution optimale.

Ces algorithmes se basent sur le fait qu'il est possible de construire une solution optimale à partir de sous-solutions optimales. Le temps de calcul est exponentiel car le nombre de ces sous-solutions est exponentiel. Une approximation du problème de Steiner peut donc être construite en ne conservant qu'un nombre polynomial de telles sous-solutions. Meilleure sera la qualité de ces sous-solutions, meilleure sera le rapport d'approximation. Nous le verrons ultérieurement dans le chapitre 2, tous les algorithmes d'approximation utilisent ce principe, et nous le réutiliserons également dans le chapitre 4.

Il faut toutefois noter que les algorithmes présentés dans cette section ne sont que rarement utilisés dans la pratique. Nous présentons dans la section suivante de ce chapitre

2. Dans un anneau ou semi-anneau $(A, +, \times)$, la transformée de Möbius de f sur un ensemble $X \subset A$ est $\sum_{S \subset X} f(S)$.

Algorithme 1 Algorithme construisant une solution optimale à partir d'un algorithme renvoyant la valeur d'une solution optimale.

Entrées : Une instance $\mathcal{I} = (G = (V, A), r, X, \omega)$ de DST et un algorithme $\mathcal{A}$ qui, connaissant une instance du problème de Steiner, renvoie la valeur d'une solution optimale de cette instance ou $+\infty$ s'il n'existe pas de solution réalisable.
Sorties : Une solution optimale de $\mathcal{I}$.

$\omega^* \leftarrow \mathcal{A}(\mathcal{I})$
Si $(\omega^* = +\infty)$ **Renvoyer** $\emptyset$
Si ($\mathcal{I}$ est une arborescence de poids ω^*) **Renvoyer** G
Pour tout nœud v de $V \backslash (\{r\} \cup X)$ **Faire**
    $\mathcal{I}_v \leftarrow \mathcal{I}$ privé du nœud v et de tous ses arcs adjacents
    Si $\mathcal{A}(\mathcal{I}_v) = \omega^*$ **Alors**
        $\mathcal{I} \leftarrow \mathcal{I}_v$
        Si ($\mathcal{I}$ est une arborescence de poids ω^*) **Renvoyer** G

deux algorithmes exacts plus efficaces basés sur des techniques connues en programmation linéaire.

1.2 Algorithmes exacts basés sur la programmation linéaire

Il existe deux représentations naturelles du problème de Steiner sous forme de programme linéaire, nommés (PP1) et (PP2) [3], dont nous reparlerons plus en détail dans le chapitre 2. Les deux associent une variable $x_a \in \{0, 1\}$ à chaque arc a du graphe selon qu'il est choisi ou non dans la solution réalisable que le programme renvoie.

Nous nous intéressons ici principalement au programme (PP2) présenté ci-après.

Dans cette version, on regarde toutes les coupes du graphe séparant la racine d'au moins un terminal. Chaque coupe S doit être traversée par au moins un arc, partant de la racine vers le terminal. Dans le programme qui suit, $\Gamma^-(S)$ décrit les arcs entrant dans l'ensemble de nœuds de S, c'est-à-dire les arcs dont l'origine est hors de S et la destination est dans S.

Programme linéaire (PP2) : Description de DST par coupes

$$\textbf{Minimiser} \quad \sum_{a \in A} \omega(a) \cdot x_a$$

$$\textbf{s.c.} \quad \sum_{a \in \Gamma^-(S)} x_a \geq 1 \qquad \text{pour } S \subseteq V \text{ tel que } r \notin S \text{ et } S \cap X \neq \emptyset$$

$$x_a \in \{0, 1\} \qquad \text{pour } a \in A$$

On peut citer comme exemples d'algorithmes ceux décrits dans [LB98, KM98, Bea84, Bea89, BBP03] basés sur des techniques qui diffèrent lors du prétraitement des instances et

3. Dans ce manuscrit, nous écrirons les noms de programme linéaire entre parenthèses pour les différencier des noms de problèmes d'algorithmique.

du traitement des programmes linéaires pour extraire des bornes supérieurs et inférieurs du poids de la solution optimale de bonne qualité. Nous nous intéressons dans cette section à deux algorithmes exacts pour DST qui présentent les principes de bases couramment utilisés par ce type d'algorithme. Le premier utilise la technique de génération de contraintes. Son intérêt réside dans sa ressemblance avec l'algorithme de Dreyfus-Wagner, en générant des sous-solutions optimales jusqu'à trouver une solution réalisable. Le second est un algorithme de *Branch and Bound* utilisant, pour accélérer ses recherches, des algorithmes d'approximations efficaces.

Algorithme de génération de contraintes [SV06]. Le programme linéaire (PP2) décrit les instances de DST sous forme de coupes. Ce programme possède un nombre exponentiel de contraintes. Construire ce programme prend donc en soit un temps exponentiel [4]. Or toute solution de base d'un programme linéaire peut être décrite comme une solution saturant un nombre de contraintes linéaire en le nombre de variables. Il suffit donc d'un nombre linéaire de contraintes (bien choisies) pour caractériser une solution de base optimale. On cherche donc à résoudre le programme sans tenir compte de toutes les contraintes, en évitant, entre autres, les contraintes redondantes.

On initialise l'algorithme en ne conservant qu'une contrainte par terminal : $\sum_{a \in \Gamma^-(\{x\})} x_a \geq 1$ pour tout $x \in X$. On utilise un solveur pour chercher une solution réalisable entière T' au programme. Si la solution renvoyée n'est pas une solution réalisable de l'instance $\mathcal{I}$, alors on ajoute, pour chaque composante connexe de T' ne contenant pas r, la contrainte relative à cette composante, et on recommence.

Si T' est une solution réalisable de l'instance, alors c'est une solution optimale. En effet, elle est optimale pour un sous-ensemble restreint de contraintes, donc elle l'est aussi pour l'ensemble des contraintes.

La complexité en temps de cet algorithme est $O(2^n t)$ où $t = O(2^m)$ est le temps de résolution du programme linéaire (m est le nombre d'arcs de l'instance et 2^m l'ensemble des solutions réalisables et non réalisables du programme linéaire). Empiriquement, cet algorithme est plus rapide que les algorithmes présentés en début de chapitre. Il utilise le même principe : calculer des sous-solutions optimales et les unifier jusqu'à trouver une solution optimale couvrant tous les terminaux depuis la racine. À la différence de l'algorithme de Dreyfus-Wagner qui teste de très nombreuses sous-solutions avant d'atteindre la racine, celui-ci est guidé, plus rapidement, par le programme linéaire.

Algorithme de Branch and Bound [dAaUW01]. Cet algorithme part du principe que l'on peut supprimer du graphe tout nœud qui n'est pas dans une solution optimale. Inversement, tout nœud qui appartient à cette solution peut être déclaré terminal et ajouté à X. On a donc pour chaque nœud deux possibilités à explorer. L'ensemble des décisions nécessite $O^*(2^n)$ opérations.

Soit $\mathcal{I}^{-v}$ l'instance $\mathcal{I}$ où v est retiré du graphe, et $\mathcal{I}^{+v}$ l'instance $\mathcal{I}$ où v est ajouté à X. On représente les choix sous forme d'un arbre des possibles $\mathcal{P}$: les nœuds du graphe, mis à part la racine et les terminaux, sont ordonnés arbitrairement : $V \backslash (\{r\} \cup X) = \{v_1, v_2, \cdots, v_{n-k-1}\}$. La racine de $\mathcal{P}$ représente $\mathcal{I}$. Son fils gauche représente $\mathcal{I}^{+v_1}$ et son fils droit $\mathcal{I}^{-v_1}$. Leurs fils gauches et droits représentent respectivement $\mathcal{I}^{+v_1,+v_2}$, $\mathcal{I}^{+v_1,-v_2}$, $\mathcal{I}^{-v_1,+v_2}$, $\mathcal{I}^{-v_1,-v_2}$, ...

4. Bien qu'il soit possible de résoudre sa relaxation continue en temps polynomial [BBP03].

Les feuilles de $\mathcal{P}$ correspondent à une instance $\mathcal{J}$ où tout nœud de V est soit la racine, soit terminal, soit retiré du graphe. Éventuellement $\mathcal{J}$ n'est pas connexe et n'a pas de solution. Dans le cas contraire, la solution optimale de $\mathcal{J}$ est une arborescence enracinée en r couvrant tous les nœuds avec un poids optimal. Cette arborescence peut se calculer en temps polynomial avec l'algorithme d'Edmonds [Edm67, Tar77].

On peut calculer une solution optimale de $\mathcal{I}$ en calculant, pour chaque feuille de $\mathcal{P}$, la solution optimale de l'instance correspondante, si elle existe, et en renvoyant celle de poids le plus faible. Nécessairement, il existe une feuille dont la solution optimale est une solution optimale T^* de l'instance. Il s'agit de la feuille correspondant à l'instance $\mathcal{I}^{-V\setminus T^*,+T^*}$.

Pour réduire le temps d'exploration, certaines branches de l'arbre des possibles sont coupées à l'aide d'une recherche de bornes supérieures et inférieures du poids de la solution optimale.

On initialise T avec une solution réalisable donnée par une heuristique quelconque de $\mathcal{I}$. Le poids $\omega(T)$ est notre première borne supérieure $\mathcal{B}^+$.

Un curseur parcourt $\mathcal{P}$ en profondeur. Quand le curseur visite un nouveau nœud d'instance $\mathcal{J}$, si ce nœud est une feuille, on applique l'algorithme d'Edmonds sur l'instance $\mathcal{J}$. Sinon on construit une solution réalisable de $\mathcal{J}$ à l'aide d'une heuristique ou d'un algorithme d'approximation polynomial. Dans les deux cas, on compare le poids de la solution obtenue à la borne $\mathcal{B}^+$. Si cette solution est moins chère, alors on diminue la borne en conséquence.

Sinon, on recherche une borne inférieure $\mathcal{B}^-$ de la solution optimale de l'instance $\mathcal{J}$ (voir ci-après) que l'on compare à $\mathcal{B}^+$. Si $\mathcal{B}^- \geq \mathcal{B}^+$, on sait que toute instance dérivée de $\mathcal{J}$ a une solution optimale plus chère que $\mathcal{B}^+$, et donc que la solution optimale de $\mathcal{I}$. Il n'y a pas d'intérêt à explorer les descendants du nœud où se situe le curseur.

Pour définir la borne inférieure $\mathcal{B}^-$, une première méthode consiste à résoudre la relaxation continue du programme linéaire (PP2). Quand cette méthode est trop lente, on peut à la place chercher une approximation du dual de (PP2). Nous verrons, dans le chapitre 2, que des heuristiques dites d'*ascension du dual* augmentent les variables duales du programme dual jusqu'à trouver une solution réalisable. Ces heuristiques peuvent être utilisées pour trouver à la fois une borne inférieure et une solution réalisable de $\mathcal{J}$.

Cet algorithme, plus efficace que les précédents, a permis de constituer un jeu de tests décrit et utilisé dans le chapitre 4.

Cet algorithme, contrairement aux autres, ne constitue pas une piste pour la conception d'un algorithme d'approximation. Il a toutefois un avantage certain face aux autres : il construit rapidement des solutions réalisables. On peut donc lui laisser un temps de calcul fixé à l'avance. Si avant la fin de ce temps, il a trouvé une solution optimale, il la renvoie. Sinon, à la fin du temps imparti, il renvoie la meilleure solution qu'il a trouvé.

Il faut également noter que son efficacité dépend de celle des algorithmes qui calculent les bornes supérieures $\mathcal{B}^+$ et inférieures $\mathcal{B}^-$. La première est calculée par un algorithme d'approximation. Ainsi, la découverte d'un algorithme d'approximation rapide et efficace implique une amélioration certaine de l'algorithme exact de Branch and Bound.

Chapitre 2.

Approximabilité du problème de l'arborescence de Steiner

Nous présentons dans ce chapitre une série de résultats d'approximabilité et d'inapproximabilité connus pour DST. La première section décrit deux résultats préliminaires concernant les graphes sans circuits et les graphes complets avec inégalité triangulaire. Les sections suivantes sont dédiées à l'approximabilité puis à l'inapproximabilité dans le cas général.

Certaines preuves de résultats connus, absentes dans la littérature mais utiles à la compréhension des méthodes employées dans ce manuscrit, sont détaillées dans les différentes sections de ce chapitre.

2.1 Résultats préliminaires.

Cette section décrit deux réductions isofacteurs [1] montrant que l'approximabilité du problème de Steiner est identique dans le cas général, dans le cas des graphes sans circuits et dans le cas des graphes complets dont les poids respectent l'inégalité triangulaire.

Cas des graphes sans circuits.

Théorème 2.1.1. *Il existe une α-approximation pour DST dans le cas général si et seulement s'il existe une α-approximation pour DST dans le cas d'un graphe sans circuits.*

Démonstration. La preuve de la condition nécessaire est triviale. Pour démontrer l'implication inverse, nous allons présenter une réduction isofacteur du cas général vers le cas sans circuits. Toute instance $\mathcal{I} = (G = (V, A), r, X, \omega)$ de DST peut être réduite en une instance sans circuits $\mathcal{I}_a$, conservant le poids de toutes les solutions réalisables. Un exemple de réduction est décrit par la figure 2.1. Cette réduction se procède ainsi :

- générer m copies de chaque nœud et les numéroter de 1 à m ;
- relier tout couple de copies d'un même nœud de numéros successifs par un arc de poids nul ;

1. Une réduction isofacteur est une réduction polynomiale qui transforme une instance d'un premier problème d'optimisation en instance d'un second. Les solutions optimales des deux instances doivent avoir même poids. Puis cette réduction transforme toute solution réalisable de la seconde instance en solution réalisable de la première de poids moins cher. Ainsi, toute approximation polynomiale de rapport α pour le second problème est une approximation de même rapport pour le premier.

- pour tout arc (u, v) dans $\mathcal{I}$, relier la i^{e} copie de u à la $(i+1)^{\text{e}}$ copie du nœud v avec un arc de poids $\omega(u, v)$, pour tout $i < m$;
- la racine de $\mathcal{I}_a$ est la première copie de r ;
- les terminaux de $\mathcal{I}_a$ sont les dernières copies des terminaux.

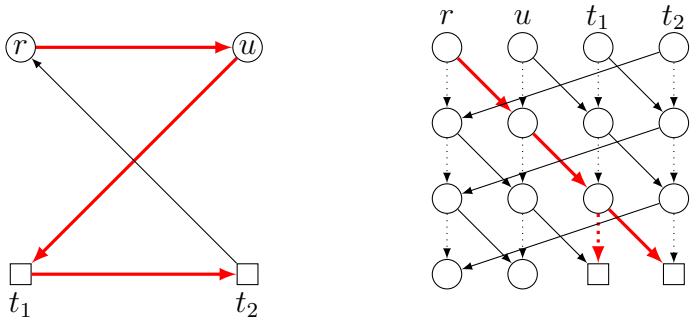


FIGURE 2.1 – *Exemple de réduction du problème de l'arborescence de Steiner depuis le cas général vers le cas d'un graphe sans circuits. Tous les arcs en trait plein sont de poids unitaire. Les arcs tracés en pointillés sont de poids nul. Une solution optimale est illustrée en traits rouges dans les deux graphes.*

Dans cette instance, les solutions réalisables et optimales de $\mathcal{I}$ sont transformables en temps polynomial en solutions de $\mathcal{I}_a$ de même poids. Inversement, à partir de toute solution réalisable T_a de $\mathcal{I}_a$, il est possible de construire en temps polynomial une solution réalisable de $\mathcal{I}$ de poids plus petit. Pour cela, on sélectionne tous les arcs ayant une copie dans T_a. Si le résultat n'est pas une arborescence, alors on peut détecter ses cycles en temps polynomial et retirer une partie des arcs des cycles pour obtenir une arborescence couvrant tous les terminaux et enracinée en r. Le poids de cette arborescence est plus petit que celui de T_a. □

Cas des graphes complets avec inégalité triangulaire. De même, dans le cas d'une instance complète dont les poids respectent l'inégalité triangulaire, l'approximabilité du problème est identique au cas général.

Pour le montrer, nous allons premièrement définir la notion d'instance des plus courts chemins, que nous serons amenés à réutiliser plusieurs fois dans les différentes parties à venir.

Définition 1. Soit $\mathcal{I} = (G = (V, A), r, X, \omega)$ une instance de DST, on construit *l'instance des plus courts chemins* $\mathcal{I}^{\triangleright} = (G^{\triangleright} = (V, A^{\triangleright}), r, X, \omega^{\triangleright})$ associée à $\mathcal{I}$ en :

- construisant le graphe $G^{\triangleright}$ orienté complet (il contient deux arcs opposés entre chaque couple de nœuds) contenant tous les nœuds de G ;
- définissant le poids $\omega^{\triangleright}$ de tout arc (u, v) de $G^{\triangleright}$ comme la valeur d'un plus court chemin dans $\mathcal{I}$ entre u et v, s'il existe, et $+\infty$ sinon.

On vérifie aisément que l'instance $\mathcal{I}^{\triangleright}$ respecte l'inégalité triangulaire.

Théorème 2.1.2. *Il existe une α-approximation pour DST dans le cas général si et seulement s'il existe une α-approximation pour DST dans le cas d'un graphe orienté complet dont les poids respectent l'inégalité triangulaire.*

Démonstration. La preuve de la condition nécessaire est triviale. Pour démontrer l'implication inverse, nous allons présenter une réduction isofacteur du cas général vers le cas orienté

complet avec inégalité triangulaire. Pour cela, on réduit toute instance $\mathcal{I}$ en son instance des plus courts chemins $\mathcal{I}^{\triangleright}$.

Toute solution réalisable T de $\mathcal{I}$ est transformable en temps polynomial en solution de $\mathcal{I}^{\triangleright}$ en sélectionnant pour tout arc (u, v) de T l'arc (u, v) de $G^{\triangleright}$. Le poids de cet arc est nécessairement plus petit dans $\mathcal{I}^{\triangleright}$ que dans $\mathcal{I}$. Inversement, à partir de toute solution réalisable $T^{\triangleright}$ de $\mathcal{I}^{\triangleright}$, il est possible de construire en temps polynomial une solution réalisable de $\mathcal{I}$ de poids plus petit. Pour cela, on sélectionne pour tout arc (u, v) de $T^{\triangleright}$ un plus court chemin dans G reliant u à v. Si le résultat n'est pas une arborescence, alors on peut détecter ses cycles en temps polynomial et retirer une partie des arcs des cycles pour obtenir une arborescence couvrant tous les terminaux et enracinée en r. Le poids de cette arborescence est plus petit que celui de $T^{\triangleright}$. □

2.2 Résultats d'approximabilité

Il existe deux techniques utilisées pour approcher le problème de Steiner :
- utiliser des algorithmes gloutons, agrégeant des couvertures partielles de terminaux ;
- utiliser la technique dite d'*ascension du dual*[2], qui construit une solution réalisable du problème de Steiner à partir d'un problème dual qui lui est associé.

2.2.1 Algorithmes gloutons pour DST : l'algorithme des plus courts chemins et l'algorithme dit *de Charikar*

L'algorithme de type glouton le plus simple connu pour approcher le problème de l'arborescence de Steiner est l'algorithme dit *des plus courts chemins*. Cet algorithme sélectionne pour chaque terminal x un plus court chemin reliant r à x et l'ajoute à la solution courante.

Théorème 2.2.1. *L'algorithme des plus courts chemins est une k-approximation pour le problème de Steiner. Ce rapport est la meilleure garantie que l'algorithme puisse donner.*

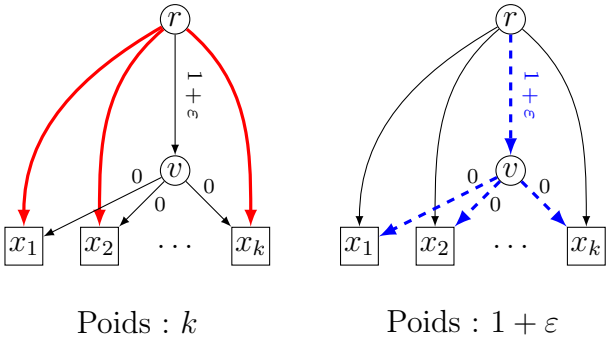
Démonstration. Étant donné un terminal x, toute solution optimale T^* contient un chemin de la racine vers x. Donc le poids d'un plus court chemin $P(r, x)$ reliant r à x est plus petit que le poids d'une solution optimale : $\omega(P(r, x)) \leq \omega(T^*)$. Donc

$$\begin{aligned}\omega(\bigcup_{t \in X} P(r, t)) &\leq \sum_{x \in X} \omega(P(r, t)) \\ &\leq k \cdot \omega(T^*)\end{aligned}$$

Nous rappelons avec la figure 2.2 la famille d'instances présentée dans l'introduction en page 3 pour laquelle la solution renvoyée par l'algorithme des plus courts chemins se rapproche autant qu'on le souhaite d'une k-approximation. Dans cette instance la racine est reliée à chaque terminal avec un arc de poids 1, mais également avec un chemin passant par un nœud v. Le poids de l'arc (r, v) est $1 + \varepsilon$ et v est relié à tous les terminaux par un arc de poids nul. Pour chaque terminal, le plus court chemin est donc l'arc de poids 1 et la solution renvoyée par l'algorithme des plus courts chemins a un poids égal à k, tandis que la solution

2. *Dual ascent* en anglais

optimale a un poids égal à $1+\varepsilon$. Le rapport d'approximation converge donc vers k lorsque ε tend vers 0. prouvant ainsi la seconde partie du théorème. □



Poids : k Poids : $1+\varepsilon$

FIGURE 2.2 – *Dans cette instance, la solution renvoyée par l'algorithme des plus courts chemins a un poids égal à k (représentée en traits gras rouges), tandis que la solution optimale a un poids égal à $1+\varepsilon$ (représentée en tirets bleus).*

Dans [CCCD98], Charikar *et al.* présentent un algorithme glouton étendant l'algorithme des plus courts chemins. Leur algorithme repose sur deux idées distinctes : trouver un arbre de faible densité et limiter la profondeur des arbres recherchés. On dénommera cet algorithme par *l'algorithme de Charikar*.

On considère comme donnée une instance $\mathcal{I} = (G, r, X, \omega)$ du problème de Steiner, de solution optimale T^*.

Trouver des arbres de faible densité.

Définition 2. Soit T une arborescence enracinée en r couvrant une partie $X(T)$ des terminaux X de $\mathcal{I}$. La *densité* de T est le rapport entre le poids de T et le nombre de terminaux couverts : $d(T) = \frac{\omega(T)}{|X(T)|}$.

Un arbre de faible densité est donc un arbre qui fait le compromis entre couvrir de nombreux terminaux et pesant peu. Mais un tel arbre ne couvre pas tous les terminaux et n'est donc qu'une solution partielle au problème de Steiner. Il faudra en associer plusieurs pour obtenir une solution réalisable. Pour ce faire si $\mathcal{A}$ est un algorithme qui renvoie une telle solution partielle, couvrant au moins un terminal, il suffit de répéter $\mathcal{A}$ suffisamment de fois (au plus k) pour obtenir une solution couvrant tous les terminaux : à chaque itération, on applique $\mathcal{A}$ qui renvoie une arborescence T, puis on retire temporairement de l'ensemble des terminaux ceux couverts par T. Charikar *et al.* montrent que s'il est possible de borner la densité des arbres renvoyés par $\mathcal{A}$ alors la solution réalisable construite en répétant $\mathcal{A}$ est un algorithme d'approximation dont on peut calculer le rapport.

Définition 3. $\mathcal{A}$ est une $f(k)$-*approximation partielle* si cet algorithme renvoie une solution (partielle ou réalisable) T vérifiant $d(T) \leq f(k) \cdot \frac{\omega(T^*)}{k}$.

Lemme 2.2.2 ([CCCD98]). *Si $\mathcal{A}$ est une $f(k)$-approximation partielle, et si $\frac{f(k)}{k}$ est une fonction décroissante de k, alors l'algorithme appliquant $\mathcal{A}$ itérativement jusqu'à construire une solution couvrant tous les terminaux est une approximation de rapport $\left(\int_0^k \frac{f(u)}{u} du\right)$.*

On peut utiliser ce lemme pour redémontrer que l'algorithme des plus courts chemins est une k-approximation.

Corollaire 2.2.3. *L'algorithme des plus courts chemins est une k-approximation.*

Démonstration. L'algorithme des plus courts chemins renvoie successivement les plus courts chemins reliant la racine à chaque terminal. Dans ce cas-ci, $\mathcal{A}$ est donc l'algorithme qui renvoie un plus court chemin de la racine vers un terminal arbitraire x. Toute solution optimale T^* contient un chemin de la racine vers x. Donc le poids d'un plus court chemin $P(r,x)$ reliant r à x est plus petit que le poids d'une solution optimale : $\omega(P(r,x)) \leq \omega(T^*)$.

En remarquant que $d(P(r,x)) = \omega(P(r,x))$ et que $\omega(T^*) = k \cdot \frac{\omega(T^*)}{k}$, on en déduit que $\mathcal{A}$ est une $f(k) = k$-approximation partielle. D'après le lemme 2.2.2, l'algorithme des plus courts chemins est une approximation de rapport $\left(\int_0^k \frac{u}{u} du\right) = k$. □

Remarque 1. La démonstration du lemme 2.2.2 de [CCCD98] se fait par récurrence sur k. Cette récurrence reste vraie si on cherche à prouver que répéter $\mathcal{A}$ est une $\left(f(1) + \int_1^k \frac{f(u)}{u} du\right)$-approximation. Ce résultat est plus intéressant car si $\mathcal{A}$ est une c-approximation partielle où c est une constante, alors en répétant $\mathcal{A}$, on obtient une $c \cdot (\log(k) + 1)$-approximation pour DST. Les résultats d'inapproximabilité qui suivent dans ce chapitre montrent que, sous certaines hypothèses de complexité, une telle approximation n'existe pas. Ainsi, sous ces mêmes hypothèses, il n'est pas possible de construire une c-approximation partielle polynomiale pour le problème de Steiner.

Chercher des arbres de profondeur bornée. Plaçons-nous en premier lieu dans l'instance des plus courts chemins $\mathcal{I}^{\triangleright}$ décrite en définition 1, page 16. Nous avons vu dans le théorème 2.1.2, que cette instance a les mêmes propriétés d'approximation que $\mathcal{I}$. On nomme donc T^* une solution optimale de $\mathcal{I}^{\triangleright}$.

L'idée de rechercher des arbres de profondeur bornée a été émise pour la première fois dans [Zel97]. L'auteur démontre que si T_l^* est une solution réalisable de profondeur au plus l de poids minimum (parmi toutes les solutions de profondeur au plus l), alors il est possible de borner $\frac{\omega(T_l^*)}{\omega(T^*)}$.

Lemme 2.2.4 ([Zel97], corrigé dans [HRZ01] [3]). $\frac{\omega(T_l^*)}{\omega(T^*)} \leq 2l \cdot (\frac{k}{2})^{\frac{1}{l}}$

Remarque 2. Dans $\mathcal{I}^{\triangleright}$, la solution renvoyée par l'algorithme des plus courts chemins est une étoile reliant r à chaque terminal, soit un arbre de profondeur 1. Appliquer ce lemme à cet algorithme nous redémontre encore une fois qu'il s'agit d'une k-approximation.

Remarque 3. Nous le verrons plus loin dans ce chapitre, construire T_l^* est un problème NP-difficile si $l \geq 2$.

3. Le résultat de [Zel97] est $\frac{\omega(T_l^*)}{\omega(T^*)} \leq k^{\frac{1}{l}}$. Ce résultat s'avère inexact [HHT06], et a été modifié dans [HRZ01]

Algorithme de Charikar L'idée de l'algorithme de Charikar est de trouver des arbres de faible densité en limitant la recherche à des arbres de profondeur bornée, puis de répéter ce processus jusqu'à ce que tous les terminaux soient couverts. Ils démontrent ainsi le théorème suivant.

Théorème 2.2.5 ([CCCD98], version corrigée [4]). *Pour tout entier l, l'algorithme de Charikar avec comme paramètre d'entrée l est une approximation de rapport $2^{1-\frac{1}{l}} \cdot l^2 \cdot (l-1) \cdot (k^{\frac{1}{l}}) = O(k^{\frac{1}{l}})$ pour le problème de l'arborescence de Steiner et a une complexité en temps $O(n^l k^{2l})$.*

Définition 4. La *profondeur* d'une instance est la distance maximale, en nombre d'arcs entre la racine et un terminal.

[Eve07] évoque la possibilité d'améliorer le théorème 2.2.5 quand la profondeur H de la solution optimale est bornée. L'algorithme de Charikar avec en paramètre d'entrée un entier l construit à chaque itération un arbre de densité d vérifiant $d \leq (l-1) \cdot d_l^*$ où d_l^* est la densité de T_l^*. D'après la remarque 1, en page 19, on montre le théorème suivant.

Théorème 2.2.6 ([Eve07]). *Si H est la profondeur de l'instance, l'algorithme de Charikar avec en paramètre d'entrée l'entier H est une $H(\log(k)+1)$-approximation pour le problème de l'arborescence de Steiner.*

On peut citer plusieurs travaux en rapport avec cet algorithme. L'algorithme de Roos [Roo01] est une implémentation plus rapide de l'algorithme de Charikar quand $l = 2$, et l'algorithme FasterDSP [HHT06] est une implémentation plus rapide quel que soit l (mais conserve *a priori* les mêmes temps de calcul que [Roo01] ou que l'algorithme des plus courts chemins quand $l \leq 2$). Dans [HF12], l'algorithme est étendu de sorte qu'on ne se limite plus aux arbres de hauteur l, mais on se limite aux arbres ne contenant aucun chemin de taille $l+1$ sans terminal. Cet algorithme ne donne pas de meilleure garantie de performance dans le cas général, mais il est nécessairement au moins aussi bon que l'algorithme de Charikar. En particulier, si le chemin le plus long du graphe induit par $V \backslash X$ contient m' arcs alors cet algorithme est une $(m'+1)\log(k)$-approximation.

2.2.2 Représentation primale-duale du problème de Steiner et ascension du dual

Il existe deux représentations naturelles du problème de Steiner sous forme de programme linéaire. Les deux associent une variable $x_a \in \{0,1\}$ à chaque arc a du graphe selon qu'il est choisi ou non dans la solution réalisable que le programme renvoie. Ces deux représentations donnent les mêmes solutions optimales réelles et entières [Won84], elles ont donc le même saut d'intégrité et fournissent les mêmes capacités d'approximation de DST.

Description par flots conjoints La première représentation construit, pour chaque terminal, un flot reliant la racine à ce terminal. Les flots de deux terminaux peuvent se croiser. Chaque variable x_a est égale au maximum des flots qui le traverse. Dans le programme (PP1), $\Gamma^-(v)$ et $\Gamma^+(v)$ décrivent respectivement les arcs entrant et sortant du nœud v.

4. Le théorème donné ici n'est pas identique à celui présenté dans [CCCD98]. En effet, leur démonstration de ce théorème utilise les résultats erronés donnés dans [Zel97] et non pas les résultats corrigés dans [HRZ01].

Programme linéaire (PP1) : Description de DST par flots conjoints

Minimiser	$\sum_{a\in A} \omega(a)\cdot x_a$			
s.c.	$\sum_{a\in\Gamma^+(r)} f_a^t - \sum_{a\in\Gamma^-(r)} f_a^t$	$=$	1	pour $t \in X$
	$\sum_{a\in\Gamma^+(t)} f_a^t - \sum_{a\in\Gamma^-(t)} f_a^t$	$=$	-1	pour $t \in X$
	$\sum_{a\in\Gamma^+(v)} f_a^t - \sum_{a\in\Gamma^-(v)} f_a^t$	$=$	0	pour $t \in X, v \in V\backslash X \cup \{r\}$
	f_a^t	$\leq$	x_a	pour $t \in X, a \in A$
	f_a^t	$\in$	$[0,1]$	pour $t \in X, a \in A$
	x_a	$\in$	$\{0,1\}$	pour $a \in A$

Ce programme linéaire a inspiré deux travaux liés à l'approximation polynomiale du problème.

Dans [Rot11], les auteurs réduisent le saut d'intégrité du programme (PP1) en utilisant la relaxation de Lasserre. Cette relaxation permet, entre autres, d'obtenir un nouveau problème dans lequel la solution optimale entière est toujours la même, mais où la solution optimale réelle est plus proche de l'entière. L'utilisation de cette relaxation peut être itérée autant de fois que souhaité pour continuer à réduire le saut d'intégrité. Leur algorithme est une $O(k^\varepsilon)$-approximation, quel que soit $\varepsilon > 0$.

Dans [ZK02], (PP1) a été modifié pour être adapté à la recherche d'arbres de faible densité. Dans le cas où le graphe privé de ses terminaux est un arbre de hauteur $H-1$, leur algorithme est une $H\log(k)$-approximation. Ils précisent que leur programme linéaire a un saut d'intégrité égal à $\Omega(\sqrt{k})$ [5].

On peut également citer [BBP03] qui étudie une relaxation lagrangienne de ce problème pour construire une heuristique, mais celle-ci ne fournit aucune garantie de performance *a priori*.

Description par coupes Dans cette version, on regarde toutes les coupes du graphe séparant la racine d'au moins un terminal. Chaque coupe S doit être traversée par au moins un arc, partant de la racine vers le terminal. Dans le programme qui suit, $\Gamma^-(S)$ décrit les arcs entrant dans l'ensemble de nœuds de S, c'est-à-dire les arcs dont l'origine est hors de S et la destination est dans S.

Programme linéaire (PP2) : Description de DST par coupes

Minimiser	$\sum_{a\in A} \omega(a)\cdot x_a$			
s.c.	$\sum_{a\in\Gamma^-(S)} x_a$	$\geq$	1	pour $S \subseteq V$ tel que $r \notin S$ et $S \cap X \neq \emptyset$
	x_a	$\in$	$\{0,1\}$	pour $a \in A$

5. Ce saut d'intégrité a été plusieurs fois attribué à tort au programme (PP1) [HKK+07, Rot11]

L'étude de ce programme linéaire a permis de montrer que le saut d'intégrité des problèmes (PP1) et (PP2) est de l'ordre de $\Omega(\frac{\log^2(n)}{\log\log(n)^2})$ mais aussi de l'ordre de $\Omega(\log^2(k))$ [HKK+07].

Plusieurs travaux d'approximation sont basés sur le programme (PP2). Ces techniques sont dites d'*ascension du dual*. Le programme dual associé à la relaxation continue de (PP2) est (DP3).

Programme linéaire (DP3) : Description de DST par coupes (programme dual)

$$\textbf{Maximiser} \quad \sum_{\substack{S\subseteq V \\ r\notin S \\ S\cap X\neq\emptyset}} y_S$$

$$\textbf{s.c.} \quad \sum_{a\in\Gamma^-(S)} y_S \leq \omega(a) \qquad \text{pour } a \in A$$

$$y_S \geq 0 \qquad \text{pour } \begin{array}{l} S \subseteq V \text{ tel que} \\ r \notin S \text{ et } S \cap X \neq \emptyset \end{array}$$

Remarque 4. Lorsque les poids des arcs sont unitaires et lorsque les variables y_S sont entières, ce programme cherche un ensemble maximal de coupes disjointes séparant la racine d'au moins un terminal.

La contrainte associée à un arc a est dite saturée si on ne peut plus augmenter les variables duales associées à cette contrainte. Deux heuristiques d'ascension du dual existent pour DST [Won84, Mel07]. L'idée est de choisir un ou plusieurs ensembles S, et d'augmenter les variables y_S jusqu'à saturation d'une contrainte associée à un arc a. On ajoute a à la solution que l'on construit et on recommence jusqu'à ce que la solution soit réalisable.

L'augmentation d'une variable duale y_S peut être vue comme la construction d'un flot qui remplit les arcs entrant dans S. Un arc peut contenir un flot égal à son poids.

Les deux heuristiques premièrement n'augmentent que des variables y_S où S est un ensemble de nœuds reliés à au moins un terminal par des arcs dont la contrainte est saturée ; secondement n'augmentent que des variables y_S où S est maximal par inclusion : si $S' \subset S$ et qu'il est possible d'augmenter y_s, alors l'heuristique n'augmente pas y'_S.

Le flot s'étend donc dans les arcs depuis les terminaux juqu'à la racine.

L'heuristique dans [Won84] choisit, à chaque itération, un ensemble S vérifiant la seconde hypothèse arbitrairement (il en existe au plus k) et augmente y_S jusqu'à saturation d'une contrainte. L'auteur ne donne pas de garantie de performance pour son algorithme.

L'heuristique dans [Mel07] choisit tous les ensembles S vérifiant la seconde hypothèse et augmente leurs variables toutes en même temps jusqu'à saturation d'une contrainte. L'auteur précise que son algorithme est une $O(k)$-approximation pour le problème de Steiner.

Remarque 5. Une méthode envisageable serait de choisir l'ensemble S vérifiant la seconde hypothèse minimisant l'augmentation y_S, on a ainsi moins de chance de tomber dans un minimum local. Mais cette dernière a le défaut de suivre les chemins d'arcs de poids faibles. C'est un défaut car on peut alors facilement tromper l'algorithme : si on transforme un arc de fort poids en un chemin constitué de nombreux arcs de faible poids, on favorise alors la saturation des contraintes de ces arcs.

2.3 Résultats d'inapproximabilité

Nous allons dans un premier temps donner la réduction qui permet de démontrer que DST n'est pas approximable avec un rapport meilleur que $\log(k)$. Nous réutiliserons cette réduction (ou une variante) à plusieurs reprises dans le manuscrit.

Cette réduction se fait depuis le problème de couverture par ensembles :

Problème : Problème de couverture par ensembles

Instance :
- Un ensemble X d'*éléments*, appelé *univers*.
- Un ensemble $\mathcal{S}$ de parties de X.

Solution réalisable : Une couverture $C \subset \mathcal{S}$ de X.

Optimisation : Minimiser $|C|$.

Théorème 2.3.1 ([RS97]). *Si $P \neq NP$, alors il existe une constante $c > 0$ tel qu'il n'existe pas de $c \cdot \log(k)$-approximation pour le problème de couverture par ensembles.*

Théorème 2.3.2 ([Fei98]). *Si $NP \not\subseteq TIME(n^{O(\log\log(n))})$, alors pour tout $\varepsilon > 0$, il n'existe pas de $\log(k)(1-\varepsilon)$-approximation pour le problème de couverture par ensembles.*

Théorème 2.3.3. *S'il existe une α-approximation pour le problème de Steiner, alors il existe une α-approximation pour le problème de couverture par ensembles.*

Démonstration. Soit $\mathcal{I}$ une instance de la couverture par ensembles, nous allons la transformer en instance $\mathcal{I}' = (G, r, X, \omega)$ de DST. Cette transformation, dont un exemple est donné en figure 2.3, se fait en :
- ajoutant un terminal dans $\mathcal{I}'$ pour chaque élément de l'univers (on confond les deux sous la dénomination terminal-élément) ;
- ajoutant un nœud dans $\mathcal{I}'$ pour chaque ensemble de $\mathcal{S}$ (on confond les deux sous la dénomination nœud-ensemble) ;
- ajoutant une racine r que l'on relie à tous les nœuds-ensembles avec un arc de poids 1 ;
- reliant chaque nœud-ensemble aux terminaux-éléments qu'il contient avec un arc de poids nul [6].

Soit C une solution réalisable de $\mathcal{I}$, alors on construit une solution réalisable T de $\mathcal{I}'$ de poids $|C|$ en sélectionnant tous les arcs reliant r aux nœuds-ensembles de C, et en reliant ces nœuds aux terminaux-éléments qu'ils contiennent (s'ils sont couverts pas plus d'un ensemble, l'un des deux est choisi arbitrairement). Inversement, si T est une solution réalisable de $\mathcal{I}$ de poids $\omega(T)$, alors l'ensemble C contenant tous les nœuds-ensembles de T est une solution réalisable de $\mathcal{I}$. Elle contient exactement $\omega(T)$ ensembles.

6. Il est possible d'effectuer cette réduction avec des arcs de poids unitaire, en remplaçant les arcs reliant la racine aux nœuds-ensembles par un chemin de longueur $H-1$ et les ensembles aux terminaux par un arc de poids unitaire. Si la valeur de H est suffisamment élevé (cette valeur dépend de k), alors la réduction est toujours isofacteur.

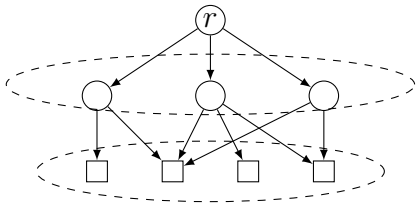


FIGURE 2.3 – *Exemple de réduction depuis une instance du problème de couverture d'ensemble vérifiant* $X = \{x_1, x_2, x_3, x_4\}$ *et* $S = \{\{x_1, x_2\}, \{x_2, x_3, x_4\}, \{x_2, x_4\}\}$

Donc s'il existe une α-approximation pour DST, il existe une α-approximation pour le problème de couverture par ensembles. □

Corollaire 2.3.4. *Si* $P \neq NP$*, alors il existe une constante* $c > 0$ *tel qu'il n'existe pas de* $c \cdot \log(k)$*-approximation pour le problème de couverture par ensembles.*

Démonstration. D'après les Théorèmes 2.3.1 et 2.3.3 □

Corollaire 2.3.5. *Si* $NP \not\subseteq TIME(n^{O(\log\log(n))})$*, alors pour tout* $\varepsilon > 0$*, il n'existe pas de* $\log(k)(1-\varepsilon)$*-approximation pour le problème de Steiner.*

Démonstration. D'après les Théorèmes 2.3.2 et 2.3.3. □

Remarque 6. La réduction du théorème 2.3.3 montre également que :

- DST n'est pas $\log(k)$-approchable même si le graphe est sans circuits et que la distance maximale séparant la racine des terminaux est 2, il est donc NP-difficile de construire une arborescence profondeur au plus l de poids minimum (parmi toutes les solutions de profondeur au plus l) si $l \geq 2$, ce qui rejoint la remarque 3, en page 19 ;
- DST est W[2]-Difficile vis-à-vis de deux paramètres : le poids de la solution optimale et le nombre de nœuds non terminaux dans cette solution. En effet, le problème de Couverture par ensemble est W[2]-Difficile vis-à-vis du poids de la solution optimale [**?**] et ce paramètre est égal, dans notre réduction, au poids de la solution optimale du problème de Steiner et du nombre de nœuds non terminaux de cette solution.

En utilisant le théorème de répétition parallèle [Raz98] avec le théorème PCP [AL97, ALM98], Halperin et Krauthgamer étendent ce résultat.

Théorème 2.3.6 ([HK03])**.** *Si* $NP \not\subseteq ZTIME(n^{O(\text{polylog}(n))})$*, alors pour tout* $\varepsilon > 0$*, il n'existe pas de* $\log^{2-\varepsilon}(k)$*-approximation pour le problème de Steiner.*

Remarque 7. En réalité, la preuve dans [HK03] démontre qu'il est impossible de trouver une approximation de rapport $H\log(k)$ où H est la profondeur d'une solution optimale, quand $H \leq \log^{1-\varepsilon}(k)$. Quand H augmente au-delà de cette borne, la borne supérieure stagne à $\log^{2-\varepsilon}(k)$.

Remarque 8. La preuve de ce théorème est faite dans le cas où l'instance contient un arbre de hauteur $H-1$, où les terminaux sont tous successeurs des feuilles de cet arbre. Donc leur réduction montre également que DST n'est pas $H\log(k)$-approchable même si le graphe est sans circuits et que la distance maximale séparant la racine des terminaux est H.

2.4 Récapitulatif

Dans le cas général, il n'existe pas d'algorithme dont le rapport d'approximation soit constant. Celui-ci dépend donc des paramètres de l'instance. Deux paramètres sont généralement utilisés pour décrire le rapport d'approximation d'un algorithme : le nombre de terminaux k et la profondeur H d'une solution optimale, soit la distance maximale, en nombre d'arcs, séparant la racine d'un terminal. On recherche donc, en fonction des valeurs de k et de H, le plus petit rapport d'approximation polynomiale $\rho(k, H)$ du problème de l'arborescence de Steiner. Une synthèse des résultats présentés dans ce chapitre est donnée en figure 2.4, décrivant le plus petit rapport d'approximation connu, autrement dit la plus petite borne supérieure de $\rho(k, H)$ et la plus haute borne inférieure de $\rho(k, H)$.

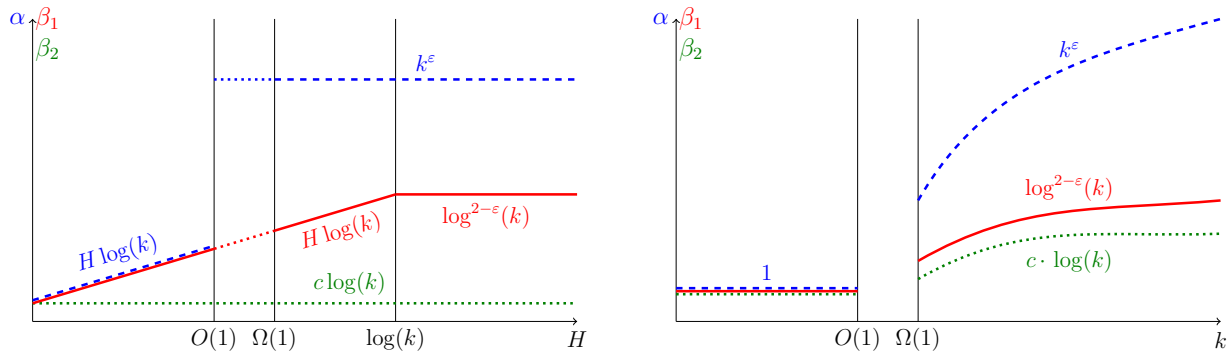


FIGURE 2.4 – *Ces deux graphiques tracent en tirets bleus l'allure de α, le plus petit rapport d'approximation connu du problème de l'arborescence de Steiner. Ils tracent en trait plein rouge l'allure de β_1, la plus haute borne inférieure connue de $\rho(k, H)$ si $NP \not\subset ZTIME(n^{\text{polylog}}(n))$ et en pointillés verts, l'allure de β_2, la plus haute borne inférieure connue de $\rho(k, H)$ si $NP \neq P$. À gauche, on suppose que le paramètre k est quelconque, et que le paramètre H varie. À droite, on suppose que le paramètre H est quelconque et que le paramètre k varie. Dans les deux cas, on fait la distinction entre un paramètre borné ou non borné par une constante.*

La figure 2.4 rappelle en particulier que, quand k est une constante, on connait un algorithme exact polynomial pour le problème de Steiner. On peut par exemple utiliser l'algorithme de Dreyfus-Wagner que nous avons détaillé dans le chapitre 1.

Dans le cas des graphes sans circuits et des graphes complets avec inégalité triangulaire, les résultats d'approximabilité et d'inapproximabilité sont identiques à ceux du cas général.

Chapitre 3.

Problèmes connexes au problème de l'arborescence de Steiner

Ce chapitre répertorie un ensemble de problèmes connexes au problème de l'arborescence de Steiner (DST) ainsi que leurs résultats d'approximabilité polynomiale. Nous nous restreignons aux problèmes dont les résultats théoriques sont en lien avec ceux de DST.

Nous les séparons en deux catégories : les problèmes de Steiner et les problèmes de couverture par ensemble.

3.1 Problèmes de Steiner

Un compendium listant des problèmes de Steiner est disponible à l'adresse suivante : `http://theory.cs.uni-bonn.de/info5/steinerkompendium`. Nous nous restreignons ici à donner les résultats des 3 principaux problèmes de Steiner.

Le problème de l'arbre de Steiner.

Problème : Problème de l'arbre de Steiner (UST)

Instance :

- Un graphe non orienté $G = (V, E)$.
- Un ensemble de k nœuds $X \subset V$ appelés *terminaux*.
- Une fonction de pondération $\omega : E \to \mathbb{R}^+$.

Solution réalisable : Un arbre $T \subset G$ couvrant tous les terminaux de X.

Optimisation : Minimiser le poids $\omega(T) = \sum_{e \in T} \omega(e)$.

Une instance du problème de l'arbre de Steiner peut être réduite en instance du problème de Steiner en transformant toute arête en deux arcs symétriques de même poids, et en choisissant pour racine un terminal arbitrairement.

UST peut être approché avec un rapport constant. Une variante de l'algorithme des plus courts chemins pour DST, décrit en page 17 du chapitre 2, donne un rapport égal à

2 [Cho78, KMB81]. Cet algorithme calcule tous les chemins de poids minimum entre les terminaux et renvoie un arbre couvrant de poids minimum du graphe restreint à ces chemins.

La première approximation dont le rapport passe en dessous de 2 est celle de Zelikowsky, de rapport $\frac{11}{6}$ [Zel93]. Cette approximation commence par calculer le graphe des plus courts chemins $G^{\triangleright}$ où tout couple de nœuds est relié par une arête de poids égal au poids du plus court chemin reliant ses extrémités dans G. Puis il calcule dans $G^{\triangleright}$ une 2-approximation avec l'algorithme des plus courts chemins (cette solution est identique à celle qu'il aurait obtenu dans le graphe d'origine G). Il vérifie ensuite s'il n'est pas plus rentable, pour chaque triplet (x, y, z) de terminaux reliés par les arcs (x, y) et (y, z) dans la solution, de remplacer ces deux arcs par une étoile $(x, v), (y, v), (z, v)$ où v est un nœud du graphe. Cet algorithme désigne alors des nœuds *intéressants*. L'ensemble de ces nœuds est ajouté à X. Appliquer l'algorithme des plus courts chemins à cette nouvelle instance permet d'obtenir une $\frac{11}{6}$-approximation.

L'idée naturelle qui suit consiste à ne plus travailler avec des triplets de terminaux, mais avec des tuples de tailles plus élevée. Une série de travaux sur cette idée a permis l'élaboration d'une $1 + \frac{\log(3)}{2} + \varepsilon < 1.55$-approximation [RZ00].

Il est possible de généraliser ces approximations au problème DST. Par exemple, la généralisation de la $\frac{11}{6}$-approximation de Zelikowsky consiste à vérifier, pour tout couple (x, y) de terminaux, s'il n'est pas plus rentable, dans la solution renvoyée par l'algorithme des plus courts chemins, de remplacer les deux chemins reliant la racine r à x et y par une arborescence de Steiner optimale couvrant x et y. Nous avons vu dans le chapitre 1 que l'algorithme de Dreyfus-Wagner calcule cette arborescence en temps polynomial. Mais si cette technique permet de diminuer le rapport d'approximation dans le cas d'un graphe non orienté, elle ne le peut quand le graphe est orienté : le rapport d'approximation de toutes ces généralisations est k. Ceci est dû à la dissymétrie provoquée par l'orientation des arcs, mais aussi par la présence de la racine qui joue le rôle d'un terminal particulier dans l'instance.

Le plus faible rapport d'approximation que l'on connaisse pour UST est $\log(4) + \varepsilon < 1.39$ pour tout $\varepsilon > 0$ suffisamment petit. Cet algorithme d'approximation utilise le programme linéaire par coupes (PP2) décrit en page 21 du chapitre 2.

Sous réserve que $P \neq NP$, la plus haute borne inférieure du rapport d'approximabilité de UST est $\frac{96}{95}$ [CC08].

Le problème de couverture de groupes par arbre de Steiner.

Problème : Problème de couverture de groupes par arbre de Steiner (GST)

Instance :
- Un graphe non orienté $G = (V, E)$.
- Un ensemble de k groupes de nœuds $g_1, g_2, \ldots, g_k \subset V$.
- Une fonction de pondération $\omega : E \to \mathbb{R}^+$.

Solution réalisable : Un arbre $T \subset G$ couvrant au moins un nœud de chaque groupe.

Optimisation : Minimiser le poids $\omega(T) = \sum_{e \in T} \omega(e)$.

Une instance de GST peut être réduite en instance de DST en remplaçant tout arête par deux arcs de sens opposés et de même poids, puis en créant pour chaque groupe g_i un terminal

x_i auquel est relié chaque nœuds de g_i par un arc de poids nul. Si $NP \not\subset ZIME(n^{\text{polylog(n)}})$, le problème GST ne peut être approché avec un rapport $\log^{2-\varepsilon}(k)$ [HK03]. La réduction étant isofacteur, c'est cette propriété qui permet de démontrer le même résultat d'inapproximabilité pour DST.

Il existe une version orientée du problème GST : le *problème de couverture de groupes par arborescence de Steiner* où on cherche à relier une racine à au moins un nœud de chaque groupe. Ce problème est équivalent à DST dans le sens où il existe une réduction isofacteur de DST vers ce problème et inversement.

Ce résultat d'inapproximabilité est vrai même si l'instance est un arbre et si les groupes ne contiennent que des feuilles de cet arbre. À l'inverse, il existe, pour ce type d'instance, une $O(\log(\max(|g_i|))\log(k))$-approximation probabiliste [GKR98]. A partir de la $O(\log(\max(|g_i|))\log(k))$-approximation dans un arbre, il est possible de construire une approximation probabiliste de rapport $O(\log(n)\log\log(n)\log(\max(|g_i|))\log(k))$ pour le cas général [GKR98]. Des algorithmes déterministes ont ensuite été générés à partir de cette approximation [CCGG98, Sri01], offrant ainsi une $O(\log(n)\log(k))$-approximation dans les arbres et $O(\log^2(n)\log\log(n)\log(k))$-approximation dans le cas général.

Le cas de l'arbre est donc presque clos si $NP \not\subset ZIME(n^{\text{polylog(n)}})$ puisque le plus petit rapport d'approximation que l'on puisse espérer se situe entre $O(\log^2(k))$ et $O(\log(n)\log(k))$.

Remarque 9. Il est possible de réduire ces instances en instances de DST en :
- choisissant une racine au hasard parmi les nœuds internes de l'arbre ;
- orientant les arêtes de la racine vers les feuilles ;
- ajoutant un terminal par groupe et en reliant les feuilles de ce groupe à ce terminal par des arcs de poids nul.

Cette réduction est isofacteur. Ces instances presque arborescentes sont utilisées dans [RN10], pour minimiser le temps de calcul de reconnaissance d'objets dans une image. L'objectif est de savoir en un minimum de temps, pour chaque objet (par exemple une tasse ou un téléviseur), s'il est présent dans l'image ou non.

Le problème de l'arbre de Steiner avec des nœuds pondérés.

Problème : Problème de l'arbre de Steiner avec des nœuds pondérés.

Instance :
- Un graphe non orienté $G = (V, E)$.
- Un ensemble de k nœuds $X \subset V$ appelés *terminaux*.
- Une fonction de pondération $\omega : V \to \mathbb{R}^+$.

Solution réalisable : Un arbre $T \subset G$ couvrant tous les terminaux de X.

Optimisation : Minimiser le poids $\omega(T) = \sum_{v \in T} \omega(v)$.

Ce problème peut être vu comme une généralisation du problème UST. En effet, il est possible de réduire toute instance de UST en instance de ce problème en insérant un nœud au milieu de chaque arête. Le poids de cette arête est alors déporté sur ce nœud. Les autres nœuds ont un poids nul.

Il est lui-même réductible à DST. On crée pour chaque nœud v un double v', relié aux même nœuds que v. Les arêtes reliés à v sont orientées vers v et les arêtes reliés à v' sont orientées depuis v'. On relie enfin v à v' par un arc de poids $\omega(v)$. Chaque terminal t de l'instance d'origine perd sa qualité de terminal au profit de t'.

Ces deux réductions isofacteurs peuvent être adaptées pour montrer que la version orientée de ce problème, le problème de l'arborescence de Steiner avec des nœuds pondérés, est équivalente à DST.

Dans le cas général, sauf si $P = NP$, il n'existe pas de $(1-\varepsilon)\log(k)$-approximation pour le problème de l'arbre de Steiner avec des nœuds pondérés, quel que soit $\varepsilon > 0$ [GK99]. Il existe néanmoins une $O(\log(k))$-approximation pour ce problème [GK99]. L'approximabilité de ce problème est donc close.

Dans le cas planaire, il existe une 6-approximation pour ce problème. Ce résultat peut être étendu à l'existence d'une approximation de rapport $c(|G'|)$ (où $|G'|$ est le nombre de sommet de G') pour toute instance n'ayant pas le graphe G' pour mineur [DHK09]. Les auteurs de ce résultat envisagent une extension pour DST mais celle-ci est encore une question ouverte.

3.2 Problèmes de couverture par ensembles

Le problème de couverture par ensembles.

Problème : Problème de couverture par ensembles

Instance :
- Un ensemble X de k *éléments*, appelé *univers*.
- Un ensemble $\mathcal{S}$ de parties de X.

Solution réalisable : Une couverture $C \subset \mathcal{S}$ de X.

Optimisation : Minimiser $|C|$.

Le problème de couverture par ensembles peut être vu comme un sous-problème de DST, en utilisant la réduction décrite en page 23 du chapitre 2, où la distance maximale séparant la racine d'un terminal est 2.

Si $NP \not\subset TIME(n^{O(\log\log(n))})$, il n'existe pas de $(1-\varepsilon)\log(k)$-approximation pour ce problème, quel que soit $\varepsilon > 0$ [Fei98]. Si $P \neq NP$, il existe une constante $c > 0$ telle qu'il n'existe pas de $c\log(k)$ approximation pour ce problème [RS97]. Il existe néanmoins une $O(\log(k))$ approximation pour ce problème [Vaz01].

Cette approximation peut être étendue en une $1 + \log(r)$ approximation exponentielle, dont la complexité en temps est $O(c^{\frac{k}{r}})$, pour tout $r > 1$ et tout algorithme exact de complexité en temps $O(c^k)$ [CKW09]. Nous reviendrons sur cette approximation dans le chapitre 6, pour créer une approximation exponentielle pour le problème de Steiner.

Le problème de couverture d'éléments rouges et bleus par ensembles.

Problème : Problème de couverture d'éléments rouges et bleus par ensembles[1]

Instance :

- Un ensemble $\mathcal{B}$ de k_B *éléments bleus*, appelé *univers bleu*.
- Un ensemble $\mathcal{R}$ de k_R *éléments rouges*, appelé *univers rouge*.
- Un ensemble $\mathcal{S}$ de parties de l'*univers* $X = \mathcal{B} \cup \mathcal{R}$.

Solution réalisable : Une couverture $C \subset \mathcal{S}$ de $\mathcal{B}$.

Optimisation : Minimiser $|C \cap \mathcal{R}|$, soit le nombre d'éléments rouges couverts par C.

Ce problème est une généralisation du problème de couverture par ensembles, dans lequel tous les éléments de base sont bleus et où chaque ensemble contient un unique élément rouge supplémentaire.

Posons $k = k_B + k_R$ le nombre d'éléments de X. Sauf si $P = NP$, il n'existe pas de $O(2^{\log^{1-\varepsilon}(k)})$-approximation pour tout $\varepsilon > 0$. À l'inverse, il existe une approximation de rapport $2\sqrt{k \log(k_B)}$.

Il est possible de voir ce problème comme une généralisation des problèmes GST et DST [CDKM00], mais la réduction employée n'est pas polynomiale. Dans DST, chaque terminal est remplacé par un élément bleu, et chaque arc est remplacé par un élément rouge. Enfin, chaque chemin reliant la racine à un terminal est remplacé par un ensemble contenant tous les éléments rouges correspondants aux arcs de ce chemin et tous les éléments bleus correspondants aux nœuds couverts par ce chemin.

Cette réduction est exponentielle en la taille du graphe, mais est isofacteur et donne une idée de la réduction inverse : associer un terminal pour chaque élément bleu, et un nœud pour chaque ensemble. Chaque ensemble doit être relié à la racine avec un unique chemin partageant ses arcs avec les chemins des autres ensembles, de la même manière qu'il partage des éléments rouges avec ces ensembles. Une telle réduction, si elle existe et est polynomiale, prouverait que le problème de Steiner n'est pas approximable avec un rapport $O(2^{\log^{1-\varepsilon}(k)})$ si $NP \not\subset ZTIME(n^{\text{polylog(n)}})$.

1. *Red Blue Set Cover*, en anglais.

Deuxième partie

Algorithmes d'approximation pour le problème de Steiner

Cette partie présente trois résultats d'approximation pour le problème de Steiner. Le premier chapitre présente deux approximation efficaces pour le cas général qui utilisent à la fois la technique d'ascension du dual et la recherche d'arbre de faible densité. Le second résultat est une approximation polynomiale dans un cas particulier d'un graphe découpé en paliers de nœuds, ceux-ci n'étant reliés qu'aux nœuds du palier suivant. Le dernier chapitre présente une approximation exponentielle qui mêle algorithme d'approximation glouton et algorithme exact.

Chapitre 4.

Approximations gloutonnes pour le cas général

Dans le chapitre 2, nous avons évoqué le meilleur rapport d'approximation polynomiale existant aujourd'hui pour le problème de l'arborescence de Steiner : $O(k^\varepsilon)$, donné par l'algorithme de Charikar [CCCD98]. Ce rapport provient de deux notions introduites par les définitions 2 et 3 en page 18 : la recherche d'arbres de bonne *densité* et la construction d'une *approximation partielle* de faible rapport.

On considère comme donnée une instance $\mathcal{I} = (G, r, X, \omega)$ du problème de Steiner. On notera $X(T)$ l'ensemble des terminaux de X couverts par une arborescence T enracinée en r, et $d(T) = \frac{\omega(T)}{|X(T)|}$ sa densité.

Ce chapitre s'intéresse à une heuristique pour trouver des arbres de faible densité afin de construire une approximation efficace du problème de Steiner. Nous allons utiliser la technique dite d'*ascension du dual*, présentée dans le chapitre 2 en page 20.

Les heuristiques d'ascension du dual tentent d'augmenter la solution duale de l'instance petit à petit jusqu'à saturation complète : l'instant où plus aucune variable duale ne peut être augmentée sans transgresser les contraintes du programme linéaire. À chaque itération, un ou plusieurs arcs sont ajoutés à la solution réalisable du primal. La question est de déterminer comment augmenter la solution duale. Par quelle variable commencer et que faire quand cette variable ne peut plus être augmentée ?

Nous décrivons dans ce chapitre une expérience de pensée qui représente, nous le verrons plus loin, une manière d'augmenter les variables duales. Contrairement aux heuristiques d'ascension du dual de [Mel07, Won84], notre algorithme ne renvoie pas une solution réalisable de DST, mais un arbre de bonne densité. Cette expérience sera ensuite décrite sous forme d'un algorithme, nommé FLAC, dont nous pourrons donner des garanties de performance. FLAC renvoie une solution partielle de bonne densité, et permet donc de construire un algorithme d'approximation glouton, nommé Greedy$_{FLAC}$, pour le problème de l'arborescence de Steiner, en utilisant l'algorithme 2.

En utilisant le lemme 2.2.2 (chapitre 2, page 19) nous serons en mesure de prouver que le rapport d'approximation de Greedy$_{FLAC}$ est k. Dans le but d'obtenir un meilleur rapport d'approximation, il est possible de travailler non pas dans l'instance $\mathcal{I}$ mais dans l'instance des plus courts chemins $\mathcal{I}^\triangleright$, vue dans le chapitre 2, en page 16.

Algorithme 2 Algorithme d'approximation glouton utilisant FLAC (Greedy_{FLAC})

Entrées : Une instance $\mathcal{I} = (G = (V, A), r, X, \omega)$ pour le problème de Steiner.
Sorties : T, une solution réalisable de $\mathcal{I}$

1: $T \leftarrow \emptyset$
2: **Tant que** $X \neq \emptyset$ **Faire**
3: $\quad T_0 \leftarrow FLAC(G, r, X, \omega)$
4: $\quad T \leftarrow T \cup T_0$
5: $\quad X \leftarrow X \backslash X(T_0)$
Renvoyer T

Nous avons vu dans le théorème 2.1.2, en page 16, que cette instance a les mêmes propriétés d'approximation que $\mathcal{I}$. À partir d'une solution réalisable $T^{\triangleright}$ de $\mathcal{I}^{\triangleright}$, il est possible de construire une solution réalisable de $\mathcal{I}$ de plus petit poids en sélectionnant pour tout arc (u, v) de $T^{\triangleright}$ un plus court chemin reliant u à v dans $\mathcal{I}$. On peut donc construire une autre approximation avec l'algorithme 3.

Algorithme 3 Algorithme d'approximation glouton utilisant FLAC dans $\mathcal{I}^{\triangleright}$ ($\text{Greedy}^{\triangleright}_{FLAC}$)

Entrées : Une instance $\mathcal{I} = (G = (V, A), r, X, \omega)$ pour le problème de Steiner.
Sorties : T, une solution réalisable de $\mathcal{I}$

1: Calculer $\mathcal{I}^{\triangleright}$.
2: $T^{\triangleright} \leftarrow \text{Greedy}_{FLAC}\ (\mathcal{I}^{\triangleright})$
3: $T \leftarrow \bigcup_{(u,v) \in T^{\triangleright}}$ un plus court chemin reliant u à v dans $\mathcal{I}$
4: **Renvoyer** T

En utilisant le lemme 2.2.2, nous serons en mesure de prouver que le rapport d'approximation de $\text{Greedy}^{\triangleright}_{FLAC}$ est $O(\sqrt{k})$. Cependant, du fait de devoir calculer l'instance $\mathcal{I}^{\triangleright}$ pour l'exécuter, son temps de calcul est nettement plus élevé que celui de Greedy_{FLAC}.

Nous décrivons dans un premier temps l'expérience de pensée pour expliquer de manière intuitive comment trouver une solution partielle de bonne densité. Nous décrivons dans la section suivante une implémentation naïve de cette expérience nommée FLAC dont nous démontrerons les propriétés d'approximation partielle. Nous prouvons ensuite les rapports d'approximation des algorithmes Greedy_{FLAC} et $\text{Greedy}^{\triangleright}_{FLAC}$. Nous montrons dans la section suivante comment ces implémentations peuvent être modifiées pour améliorer leur complexité temporelle. Enfin, nous décrivons en quoi FLAC utilise la technique d'ascension du dual.

Ce chapitre se clôt sur une évaluation des performances dans laquelle nous comparons premièrement Greedy_{FLAC} et $\text{Greedy}^{\triangleright}_{FLAC}$ pour déterminer lequel des deux il est préférable d'utiliser en pratique. Nous comparons dans la seconde partie de cette évaluation l'algorithme Greedy_{FLAC} avec d'autres algorithmes utilisés dans la pratique, tel que l'algorithme des plus courts chemins.

4.1 Représentation par une expérience de pensée

Supposons que l'instance ne contienne qu'un seul terminal x. On suppose que x est une source qui émet de l'eau en continu à travers ses arcs entrants. Chaque arc est un tube pouvant contenir un volume égal à son poids en litres. L'eau remonte dans les arcs en sens inverse. Ceux-ci se remplissent tous avec un débit égal à 1 litre par seconde. Quand un arc est rempli, il est dit *saturé*. La figure 4.1 illustre le remplissage des arcs entrant dans x.

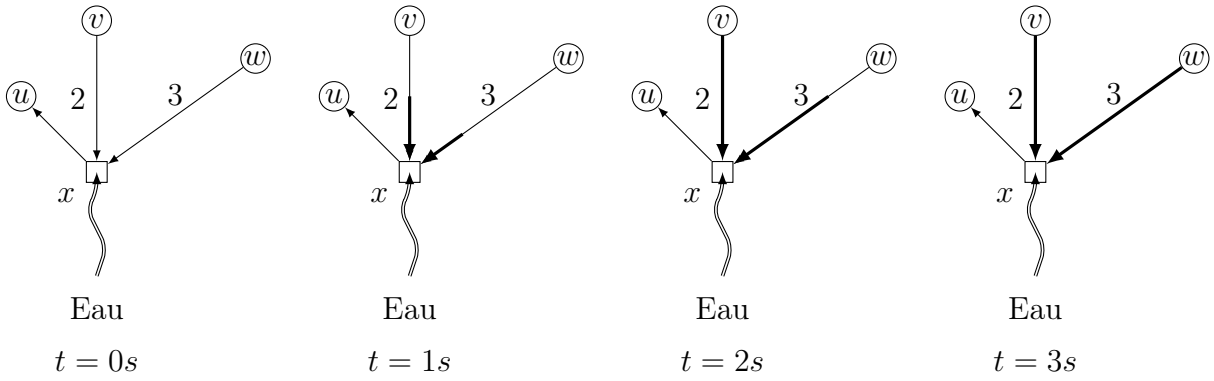


Figure 4.1 – *Exemple d'évolution du liquide autours du terminal x pendant 3 secondes. Le poids de l'arc (x, u) est 1. La présence et la quantité d'eau dans un arc sont indiquées en gras sur l'arc. À l'instant initial, tous les arcs sont vides et les arcs entrant dans x commencent à se remplir avec un litre d'eau par seconde. L'arc (x, u) ne reçoit aucun flot car il est sortant.*

Lorsqu'un arc (u, v) est saturé, l'eau commence alors à s'infiltrer dans les arcs entrant en u, également avec 1 litre par seconde dans chaque arc (comme si l'eau se démultipliait à chaque intersection). La figure 4.2 illustre ce cas.

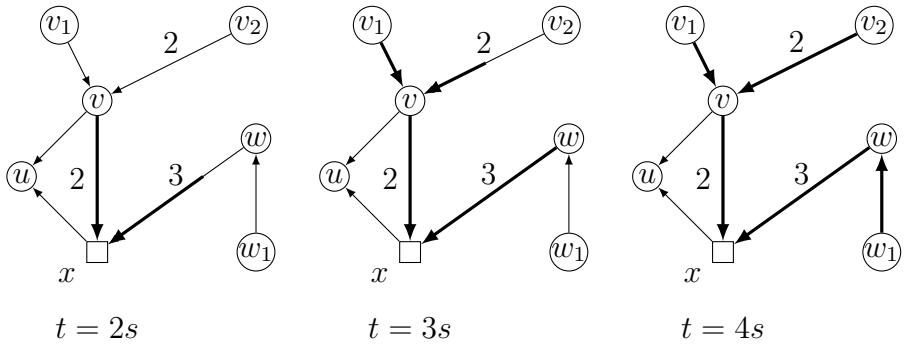


Figure 4.2 – *Exemple d'évolution du liquide autours des nœuds dont un arc sortant est saturé. Les poids des arcs non précisés sur la figure sont tous égaux à 1. La présence et la quantité d'eau dans un arc sont indiqués en gras sur l'arc. Avant les deux premières secondes, aucun arc n'est saturé. Puis l'arc (v, x) est saturé. Les arcs (v_1, v) et (v_2, v) commencent alors à se remplir d'eau avec un litre par seconde. L'arc (v, u) ne se rempli pas car c'est un arc sortant de v. Une seconde plus tard, les arcs (v_1, v) et (w, x) sont saturés et l'arc (w_1, w) commence à se remplir. À la quatrième seconde, (v_2, v) et (w_1, w) sont saturés.*

Enfin, si deux arcs a et b sortant d'un nœud v sont saturés, alors le terminal x envoie du flot à v par deux moyens distincts. Pour éviter que cela ne perturbe le remplissage des arcs entrant en v, on bloque le passage de l'eau dans l'arc a ou b saturé en deuxième (on en choisit un arbitrairement s'ils sont saturés exactement au même moment). On bloque également tout arc sortant de x qui devient saturé.

On observe l'évolution du flot dans le graphe au cours du temps. En particulier, on s'intéresse à l'instant où la racine est atteinte par l'eau émise par x. On remarque dans l'exemple de la figure 4.3 que ce temps est égal au poids d'un plus court chemin de r à x et que le chemin d'arcs saturés non bloqués de r vers x est un plus court chemin.

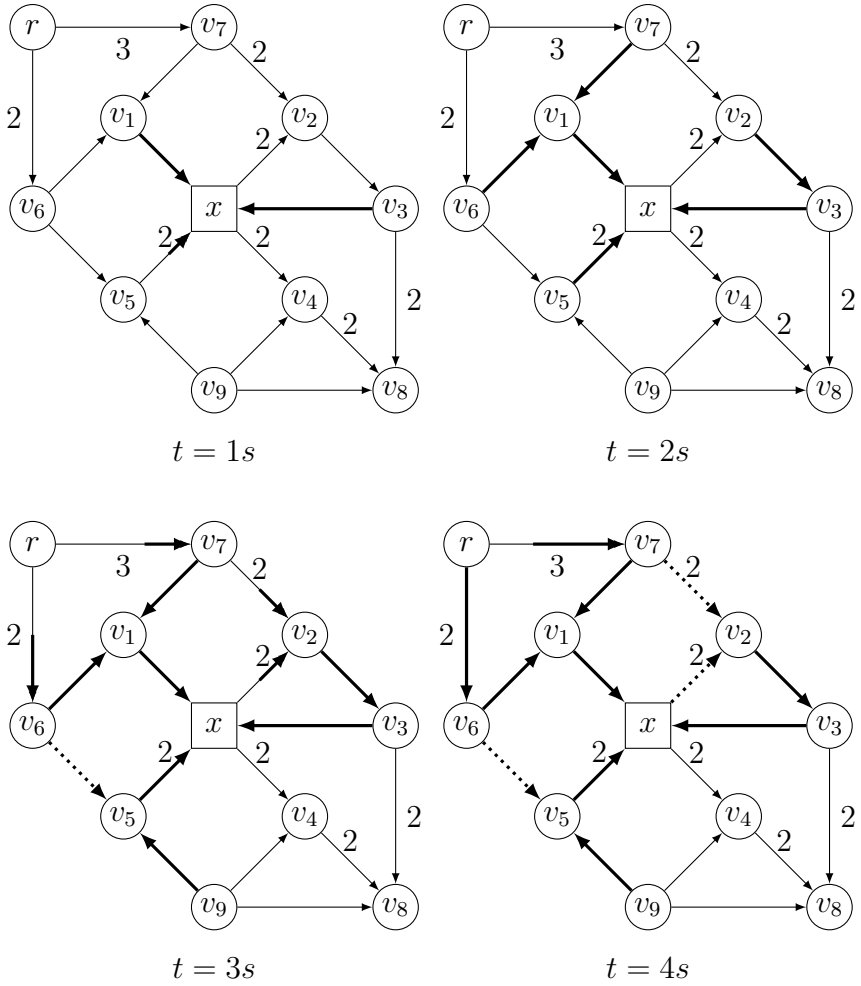


Figure 4.3 – *Exemple d'évolution du liquide. Les arcs dont le poids n'est pas précisé sont de poids 1. La présence et la quantité d'eau dans un arc sont indiqués en gras sur l'arc. Un arc bloqué est noté en pointillé. La racine est atteinte au bout de* 4 *secondes. À la troisième seconde, l'arc* (v_6, v_5) *est bloqué car il devient saturé alors que* (v_6, v_1) *l'est déjà. À la dernière seconde, les arcs* (v_7, v_2) *et* (x, v_2) *sont également bloqués.*

On rajoute maintenant d'autres terminaux à l'instance. Ils envoient tous de l'eau dans le graphe de la même façon. Si les flots en provenance de plusieurs terminaux distincts se rejoignent, alors ils s'additionnent, augmentant ainsi le débit dans l'arc. Ainsi, un arc alimenté par plusieurs terminaux se rempli plus rapidement qu'un arc alimenté par un seul terminal.

Si, à un instant donné, un même nœud est alimenté par le même terminal via deux chemins distincts d'arcs saturés, alors on bloque le dernier arc saturé par ce terminal (si plusieurs arcs ont été saturés au même instant, on en bloque un arbitrairement). De même, si la saturaction d'un arc provoque l'apparition d'un circuit d'arcs saturés, il est bloqué.

On s'intéresse de nouveau à l'instant où la racine est atteinte par l'eau émise par les terminaux. On remarque dans l'exemple de la figure 4.4 que la racine est relié à une partie des terminaux par une arborescence T_0 d'arcs saturés. non bloqués, on remarque également que le temps nécessaire pour atteindre la racine est plus petit que le poids d'un plus court chemin et qu'il est égal à la densité de l'arborescence.

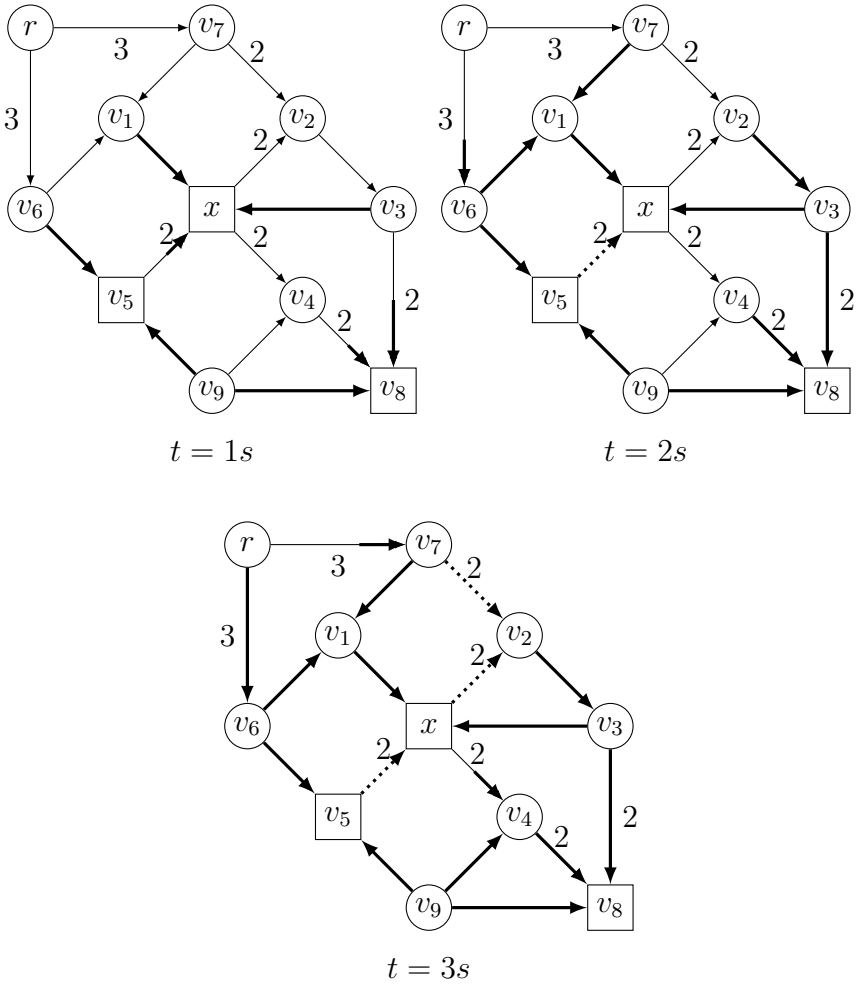


Figure 4.4 – *Exemple d'évolution du liquide dans un graphe avec 3 terminaux : x, v_5 et v_8. Les arcs dont le poids n'est pas précisé sont de poids 1. La présence et la quantité d'eau dans un arc sont indiqués en gras sur l'arc. Un arc bloqué est noté en pointillé. On remarque qu'à la deuxième seconde, la saturation simultanée de (v_6, v_1) et (v_5, x) provoque le blocage de ce dernier. En effet, v_6 est alors alimenté par x de deux manières différentes. Le choix de bloquer (v_5, x) plutôt que (v_6, v_1) est arbitraire. On observe également pendant la troisième seconde une accélération du flot dans les arcs (r, v_6), (v_7, v_2) et (x, v_2) car ils sont chacun alimentés par deux terminaux.*

Un arc est plus vite saturé si son volume est faible ou si plusieurs terminaux lui envoient du flot. Parallèlement un arbre a une bonne densité si son poids est faible ou s'il couvre de nombreux terminaux. Intuitivement, il existe donc, quelque soit l'instance, un lien entre la densité de T_0 et le temps en secondes qu'il a mis pour être saturé.

La section suivante présente une implémentation naïve de l'expérience. Cet algorithme est étudié en premier dans un arbre pour démontrer le lien qui existe entre densité et temps de saturation.

4.2 Algorithme naïf implémentant de l'expérience

Afin d'implémenter l'expérience, nous allons remplacer la notion de temps physique continu, présente dans la section précédente, par une variable t de temps discret qui augmentera à chaque itération de manière non nécessairement linéaire. À chaque itération, un unique arc est saturé, et t sera incrémenté par la durée nécessaire à cette saturation. Si plusieurs arcs saturent exactement au même instant dans l'expérience, alors l'un (choisi arbitrairement) est saturé à la première itération, et les autres sont saturés aux suivantes. La variable t n'est pas modifiée entre ces itérations.

Nous allons définir le graphe G_{SAT} comme étant le graphe induit par les arcs saturés de G (il possède néanmoins tous les nœuds de G). À la première itération, G_{SAT} ne possède aucun arc. Pour un arc $a = (u, v)$, on définit les notations suivantes :

- le flot contenu à l'intérieur de cet arc est $f(a)$, initialisé à 0 pour tous les arcs ;
- le débit $k(v)$ dans le nœud v est le nombre de terminaux auxquels v est relié par un chemin dans G_{SAT} ;
- le débit $k(a)$ dans l'arc a est égal au débit $k(v)$;
- $t(a) = \frac{\omega(a)-f(a)}{k(a)}$ représente la durée maximale restante avant saturation de a. Si $k(a)$ est nul, alors $t(a) = +\infty$.

On peut remarquer que la valeur de $t(a)$ diminue au fur et à mesure que a se remplit et que de nouveaux terminaux viennent alimenter en flot l'arc a.

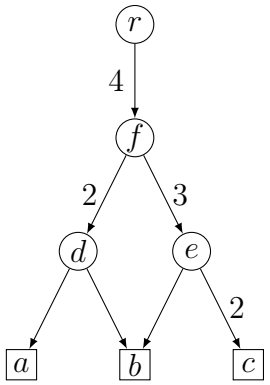
On dira que le flot est *dégénéré* quand G_{SAT} contient deux chemins distincts d'un nœud v vers un terminal t. Afin de toujours renvoyer une arborescence, on corrige le flot dès qu'il est dégénéré. Pour ce faire, on va *marquer* le dernier arc parmi les arcs saturés et le retirer de G_{SAT}. Le marquage d'un arc est équivalent au blocage d'un arc dans l'expérience : aucun flot ne peut passer à travers un arc marqué. On note $\mathcal{M}$ l'ensemble des arcs marqués. Le fait de retirer cet arc de G_{SAT} supprime la dégénérescence du flot. Le fait d'empêcher le flot de passer par cet arc supprime son impact sur l'écoulement futur du flot, comme si cet arc n'avait jamais été présent dans ce graphe.

L'algorithme 4 est une implémentation de l'expérience. Dès qu'un arc sortant de la racine est saturé, l'algorithme s'arrête et renvoie une solution T_0. Cette solution est constituée de l'ensemble des chemins G_{SAT} reliant la racine à des terminaux. Il est possible que, parmi les arcs saturés, certains n'appartiennent pas à ces chemins, ils seront donc ignorés et ne feront pas partie de T_0. Puisque le marquage des arcs empêche la dégénérescence d'un flot, T_0 est une arborescence.

On rappelle également que T_0 peut ne couvrir qu'une partie des terminaux. On utilise l'algorithme 2, donné en page 36, pour construire à l'aide de l'algorithme 4 une solution réalisable de l'instance. La figure 4.5 donne un exemple d'application de cet algorithme.

Algorithme 4 FLow Algorithm Computation (FLAC)

$G_{SAT} = (V, \emptyset)$
$t = 0$
$\mathcal{M} \leftarrow \emptyset$
for $a \in A$ **do** $f(a) = 0$
Tant que Vrai **Faire**
    $(u, v) \leftarrow \arg\min(t(a), a \in A \backslash (\mathcal{M} \cup G_{SAT}))$
    Pour $a \in A \backslash (\mathcal{M} \cup G_{SAT})$ **Faire**
        $f(a) \leftarrow f(a) + k(a) \cdot t(u, v)$
    $t \leftarrow t + t(u, v)$
    $G_{SAT} \leftarrow G_{SAT} \cup (u, v)$
    Si $(u = r)$ **Renvoyer** l'arborescence T_0 dans G_{SAT} reliant r à des terminaux
    Si le flot est dégénéré **Alors**
        $\mathcal{M} \leftarrow \mathcal{M} \cup (u, v)$
        Retirer (u, v) de G_{SAT}



Itération	(u, v)	t	$\mathcal{M}$	$t(d, a)$	$t(d, b)$	$t(e, b)$	$t(e, c)$	$t(f, d)$	$t(f, e)$	$t(r, f)$
0		0	$\emptyset$	1	1	1	2	$+\infty$	$+\infty$	$+\infty$
1	(d, a)	1	$\emptyset$	0	0	0	1	2	$+\infty$	$+\infty$
2	(d, b)	1	$\emptyset$	-	0	0	1	1	$+\infty$	$+\infty$
3	(e, b)	1	$\emptyset$	-	-	0	1	1	3	$+\infty$
4	(e, c)	2	$\emptyset$	-	-	-	0	0	1	$+\infty$
5	(f, d)	2	$\emptyset$	-	-	-	-	0	1	2
6	(f, e)	3	$\{(f, e)\}$	-	-	-	-	-	0	1
7	(r, f)	4	$\{(f, e)\}$	-	-	-	-	-	-	0

Figure 4.5 – *Exemple d'application de l'algorithme FLAC. Le tableau précise l'arc saturé au début de chaque itération, et les valeurs des variables t, $\mathcal{M}$ et $t(a)$ pour chaque arc a à la fin de l'itération. À la fin, le graphe G_{SAT} contient tous les arcs sauf (f, e) qui est marqué. L'arbre T_0 renvoyé est donc $\{(r, f), (f, d), (d, a), (d, b)\}$.*

Nous allons dans un premier temps étudier certaines des propriétés de l'algorithme 4, FLAC, qui nous permettront par la suite de démontrer les résultats d'approximations de l'algorithme 2, Greedy$_{FLAC}$.

4.2.1 Propriétés fondamentales de FLAC

L'expérience de pensée est décrite comme un processus continu dans le temps alors que l'algorithme FLAC décrit les évènements de manière discrète. Nous allons vérifier que les variables de l'algorithme décrivent correctement les comportements physiques du flot et du temps lors du déroulement de l'expérience. Les lemmes qui suivent reflètent une ou plusieurs caractéristiques physiques de l'expérience et les traduisent à l'aide des variables de l'algorithme. Le lemme 4.2.1 s'assure par exemple qu'aucun arc ne contient plus de flot que son volume. Pour faciliter le raisonnement des démonstrations, les balises *Intuition* mettent en évidence ces caractéristiques.

Une fois démontrés, ces lemmes nous permettront de décrire le lien qui existe entre la densité de l'arbre renvoyé par FLAC et la valeur de la variable t à la fin de l'algorithme.

On définit avec $f_j(a)$, $k_j(a)$, $t_j(a)$ et t_j respectivement les valeurs des variables $f(a)$, $k(a)$, $t(a)$ et t au début de la j^{e} itération. Enfin, $(u,v)_j$ est l'arc (u,v) choisi à la ligne 6 de l'itération j. On peut dès à présent noter que t_{j+1} est donc la valeur de t à la fin de cette même itération.

Lemme 4.2.1. *Si au début de l'itération l, $a \notin G_{SAT} \cup \mathcal{M}$, alors $f_l(a) \leq \omega(a)$. Dans le cas contraire $f_l(a) = \omega(a)$.*

Intuition. Ce lemme démontre qu'il n'y a jamais plus de flot dans un arc a que celui ci ne peut en supporter, c'est-à-dire son volume $\omega(a)$. De plus si un arc a été complètement saturé, sa quantité de flot est égale à son volume. Cela se vérifie dans l'algorithme car l'arc choisi au début de chaque itération est celui pour lequel le temps d'écoulement du flot avant saturation est minimum. Aucun autre arc ne peut donc être saturé pendant cette durée, et donc tout arc non saturé ne peut avoir plus de flot en lui que son volume maximal.

Démonstration. Si $l = 0$, le lemme est vérifié, puisque le flot est initialisé à 0 dans tous les arcs. On suppose désormais que $l > 0$.

Si $a \notin G_{SAT} \cup \mathcal{M}$ au début de l'itération l, alors il en était de même à l'itération précédente. Donc $t_{l-1}(a) \geq t_{l-1}((u,v)_{l-1})$ d'après la ligne 6 de l'algorithme FLAC.

D'après la ligne 8 de l'algorithme,

$$\begin{aligned} f_l(a) &= f_{l-1}(a) + k_{l-1}(a) \cdot t_{l-1}((u,v)_{l-1}) \\ &\leq f_{l-1}(a) + k_{l-1}(a) \cdot t_{l-1}(a) \\ &\leq f_{l-1}(a) + k_{l-1}(a) \cdot \frac{\omega(a) - f_{l-1}(a)}{k_{l-1}(a)} \\ &\leq \omega(a) \end{aligned}$$

Si, en revanche, $a \in G_{SAT} \cup \mathcal{M}$, alors soit $i \leq l$ tel qu'au début de l'itération $i-1$, $a \notin G_{SAT} \cup \mathcal{M}$ et à la fin $a \in G_{SAT} \cup \mathcal{M}$. Alors, nécessairement, a est l'arc $(u,v)_{i-1}$. En reprenant les formules précédentes, on déduit que $f_i(a) = \omega(a)$. Puisque le flot d'un arc saturé ou marqué n'est plus mis à jour au cours des itérations suivantes de FLAC, $f_l(a) = \omega(a)$. □

Le lemme et le corollaire suivants traitent du lien qui existe entre débit et volume au sein d'un arc.

Lemme 4.2.2. *Si au début de l'itération $l-1$, $a \notin G_{SAT} \cup \mathcal{M}$, alors pour tout $i \leq l-1$, $f_l(a) = f_i(a) + \sum_{j=i}^{l-1} k_j(a) \cdot (t_{j+1} - t_j)$.*

Intuition. Ce lemme transcrit le lien qui existe entre débit et volume : le débit est la dérivée du volume par rapport au temps. Puisqu'ici le temps est discrétisé, l'intégrale du débit est représentée par une somme discrète.

Démonstration. Soit $i \leq l-1$. Si a n'est ni saturé ni marqué à l'itération $l-1$, alors il n'a été ni saturé ni marqué durant une itération précédente. Donc pour tout $j \in [\![i; l-1]\!]$, $a \notin G_{SAT} \cup \mathcal{M}$.

L'écoulement du temps pendant l'itération j est, d'après la ligne 9 de l'algorithme FLAC, $t_{j+1} - t_j = t_j((u,v)_j)$. La variation du flot contenu dans a pendant l'itération j à la ligne 8 de l'algorithme est donc $f_{j+1}(a) - f_j(a) = k_j(a) \cdot (t_{j+1} - t_j)$.

On peut donc déduire les égalités suivantes :

$$\begin{aligned} f_l(a) - f_i(a) &= \sum_{j=i}^{l-1} f_{j+1}(a) - f_j(a) \\ &= \sum_{j=i}^{l-1} k_j(a) \cdot (t_{j+1} - t_j) \end{aligned}$$

□

Corollaire 4.2.3. *Si au début de l'itération $l-1$, $a \notin G_{SAT} \cup \mathcal{M}$, alors pour tout $i \leq l$, $f_l(a) \geq f_i(a) + (t_l - t_i) \cdot k_i(a)$.*

Intuition. Ce corollaire utilise le fait que le débit dans un arc ne fait qu'augmenter, et que le flot augmente conséquemment au débit. Donc si le débit dans a à un instant donné t est $k(a)$, alors après une durée d, le flot aura, au minimum, augmenté d'une valeur $k(a) \cdot d$.

Démonstration. Il faut remarquer que la valeur de $k_j(a)$, le débit, ne peut qu'augmenter avec j. En effet, entre le début de l'itération j et le début de la suivante, au mieux un arc a été ajouté à G_{SAT} et au pire aucun. Ainsi, le nombre de terminaux reliés à un nœud donné ne peut qu'augmenter. Donc pour tout $j \in [\![i; l-1]\!]$, $k_j(a) \geq k_i(a)$.

$$\begin{aligned} f_l(a) - f_i(a) &= \sum_{j=i}^{l-1} k_j(a) \cdot (t_{j+1} - t_j) \\ &\geq k_i(a) \cdot \sum_{j=i}^{l-1} (t_{j+1} - t_j) \\ &\geq k_i(a) \cdot (t_l - t_i) \end{aligned}$$

□

Enfin les deux lemmes ci-après traitent des temps de saturation prévus $t(a)$.

Lemme 4.2.4. *Si au début de l'itération $l-1$, $a \notin G_{SAT} \cup \mathcal{M}$, alors pour tout $i \leq l$, $t_l + t_l(a) \leq t_i + t_i(a)$.*

Intuition. Ce lemme montre que l'instant de saturation prévu ne peut qu'avancer au cours du temps (se placer *plus tôt* sur la ligne de temps). Connaissant un débit constant dans un arc, on peut déduire le temps qu'il reste avant sa saturation, et donc l'instant de saturation. Si maintenant le débit augmente, cet instant avance. Puisque le débit ne peut qu'augmenter, l'instant ne peut qu' avancer.

Démonstration. D'après le lemme 4.2.1, $f_l(a) \leq \omega(a)$. Démontrons que cette assertion implique le résultat recherché :

$$
\begin{aligned}
& & t_l + t_l(a) &\leq t_i + t_i(a) \\
\Leftarrow & & t_l + \frac{\omega(a) - f_l(a)}{k_l(a)} &\leq t_i + \frac{\omega(a) - f_i(a)}{k_i(a)} \\
\Leftarrow & & (t_l - t_i)k_i(a) + (\omega(a) - f_l(a)) \cdot \frac{k_i(a)}{k_l(a)} &\leq \omega(a) - f_i(a) \\
\Leftarrow & & (t_l - t_i)k_i(a) + f_i(a) &\leq \omega(a) - (\omega(a) - f_l(a)) \cdot \frac{k_i(a)}{k_l(a)}
\end{aligned}
$$

D'après le corollaire 4.2.3, $f_l(a) \geq f_i(a) + (t_l - t_i) \cdot k_i(a)$.

$$
\begin{aligned}
\Leftarrow & & f_l(a) &\leq \omega(a) - (\omega(a) - f_l(a)) \cdot \frac{k_i(a)}{k_l(a)} \\
\Leftarrow & & f_l(a) \cdot (1 - \frac{k_i(a)}{k_l(a)}) &\leq \omega(a) \cdot (1 - \frac{k_i(a)}{k_l(a)})
\end{aligned}
$$

Le débit ne peut qu'augmenter au cours des itérations, donc $k_i(a) \leq k_l(a)$.

$$
\Leftarrow \qquad f_l(a) \leq \omega(a)
$$

□

Lemme 4.2.5. *Si $t_l > t_i + t_i(a)$ pour i vérifiant $i \leq l-1$, alors l'algorithme 4 a saturé ou marqué l'arc a pendant une itération $j - 1 < (l-1)$, et $t_j \leq t_i + t_i(a)$.*

Intuition. Ce second lemme explique que si, à un instant donné, il était prévu que l'arc a mette au maximum $t(a)$ secondes avant d'être saturé, alors a a nécessairement été saturé ou marqué avant que ne s'écoulent plus de $t(a)$ secondes. Ainsi, s'il s'est écoulé strictement plus de $t(a)$ secondes, on est certain que a a été saturé ou marqué.

Démonstration. Si nous supposons par l'absurde que a n'a été ni saturé ni marqué pendant ou avant cette itération, alors d'après le lemme 4.2.1 et le corollaire 4.2.3,

$$
\begin{aligned}
\omega(a) &\geq f_i(a) + (t_l - t_i)k_i(a) \\
&> f_i(a) + t_i(a)k_i(a) \\
&> f_i(a) + \frac{\omega(a) - f_i(a)}{k_i(a)} k_i(a) \\
&> \omega(a)
\end{aligned}
$$

Ceci est une contradiction. Donc a a été saturé ou marqué pendant ou avant l'itération $l-1$. Soit $j-1$ cette itération. D'après la ligne 9 et le lemme 4.2.4, $t_j = t_j + t_j(a) \leq t_i + t_i(a)$.

Si $j = l$, alors $t_l = t_{l-1} + t_{l-1}(a) \leq t_i + t_i(a)$, ce qui est exclu par hypothèse. Donc a a été saturé ou marqué strictement avant l'itération $l-1$. □

Nous allons maintenant étudier l'arborescence renvoyée T_0 par l'algorithme FLAC dans le cas où le graphe est également une arborescence T. À l'aide des lemmes démontrés précédemment, nous allons expliciter le lien qui existe entre la densité de T_0 et la valeur de la variable t à la fin de l'algorithme. Puis nous verrons que T_0 est un sous-arbre de T de densité minimum.

4.2.2 Étude de l'algorithme FLAC appliqué à une arborescence

On recherche le lien qui existe entre la densité d'une arborescence et le temps qu'il faut à l'expérience décrite précédemment pour la saturer complètement.

On verra que la propriété qui découle de cette étude montre qu'il est aisé de trouver dans cette arborescence le sous-arbre de densité minimum à l'aide de l'algorithme FLAC.

Soit T une arborescence enracinée en r. Soit X l'ensemble des feuilles de T, toutes terminales. L'arborescence T est trivialement la solution optimale de cette instance du problème de Steiner mais ce n'est pas le résultat qui nous intéresse. Supposons dans un premier temps qu'on attache à la racine de T un arc $b = (r_0, r)$ de poids infini, et que r_0 remplace r en tant que racine de l'instance de Steiner. On peut voir b comme un puits sans fond qui récupère tout l'excès d'eau transmis à travers r. Donc si on applique l'algorithme FLAC sur cette instance, cet arc sera nécessairement saturé en dernier et l'algorithme ne s'arrêtera donc jamais à la ligne 11 avant d'avoir saturé tous les arcs de T.

On vérifie donc aisément que pour tout arc il existe une itération pendant laquelle cet arc est saturé, et il existe une itération φ au début de laquelle tous les arcs de T sont saturés.

Résultat général. Soit ω le poids de T, k le nombre de feuilles de T et $d = \frac{\omega}{k}$ sa densité. Soit φ l'itération au début de laquelle tous les arcs de T sont saturés. On désigne par ε le flot $f_\varphi(b)$ dans l'arc b au début de cette itération.

Théorème 4.2.6.

$$t_\varphi = d + \frac{\varepsilon}{k}$$

Intuition. Si on interprète cette formule plus intuitivement, en multipliant les deux membres de l'égalité par k, on obtient $k \cdot t_\varphi = \omega + \varepsilon$. À gauche de l'égalité, on a l'ensemble du flot débité par les terminaux durant l'expérience qui a duré t_φ secondes. En effet, par définition, le débit sortant d'un terminal est l'ensemble des nœuds auquel il est relié par un chemin d'arcs saturés. Puisque nous nous sommes placés dans une arborescence, ce débit est toujours égal à 1 pour tous les terminaux. À droite de l'égalité, il s'agit du volume total de flot contenu dans l'arborescence et l'arc b à la fin de l'expérience. Cette égalité se contente donc simplement de certifier qu'il y a conservation du flot dans une arborescence tout au long de l'expérience.

La démonstration de ce théorème, du fait de sa longueur est donnée dans l'annexe A, en page 163.

Le théorème 4.2.6 met en avant le fait qu'un arbre pour lequel $\varepsilon = 0$ a une densité égale au temps de saturation. Si $\varepsilon = 0$, alors tous les arcs reliant la racine à chacun de ses fils ont été saturés en dernier et en même temps.

Étude du premier sous-arbre saturé. On s'intéresse maintenant à la réponse de l'algorithme FLAC si on définit de nouveau r comme étant la racine de l'instance. Soit T_0 le sous-arbre saturé de T renvoyé par FLAC.

Soient ω_0, k_0 et d_0 les poids, nombre de feuilles, et densité de T_0. Soit $\varphi_0 - 1$ l'itération pendant laquelle T_0 est renvoyé. On note respectivement t_{φ_0} et ε_0 la valeur qu'aurait la variable t et le volume d'eau qu'aurait b au début de l'itération suivante.

Lemme 4.2.7. *$d_0 = t_{\varphi_0}$ et $\varepsilon_0 = 0$.*

Démonstration. Les terminaux dans $T \backslash T_0$ n'affectent pas la saturation des arcs de T_0. En effet, si c'était le cas, alors au moins une des feuilles de T non feuille de T_0 enverrait du flot dans un des arcs de T_0 pour accélérer sa saturation. Alors ce terminal serait relié à la racine par des arcs saturés avant l'instant t_{φ_0} et cette feuille appartiendrait à T_0. Donc le flot s'infiltre de la même manière dans T_0 si on supprimait tous les arcs de $T \backslash T_0$. Ainsi, par le théorème 4.2.6, $d_0 = t_{\varphi_0} - \frac{\varepsilon_0}{k_0}$.

D'après la définition de T_0, aucun flot n'a été envoyé dans l'arc b pendant ou avant l'itération $\varphi_0 - 1$. En effet, dans le cas contraire, un arc sortant de r aurait été saturé strictement avant l'itération $\varphi_0 - 1$ et un autre sous-arbre aurait déjà été renvoyé à la ligne 11 lors de cette même itération. Cela rentre en contradiction avec la définition de T_0. Donc $\varepsilon_0 = 0$ et $d_0 = t_{\varphi_0}$. □

Lemme 4.2.8. *$d_0 \leq d$*

Démonstration. D'après le théorème 4.2.6 et le lemme 4.2.7, on souhaite démontrer $t_{\varphi_0} = d_0 \leq d = t_\varphi - \frac{\varepsilon}{k}$, soit $\varepsilon \leq k \cdot (t_\varphi - t_{\varphi_0})$.

D'après le lemme 4.2.2,

$$\begin{aligned}
\varepsilon &= \varepsilon_0 + \sum_{j=\varphi_0}^{\varphi-1} k_j(r) \cdot (t_{j+1} - t_j) \\
&\leq k \cdot \sum_{j=\varphi_0}^{\varphi-1} (t_{j+1} - t_j) \\
&\leq k \cdot (t_\varphi - t_{\varphi_0})
\end{aligned}$$

□

Théorème 4.2.9. *T_0 est un sous-arbre de T de densité minimum.*

Intuition. Si un arbre T_1 avait une meilleure densité que T_0, on pourrait utiliser le lien qui existe entre temps de saturation et densité pour démontrer une absurdité : nécessairement, l'arbre T_1 devrait atteindre (partiellement ou complètement) la racine avant T_0. En effet, si on prend T_1 à part, et qu'on lance l'expérience dans cet arbre, on trouve d'après le lemme 4.2.8 un arbre T_2 de densité meilleure que T_1 donc que T_0. De plus, le temps de saturation de T_2

est égal à sa densité. Or puisqu'il y a plus de sources d'eau dans T que dans T_1 seul, T_2 est saturé encore plus vite : donc nécessairement avant t_{φ_0}, ce qui est exclu puisque T_0 est le premier arbre à atteindre la racine.

La preuve de ce théorème est donnée en annexe A en page 168, du fait de sa longueur.

Corollaire 4.2.10. *Si $\varepsilon = 0$, alors le propre sous-arbre de densité minimum de T est T lui-même.*

Remarque 10. Toutes les propriétés précédentes restent valident même si certains terminaux ne sont pas des feuilles. En effet, dans ce cas, il suffit de copier ce terminal et de relier l'original à la copie par un arc de poids nul. La copie remplace l'original dans l'ensemble des terminaux.

Nous allons maintenant étudier l'arborescence renvoyée T_0 par l'algorithme FLAC dans le cas où le graphe est quelconque. À l'aide des lemmes démontrés dans cette sous-section et la sous-section précédente, nous allons démontrer les propriétés de la densité de T_0. Puis nous démontrerons le rapport d'approximation de Greedy$_{FLAC}$.

4.2.3 Étude des l'algorithmes FLAC et Greedy$_{FLAC}$ dans un graphe

Contrairement au cas de l'arbre, le cas du graphe est soumis au marquage des arcs durant les diverses itérations. Un arc marqué ne transmet plus de flot.

On considère à présent comme donnée une instance arbitraire $\mathcal{I} = (G = (V, A), r, X, \omega)$ du problème de Steiner.

L'intérêt de marquer des arcs est la garantie de commencer toute itération avec un flot non dégénéré et de toujours renvoyer un arbre. Marquer le dernier arc saturé permet de prendre une décision rapidement. Un flot dégénéré signifie que deux chemins relient un nœud v du graphe G_{SAT} à un terminal x. Un de ces deux chemins a été complètement saturé en dernier au cours de l'exécution. On peut supposer qu'il vaut mieux continuer à relier v à x avec le premier chemin plutôt qu'avec v. Cette décision est rapide. En effet, marquer un arc revient à remettre en cause le calcul d'écoulement du flot qui a été fait après sa saturation et il faut recommencer l'algorithme à partir de l'itération où cet arc s'est vu saturé. Or l'arc est ici le dernier à avoir été saturé : il n'y a donc aucune itération précédente remise en cause.

Cette méthode a néanmoins un défaut, décrit ci-après. Nous verrons plus loin que, malgré ce défaut, il est possible de donner une garantie de performance pour l'algorithme Greedy$_{FLAC}$.

Prenons l'exemple suivant, donné en figure 4.6 : le terminal x est relié à un nœud v via 2 chemins, le chemin singleton (v, x) et un chemin de longueur 2, (v, w, x). Rajoutons $k - 1$ terminaux successeurs de w. Enfin, relions la racine à v avec un chemin de longueur $H \geq 1$. Tous les arcs sont de poids unitaire. Clairement, sauf si $k = 1$, un arbre de meilleure densité est le graphe privé de l'arc (v, x). Cependant, FLAC va saturer (v, x) ainsi que tous les arcs sortant de w, puis marquer (v, w) car le flot sera dégénéré : il existe deux chemins reliant v à x. Donc il renverra le chemin (r, v, x).

Ce choix n'est pas optimal du fait de deux phénomènes. Le premier est que les terminaux ne s'unissent pour accélérer le flot qu'au bout d'une seconde. Ce temps a déjà permis à x de rejoindre v tout seul par un autre chemin. Le second est que FLAC prend la décision de marquer l'arc (v, w) et de bloquer l'accès à la racine de $k - 1$ terminaux uniquement à cause

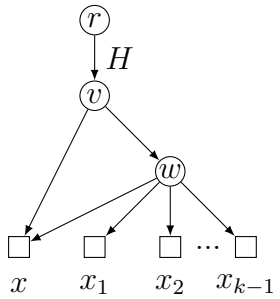


FIGURE 4.6 – *Exemple où le choix de l'algorithme FLAC n'est pas optimal.*

de x, sans chercher à trouver un compromis où à la fois x et les autres terminaux puissent continuer à déverser du flot. Pour corriger ces défauts, à chaque conflit, il faudrait tester chaque possibilité. Néanmoins, cette méthode est exponentielle en le nombre de conflits, et ce nombre peut se rapprocher de la taille de l'instance.

Étude de l'arbre renvoyé par FLAC. Nous allons étudier l'arbre T_0 renvoyé par FLAC. Nous l'avons vu précédemment, dans une arborescence, T_0 est le sous-arbre de densité minimum, mais ce n'est plus vrai dans un graphe. Nous allons toutefois voir que T_0 conserve des propriétés similaires. Nous conservons les notations de la partie précédente, y compris la variable ε_0 correspondant au flot qui traverse la racine avant que T_0 ne soit renvoyé.

Lemme 4.2.11. *$d_0 = t_0$ et $\varepsilon_0 = 0$.*

Démonstration. La démonstration rappelle celle du lemme 4.2.7.

Les arcs dans $G \backslash T_0$ n'affecte pas la saturation des arcs de T_0. En effet, sinon, il existerait une itération où un tel arc serait saturé. Si pendant cette itération l'arc était marqué, il ne transmettrait plus de flot et n'accélèrerait donc pas la saturation des arcs de T_0. Et si l'arc n'était pas marqué, alors il appartiendrait à T_0, ce qui est exclu.

Donc le flot s'infiltre à l'intérieur des arcs de T_0 de la même manière dans G que si on appliquait FLAC à l'arbre T_0 seul. Ainsi, par le théorème 4.2.6, $d_0 = t_0 - \frac{\varepsilon_0}{k_0}$.

D'après la définition de T_0, aucun flot n'a été envoyé à travers la racine pendant ou avant l'itération $\varphi_0 - 1$. En effet, dans le cas contraire, un arc sortant de r aurait été saturé strictement avant l'itération $\varphi_0 - 1$ et une autre arborescence aurait déjà été renvoyée à la ligne 11 lors de cette même itération. Cela rentre en contradiction avec la définition de T_0. Donc $\varepsilon_0 = 0$ et $d_0 = t_0$. □

Théorème 4.2.12. *Soit T_1 un arbre de G enraciné en r de densité d_1. Si aucun arc de T_1 n'est marqué, alors $d_0 \leq d_1$.*

Démonstration. La démonstration rappelle celle du théorème 4.2.9.

Supposons que $d_1 < d_0$. Si nous appliquons l'algorithme FLAC à T_1 seul, il en résulte un arbre T_2 de densité $d_2 \leq d_1 < d_0$ vérifiant $t_2 = d_2$.

Ajouter dans G des arcs et des terminaux à T_2 ne peut qu'accélérer sa saturation. Puisqu'aucun arc de T_1 n'a été marqué, il en est de même pour T_2, donc le flot dans T_2 n'a pas été bloqué. Puisque T_0 atteint la racine lors d'une itération vérifiant $t = t_0 = d_0 > d_2 = t_2$, T_2 devrait être déjà complètement saturé lors d'une itération précédente. Ce qui est exclu par définition de T_0. □

Rapport d'approximation de l'algorithme Greedy$_{FLAC}$. Nous allons maintenant nous intéresser au rapport d'approximation de Greedy$_{FLAC}$. Nous allons voir dans un premier temps qu'il n'existe pas de plus court chemin reliant la racine à un terminal dont la densité (égale à son coût) est plus petite que celle de l'arbre renvoyée par FLAC.

Cet algorithme est naturellement conçu pour trouver une meilleure solution partielle que ce chemin. En effet, si X ne contient qu'un seul terminal x, alors le premier flot à rejoindre la racine dessine dans le graphe un plus court chemin P reliant r à x, comme le ferait l'algorithme de Dijkstra. On conserve de plus la propriété principale de FLAC qui est que le temps en secondes qu'il a fallu à ce flot pour rejoindre r est exactement le poids de P. Si on ajoute d'autres terminaux dans X, on observe nécessairement une accélération de ce flot et la racine est rejointe en moins de temps qu'il ne faut pour remplir P entièrement.

Cette intuition ne tient pas compte du marquage des arcs au cours de l'algorithme. Potentiellement, en faisant agir tous les terminaux, le remplissage du chemin P a été bloqué car un de ses arcs est marqué. Nous verrons que ça n'a aucun impact.

Soit x un terminal tel qu'un chemin de poids minimum reliant la racine à x ait un poids plus petit que tout chemin reliant la racine à un terminal, et soit P ce chemin.

Lemme 4.2.13. *Le temps t_{φ_0} n'excède pas la densité (ou le poids) de P.*

Intuition. P ne couvre que le terminal x, qui le remplit avec un débit de 1 Ls^{-1}. Si un arc a de P est marqué, alors un autre groupe de terminaux (comprenant x ou non), est arrivé avant x au niveau de a et a déjà commencé à saturer le sous-chemin de P reliant r à a. Ce groupe est donc arrivé plus vite que x, plus haut dans le chemin P, et sature plus rapidement les arcs restants de P (car ils sont plus nombreux que x). Donc la racine est atteinte plus rapidement que si seul x saturait P.

Démonstration. Soient $v_l = r$, v_{l-1}, …, v_1, $v_0 = x$ la liste des nœuds successifs de P, et soit ω_i le poids du sous-chemin de P reliant v_i à x. Nous allons montrer par récurrence sur i que soit $t_{\varphi_0} \leq \omega_i$, soit il existe un itération j_i vérifiant $k_{j_i}(v_i) \geq 1$ et $t_{j_i} \leq \omega_i$.

Si $i = 0$, alors, en posant $j_0 = 1$, la propriété est vraie.

Si la propriété est vraie pour $i-1$, et si $t_{\varphi_0} > \omega_i \geq \omega_{i-1}$, alors, par hypothèse de récurrence, il existe une itération $j_{(i-1)}$ telle que $t_{j_{(i-1)}} \leq \omega_{i-1}$ et $k_{j_{(i-1)}}(v_i, v_{i-1}) \geq 1$. Cette dernière inégalité nous montre que $t_{j_{(i-1)}}(v_i, v_{i-1}) = (\omega_i - \omega_{i-1} - f_{j_{(i-1)}}(a))/k_{j_{i-1}}(v_{i-1}) \leq \omega_i - \omega_{i-1}$.

Or, on rappelle que $t_{\varphi_0} > \omega_i$. Donc $t_{\varphi_0} > t_{j_{(i-1)}} + t_{j_{(i-1)}}(v_i, v_{i-1})$. On en déduit, d'après le lemme 4.2.5, que (v_i, v_{i-1}) a été saturé ou marqué lors d'une itération $j_i - 1 < \varphi_0 - 1$ et $t_{j_i} \leq t_{j_{(i-1)}} + t_{j_{(i-1)}}(v_i, v_{i-1}) \leq \omega_i$. Si au début de l'itération j_i, l'arc (v_i, v_{i-1}) est saturé ou marqué, v_i est relié à au moins un terminal dans G_{SAT}. Donc $k_{j_i}(v_i) \geq 1$.

Ceci conclue la récurrence, on a donc $t_{\varphi_0} \leq \omega_l$ ou il existe une itération j_l vérifiant $k_{j_l}(v_l) \geq 1$ et $t_{j_l} \leq \omega_l$. Or $v_l = r$, donc si r est relié à un terminal dans G_{SAT}, l'algorithme s'est arrêté et a renvoyé une solution : $j_l = \varphi_0$. On vérifie donc également dans ce cas que $t_{\varphi_0} \leq \omega_l$. □

Corollaire 4.2.14. $d_0 \leq \omega(P)$

Démonstration. D'après le lemme 4.2.11. □

Soit maintenant T^* une solution optimale pour l'instance $\mathcal{I}$ du problème de Steiner, et soit ω^* son poids. Nous avons montré que FLAC renvoie un arbre de meilleur densité que le plus court chemin P, alors nous pouvons démontrer que le rapport d'approximation de Greedy$_{FLAC}$ est meilleur que celui de l'algorithme des plus courts chemins.

Théorème 4.2.15. *Greedy$_{FLAC}$ est une k-approximation polynomiale pour le problème de l'arborescence de Steiner.*

Démonstration. D'après le corollaire 4.2.14, $d_0 \leq \omega(P)$. Or toute solution optimale contient un chemin de r à x. Donc $d_0 \leq \omega^* = k\frac{\omega^*}{k}$. D'après le lemme 2.2.2, Greedy$_{FLAC}$ est une $\int_0^k du = k$-approximation. □

La section 4.3 sera dédiée à l'amélioration de ce rapport d'approximation. Nous déterminons ci-après la complexité temporelle et spatiale de l'algorithme Greedy$_{FLAC}$.

4.2.4 Complexités temporelle et spatiale

Lemme 4.2.16. *Les complexités temporelle de FLAC est $O(m \cdot (kn + m)$. Sa complexité spatiale est $O(n + m)$.*

Démonstration. À chaque itération, un arc est saturé. L'algorithme s'arrête dès qu'un arc sortant de r est saturé. On effectue donc au plus m itérations. Durant ces itérations, on recherche, à la ligne 6 de l'algorithme 4, l'arc saturé durant cette itération ce qui se fait en temps $O(m)$. Puis, on met à jour le flot de chaque arc à la ligne 8, ce qui nécessite m opérations également. On vérifie enfin que le flot n'est pas dégénéré, c'est-à-dire que G_{SAT} ne contient pas deux chemins distincts d'un nœud vers un terminal. Pour cela, on vérifie que l'ensemble des ancêtres de chaque terminal est un arbre. Cela peut se faire en temps $O(kn)$. À la dernière itération, on construit et renvoie T_0 ce qui se fait en temps $O(n)$ en parcourant en largeur G_{SAT} depuis r. La complexité temporelle est donc $O(m \cdot (kn + m))$.

L'espace utilisé par cet algorithme concerne le graphe G_{SAT} de taille $O(n+m)$, l'ensemble des arcs marqués de taille $O(m)$ et l'ensemble des variables $t(a), f(a)$ et $k(a)$ pour chaque arc a, de taille $O(m)$. la complexité spatiale est donc $O(n + m)$. □

Théorème 4.2.17. *Les complexités temporelle et spatiale de Greedy$_{FLAC}$ sont respectivement $O(k \cdot m \cdot (kn + m)$ et $O(n + m)$.*

Démonstration. Greedy$_{FLAC}$ applique l'algorithme FLAC au plus k fois, car à chaque itération, au moins un nouveau terminal est couvert. L'espace nécessaire à l'exécution de Greedy$_{FLAC}$ est le même que celui de FLAC, plus un espace réservé à la construction de la solution réalisable renvoyée, de taille $O(n)$. D'après le lemme 4.2.16. □

4.3 Amélioration du rapport d'approximation

Il est possible de réduire le rapport d'approximation de l'algorithme Greedy$_{FLAC}$ à $O(\sqrt{k})$ sans modifier fondamentalement ni Greedy$_{FLAC}$ ni FLAC. Au lieu de travailler dans l'instance $\mathcal{I}$ nous allons travailler dans l'instance équivalente des plus courts chemins $\mathcal{I}^{\triangleright} = (G^{\triangleright}, r, X, \omega^{\triangleright})$, définie dans le chapitre 2 en page 16.

Comme le décrit l'algorithme 3, page 36, Greedy$^{\triangleright}_{FLAC}$ est l'algorithme qui calcule l'instance des plus courts chemins, puis applique Greedy$_{FLAC}$ à cette instance et en déduit une solution réalisable de $\mathcal{I}$ de poids plus petit ou égal. Nous appliquons donc désormais FLAC à l'instance $\mathcal{I}^{\triangleright}$.

Soit T_2 une solution partielle de $\mathcal{I}^{\triangleright}$ telle que

- T_2 est de hauteur 2 ;
- T_2 couvre $l \leq k$ terminaux ;
- la densité de T_2 est minimum parmi toutes les solutions partielles de hauteur 2.

On suppose sans perte de généralité que, dans T_2, la racine n'a qu'un fils v. En effet, dans le cas contraire, alors T_2 est une union disjointe d'arborescences enracinées en r et toutes ces arborescences ont la même densité : $d(T_2)$. Soit $(x_1, x_2, \ldots, x_l)$ les terminaux couverts par T_2, $\alpha = \omega^{\triangleright}(r, v)$ et $\beta_i = \omega^{\triangleright}(v, x_i)$.

Lemme 4.3.1. *Le temps t_{φ_0} n'excède pas la densité de T_2.*

Intuition. Cela est dû au fait que T_2 est de hauteur 2. Si aucun arc de T_2 n'est marqué, alors le premier arbre atteignant r a une meilleure densité que T_2. Si un arc de T_2 est marqué, ce ne peut être qu'un arc sortant de v. Sinon la racine serait atteinte et l'algorithme s'arrêterait. Cet arc relie v à un terminal x. Si cet arc est marqué, c'est qu'un flot issu d'un groupe de terminaux a atteint le nœud v avant x. Ce groupe commence donc à saturer (r, v) plus rapidement que x et le sature avec plus de débit (car ils sont au moins aussi nombreux que x seul). Donc l'arbre qui arrive en premier à la racine est plus rapide que T_2.

Démonstration. Montrons que pour tout $i \leq l$, $\beta_i \leq d(T_2)$. Si ce n'est pas le cas, il existe $j \leq l$ tel que $\beta_j > d(T_2)$. Montrons que cette hypothèse est absurde. Soit T_2^j le sous-arbre de T_2 privé de l'arc (v, x_j) et du terminal x_j. Par définition de T_2, $d(T_2^j) \geq d(T_2)$.

$$
\begin{aligned}
d(T_2) &< \beta_j \\
\frac{\alpha + \sum_{i \neq j} \beta_i + \beta_j}{l} &< \beta_j \\
\alpha + \sum_{i \neq j} \beta_i &< (l-1) \cdot \beta_j \\
d(T_2^j) &< \beta_j
\end{aligned}
\qquad\qquad
\begin{aligned}
d(T_2^j) &\geq d(T_2) \\
\frac{\alpha + \sum_{i \neq j} \beta_i}{l-1} &\geq \frac{\alpha + \sum_{i \neq j} \beta_i + \beta_j}{l} \\
\alpha + \sum_{i \neq j} \beta_i &\geq (l-1) \cdot \beta_j \\
d(T_2^j) &\geq \beta_j
\end{aligned}
$$

Ces deux inégalités s'excluent mutuellement. Donc pour tout $i \leq l$, $\beta_i \leq d(T_2)$.

À la première itération, chaque arc (v, x_i) prévoit d'être saturé après β_i secondes d'écoulement du flot : $t_1 + t_1(v, x_i) = \beta_i$. S'il existe i tel que $t_{\varphi_0} \leq \beta_i \leq d(T_2)$, alors le lemme est prouvé. Sinon, d'après le lemme 4.2.5, il existe pour chaque arc (v, x_i) une itération $j_i - 1 < \varphi_0 - 1$ où cet arc est saturé et $t_{j_i} \leq t_1 + t_1(v, x_i) = \beta_i$. On suppose sans perte de généralité que $j_1 \leq j_2 \leq \cdots \leq j_l$. Au début de l'itération j_i, le débit $k_{j_i}(r, v)$ vaut au moins i.

D'après le lemme 4.2.2,

$$\begin{aligned} f_{j_l}(r,v) &= f_1(a) + \sum_{j=i}^{j_l} k_j(r,v)\cdot(t_{j+1}-t_j) \\ &= \sum_{j=1}^{j_l} k_j(r,v)\cdot(t_{j+1}-t_j) \\ &= \sum_{j=1}^{j_1} k_j(r,v)\cdot(t_{j+1}-t_j) + \sum_{i=1}^{l-1}\sum_{j=j_i}^{j_{(i+1)}} k_j(r,v)\cdot(t_{j+1}-t_j) \\ &\geq \sum_{i=1}^{l-1}\sum_{j=j_i}^{j_{(i+1)}} k_j(r,v)\cdot(t_{j+1}-t_j) \end{aligned}$$

Au début de l'itération j_i, le débit $k_{j_i}(r,v)$ vaut au moins i.

$$\begin{aligned} &\geq \sum_{i=1}^{l-1} i\cdot(t_{j_{(i+1)}}-t_{j_i}) \\ &\geq \sum_{i=1}^{l-1} i\cdot(t_{j_{(i+1)}}-\beta_i) \\ &\geq \sum_{i=1}^{l-1}(t_{j_l}-\beta_i) \end{aligned}$$

Donc au début de l'itération j_l, le flot dans (r,v) vaut au moins $\sum_{i=1}^{l-1}(t_{j_l}-\beta_i)$. Donc $t_{j_l}+t_{j_l}(r,v) \leq t_{j_l} + \frac{\alpha-\sum_{i=1}^{l}(t_{j_l}-\beta_i)}{l} = d(T_2)$.

D'après le lemme 4.2.5, si $t_{\varphi_0} > d(T_2) \geq t_{j_l}+t_{j_l}(r,v)$, alors (r,v) a été saturé ou marqué strictement avant l'itération $\varphi_0 - 1$. Ce qui est exclu par la définition de φ_0. □

Corollaire 4.3.2. $d_0 \leq d(T_2)$

Démonstration. D'après le lemme 4.2.11. □

Soit maintenant T^* une solution optimale pour l'instance $\mathcal{I}^{\triangleright}$ du problème de Steiner, et soit ω^* son poids. Nous avons montré que FLAC renvoie un arbre de meilleure densité que T_2, alors nous pouvons démontrer que le rapport d'approximation de $\text{Greedy}^{\triangleright}_{FLAC}$ est meilleur que $4\sqrt{2k}$.

Théorème 4.3.3. *$\text{Greedy}^{\triangleright}_{FLAC}$ est une $4\sqrt{2k}$-approximation polynomiale pour le problème de l'arborescence de Steiner.*

Démonstration. Nous reproduisons ici une preuve plus générale donnée dans [CCCD98]. Une arborescence T_2 de hauteur 2 de densité minimum a nécessairement une densité plus petite que toute arborescence T_2^* de hauteur 2 couvrant X de poids minimum. T_2^* est une $(4\sqrt{k/2})$-approximation de T^* [Zel97, HRZ01].

D'après le corollaire 4.3.2, $d(T_0) \leq d(T_2) \leq 4\sqrt{k/2}\frac{\omega(T^*)}{k}$. D'après le lemme 2.2.2, en appliquant Greedy$_{FLAC}$ à $G^{\triangleright}$, Greedy$^{\triangleright}_{FLAC}$ trouve une solution réalisable $T^{\triangleright}$ de poids au plus $\int_0^k \frac{4\sqrt{u/2}}{u} du \cdot \omega(T^*) = 4\sqrt{2k} \cdot \omega(T^*)$. □

Greedy$^{\triangleright}_{FLAC}$ permet donc d'améliorer le rapport d'approximation de Greedy$_{FLAC}$, et est un point de départ à la recherche de meilleures approximations conçues avec FLAC.

4.3.1 Complexités temporelle et spatiale

Théorème 4.3.4. *Les complexités temporelle et spatiale de Greedy$^{\triangleright}_{FLAC}$ sont respectivement $O(k \cdot n^4)$ et $O(n^2)$.*

Démonstration. Greedy$^{\triangleright}_{FLAC}$ applique l'algorithme Greedy$_{FLAC}$ dans un graphe complet orienté : ce graphe a n^2 arcs. D'après le théorème 4.2.17, cette exécution se fait en temps $O(k \cdot n^4)$ et en espace $O(n^2)$. La construction de l'instance des plus courts chemins nécessite le calcul du poids des plus courts chemins entre chaque couple de nœuds. L'algorithme de Floyd-Warshall permet de calculer ces chemins en temps $O(n^3)$ et en espace $O(n^2)$ [Flo62]. □

4.4 Optimisation de l'algorithme FLAC

Dans cette section, nous décrivons une seconde implémentation de l'expérience de base. La complexité en temps de cette implémentation est plus basse que celle de l'algorithme 4. Trois points dans cet algorithme peuvent être améliorés : comment trouver à chaque itération le prochain arc (u, v) qu'il faut saturer ou marquer ? Comment détecter un conflit et le corriger ? Et comment construire l'arbre T_0 que l'on renvoie à la fin ?

4.4.1 Description de l'algorithme

Définition et initialisation des variables et des structures de données utilisées. La première amélioration provient du fait suivant : à tout instant, le débit $k(a)$ qui remplit un arc $a = (u, v)$ et le débit $k(v)$ qui traverse v sont les mêmes, à savoir le nombre de terminaux auxquels v est relié par un chemin d'arcs saturés. Donc le flot contenu dans tous les arcs non saturés entrant en v est le même, et les arcs entrant en v sont saturés toujours par ordre croissant de poids.

Pour trouver le prochain arc qui sera saturé, il n'est donc plus nécessaire de comparer tous les arcs, mais uniquement l'arc non saturé entrant en v qui est non marqué et de poids minimum pour tout nœud v du graphe. De plus le flot ne devrait plus être enregistré pour chaque arc mais pour chaque nœud. Cependant, nous allons voir que cette variable n'est plus nécessaire. À chaque nœud est associée la liste $\Gamma^-(v)$ des arcs entrant en v non saturés, triés par ordre de poids et les ensembles $\Gamma^-_{SAT}(v)$ et $\Gamma^+_{SAT}(v)$ des arcs saturés entrant en v et sortant de v. Soit a_v le premier arc de cet ensemble. On avait défini précédemment la variable $t(a)$ comme le temps en secondes avant saturation de a. Nous allons légèrement modifier cette définition. On définit $t(v)$ comme **l'instant** en secondes où l'arc a_v est saturé. Ce n'est donc plus une durée. Pour initialiser $t(v)$, si v n'est pas un terminal, $t(v) = +\infty$, et sinon $t(v) = \omega(a_v)$.

On associe également à v un tableau X_v de taille k : si x est un terminal, $X_v[x]$ est un booléen vrai si et seulement s'il existe un chemin d'arcs saturés reliant v à x, et $k(v)$ est le nombre de tels booléens ayant pour valeur vrai. On initialise $X_x[x]$ à vrai, $X_x[y]$ à faux pour tout terminal y différent de x et $k(x)$ à 1 si x est un terminal. Pour tout autre nœud v non terminal, et tout terminal x, $X_x[v]$ est initialisé à faux et 0.

Nous maintenons toujours à jour la variable t, qui dans cette implémentation aura son importance. t est initialisée à 0.

Enfin, nous définissons $\mathcal{F}$ comme un tas de Fibonacci qui contient chaque nœud v de G, classé avec $t(v)$. L'intérêt du tas de Fibonacci est sa capacité à rapidement insérer un élément, renvoyer le premier élément et surtout décroître la clé de tri de ses éléments. Or tant que a_v n'est pas saturé, l'instant $t(v)$ ne peut que diminuer.

Début de chaque itération : trouver le premier arc saturé. Si v_0 est le premier élément de $\mathcal{F}$, le prochain arc à saturer est $a_{v_0} = (u_0, v_0)$. On met maintenant à jour certaines des variables :

- $t = t(v_0)$
- On retire v_0 de $\mathcal{F}$.
- Soit a'_{v_0} le deuxième arc de $\Gamma^-(v_0)$ s'il existe. On retire a_{v_0} de $\Gamma^-(v_0)$. Ainsi a'_{v_0} devient le prochain arc saturé de v_0. On n'ajoute pas immédiatement cet arc aux ensembles $\Gamma^-_{SAT}(v_0)$ et $\Gamma^+_{SAT}(u_0)$ car il est possible qu'il soit marqué au lieu d'être saturé.
- Si $\Gamma^-(v_0)$ est non vide, on va calculer l'instant de saturation $t(v_0)$ du prochain arc saturé de $\Gamma^-(v_0)$. Puisque tous les arcs non saturés entrant en v ont toujours la même quantité de flot, alors le flot dans a'_{v_0} vaut $\omega(a_{v_0})$. Donc le flot restant à envoyer par v_0 pour remplir a'_{v_0} est la différence du poids entre ces deux arcs. La vitesse actuelle du flot passant par v_0 est $k(v_0)$ et n'est pas modifiée par la saturation (ou le marquage) de a_{v_0}. On met donc à jour $t(v_0)$ avec $t + \frac{\omega(a'_{v_0}) - \omega(a_{v_0})}{k(v_0)}$ et v_0 est réinséré dans $\mathcal{F}$ avec cette nouvelle valeur de $t(v_0)$.

Détection d'une dégénérescence du flot. La saturation de l'arc $a_{v_0} = (u_0, v_0)$ ne provoque pas de changements chez tous les nœuds : uniquement sur l'ensemble des nœuds pour lesquels il existe un chemin d'arcs saturés vers u_0. On note T_{u_0} cet ensemble. Encore une fois, si on suppose le flot avant saturation de cet arc non dégénéré, les nœuds de T_{u_0} et les arcs qui les relient constituent à chaque début d'itération un arbre d'ancêtres de u_0. L'ensemble T_{u_0} n'est pas maintenu pendant le déroulement de l'algorithme mais on peut le parcourir en largeur en temps linéaire : soit w un nœud de T_{u_0} visité par ce parcours, on ajoute à la liste des nœuds à visiter l'ensemble des prédécesseurs de w reliés à w par un arc de $\Gamma^-_{SAT}(w)$.

Pour détecter si le flot est dégénéré, on parcourt chaque nœud w de T_{u_0} comme précisé précédemment, et on regarde si un des terminaux auxquels w est relié par un arc saturé n'est pas également un terminal auquel v_0 est relié, c'est-à-dire s'il existe un terminal x pour lequel $X_{v_0}[x] = X_w[x] =$ vrai ? Dans ce cas, en saturant a_{v_0}, il existera nécessairement deux chemins distincts reliant w à x : un qui utilise a_{v_0} et un autre qui ne l'utilise pas. On peut remarquer que cette méthode détecte également les cycles constitués d'arcs saturés.

Si le flot est dégénéré, on marque l'arc a_{v_0}, on arrête l'itération en cours et on commence la suivante. Sinon on ajoute cet arc à $\Gamma^-_{SAT}(v_0)$ et $\Gamma^+_{SAT}(u_0)$.

Mise à jour des variables en cas de flot non dégénéré. Si le flot n'est pas dégénéré, on ajoute l'arc a_{v_0} à l'ensemble des arcs saturés. On peut désormais mettre à jours les variables telles que le débit et l'instant de saturation du premier arc entrant de tous les nœuds de T_{u_0}. On parcourt une nouvelle fois T_u, toujours en largeur depuis u_0. Pour tout sommet w de T_u, on pose $X_w[x] = X_w[x] \wedge X_{v_0}[x]$ et $k'(w) = k(w) + k(v_0) : k'(w)$ est le nombre de terminaux pour lesquels $X_w[x]$ est désormais vrai. Si $k(w) = 0$, alors aucun arc entrant de w n'a reçu de flot : $txx(w)$ vaut toujours $+\infty$. Le prochain arc à saturer est le premier arc a_w de $\Gamma^-(w)$. Dès la prochaine itération, cet arc recevra du flot avec un débit égal à $k'(w)$. On diminue donc dans $\mathcal{F}$ la clé $t(w)$ de w à $t + \frac{\omega(a_w)}{k'(w)}$. Sinon, si $k(w) \neq 0$, alors la valeur de $t(w)$ était finie. Connaissant cet instant, la valeur du temps t de cette itération et le débit précédent $k(w)$, on peut déduire la quantité de flot qu'il restait à envoyer dans a_w pour le saturer : $(t(w) - t) \cdot k(w)$. On peut donc mettre à jour $\mathcal{F}$ en diminuant la clé $t(w)$ à $t + \frac{(t(w)-t)\cdot k(w)}{k'(w)}$. Enfin, on met à jour X_w et $k(w)$ avec X'_w et $k'(w)$.

Création de T_0. Lorsque le flot a atteint la racine, il faut construire l'arborescence T_0 et la renvoyer. Puisque le flot n'est pas dégénéré, on est certain que le sous-graphe des arcs saturés descendants de r est une arborescence. Un simple parcours en largeur depuis la racine suffit à construire T_0 : si le parcours visite un nœud w (en commençant par r), il ajoute à la liste des nœuds à visiter l'ensemble des nœuds auxquels w est relié par un arc de $\Gamma^+_{SAT}(w)$.

4.4.2 Pseudo-code de l'implémentation

L'algorithme 5, nommé FLAC 2, est une implémentation qui suit les remarques précédentes. Nous verrons dans la section suivante que cette implémentation est correcte (vis-à-vis de l'implémentation de l'algorithme 4) et qu'elle est plus efficace (en terme de complexité temporelle). C'est cet algorithme qui a été évalué plus loin dans ce chapitre.

La figure 4.7 montre un exemple d'application de cette implémentation.

Nous montrons dans l'annexe A, en page 172, que l'implémentation FLAC 2 renvoie les mêmes solutions que l'implémentation FLAC.

4.4.3 Complexités temporelle et spatiale

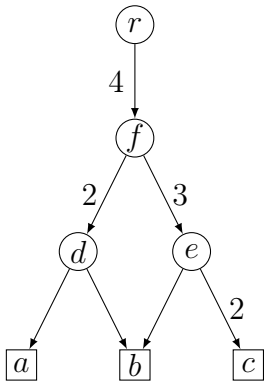
Lemme 4.4.1. *Les complexités temporelle et spatiale de l'algorithme 5 sont respectivement* $O(n^2 \log(n) + m \log(n) + mkn)$ *et* $O(nk + m)$.

Démonstration. Lors de l'initialisation, l'algorithme lance une boucle de n itérations où chaque nœud v va initialiser ses variables et être inséré dans $\mathcal{F}$. Parmi les variables à initialiser, il y a la liste $\Gamma^-(v)$. Un nœud v a au plus n prédécesseurs : donc le tri des arcs entrant en v nécessite $O(n \log(n))$ opérations. L'insertion dans un tas de Fibonacci se fait en temps constant amorti : $O(n)$. Ainsi l'initialisation se fait en temps $O(n^2 \log(n))$.

Il y a ensuite une boucle *tant que*, dans laquelle chaque itération, sauf la dernière, se compose de trois parties : la recherche de l'arc saturé, la détection de conflit et la mise à jour s'il n'y a pas conflit. À chaque itération, un arc est saturé ou marqué. On fait donc au plus m itérations. Durant la phase de recherche de l'arc saturé, on retire le premier élément du tas de Fibonacci, en temps amorti $O(\log(n))$. Les autres opérations du début de l'itération se font en temps constant. La recherche de conflit demande un parcours d'au plus n tableaux

Algorithme 5 FLow Algorithm Computation 2 (FLAC 2)

$t \leftarrow 0$
$\mathcal{F} \leftarrow []$ # Tas de Fibonacci
$\mathcal{M} \leftarrow \emptyset$ # Arcs marqués
Pour $v \in V$ **Faire**
    Construire $\Gamma^-(v)$
    $a_v \leftarrow \Gamma^-(v)[0]$
    Si $v \in X$ **Alors**
        $t(v) \leftarrow \omega(a_v)$
        $X_v[v] \leftarrow$ Vrai
        $k(v) \leftarrow 1$
    Sinon
        $t(v) \leftarrow +\infty$
        $k(v) \leftarrow 0$
    Ajouter $(v, t(v))$ à $\mathcal{F}$
Tant que Vrai **Faire**
    $(v_0, t(v_0)) \leftarrow$ le premier élément de $\mathcal{F}$, que l'on retire.
    $a_0 = (u_0, v_0) \leftarrow$ le premier élément de $\Gamma^-(v_0)$, que l'on retire.
    $t \leftarrow t(v_0)$
    Si $u_0 = r$ **Alors**
        Ajouter a_0 à $\Gamma^-_{SAT}(v_0)$ et $\Gamma^+_{SAT}(u_0)$
        Renvoyer l'arborescence T_0 d'arcs saturés reliant r à des terminaux
    Si $\Gamma^-(v_0) \neq \emptyset$ **Alors**
        $a'_0 \leftarrow \Gamma^-(v_0)[0]$
        $t(v_0) \leftarrow t + \frac{\omega(a'_{v_0}) - \omega(a_{v_0})}{k(v_0)}$
        Ajouter $(v_0, t(v_0))$ à $\mathcal{F}$
    Pour $w \in T_{u_0}$ et $x \in X$ **Faire**
        Si $(X_v[x] = X_w[x] =$ vrai) ajouter a_0 à $\mathcal{M}$ et passer à l'itération suivante.
    Ajouter a_0 à $\Gamma^-_{SAT}(v_0)$ et $\Gamma^+_{SAT}(u_0)$
    Pour $w \in T_{u_0}$ **Faire**
        Pour tout $(x \in X)$ $X_w[x] = X_w[x] \wedge X_{v_0}[x]$
        $k'(w) \leftarrow k(w) + k(v_0)$
        Si $k(w) = 0$ **Alors**
            $t(w) \leftarrow t + \frac{\omega(a_w)}{k'(w)}$
        Sinon
            $t(w) \leftarrow t + \frac{(t(w) - t) \cdot k(w)}{k'(w)}$
        $k(w) \leftarrow k'(w)$



Itération	v_0	a_0	t	$\mathcal{M}$	$t(a)$	$t(b)$	$t(c)$	$t(d)$	$t(e)$	$t(f)$
0			0	$\emptyset$	1	1	2	$+\infty$	$+\infty$	$+\infty$
1	a	(d,a)	1	$\emptyset$	1	1	2	3	$+\infty$	$+\infty$
2	b	(d,b)	1	$\emptyset$	-	1	2	2	$+\infty$	$+\infty$
3	b	(e,b)	1	$\emptyset$	-	1	2	2	4	$+\infty$
4	c	(e,c)	2	$\emptyset$	-	-	2	2	3	$+\infty$
5	d	(f,d)	2	$\emptyset$	-	-	-	2	3	4
6	e	(f,e)	3	$\{(f,e)\}$	-	-	-	-	3	4
7	f	(r,f)	4	$\{(f,e)\}$	-	-	-	-	-	4

FIGURE 4.7 – *Exemple d'application de l'algorithme FLAC 2. Le tableau précise le nœud retiré de $\mathcal{F}$ et l'arc saturé au début de chaque itération, et les valeurs des variables t, $\mathcal{M}$ et $t(v)$ pour chaque nœud v à la fin de l'itération. À la fin, le graphe G_{SAT} contient tous les arcs sauf (f,e) qui est marqué. L'arbre T_0 renvoyé est donc $\{(r,f),(f,d),(d,a),(d,b)\}$.*

de taille k, donc en $k \cdot n$ opérations. Enfin, pour la mise à jour, on refait un parcours de ces même tableaux pour les mettre à jour ; tout autre opération prend un temps constant. L'ensemble des itérations de cette boucle, sauf la dernière, nécessite donc $O(m\log(n) + mkn)$ opérations.

Enfin, la dernière itération renvoie l'arbre T_0, calculé en parcourant en largeur le graphe depuis r, via les arcs saturés, donc en temps $O(n)$.

Ainsi, la complexité temporelle de l'algorithme est $O(n^2\log(n) + m\log(n) + mkn)$.

Les structures de données utilisées sont : le tas $\mathcal{F}$ de taille au plus n, l'ensemble des listes Γ^-, Γ^-_{SAT} et Γ^+_{SAT} dont la taille totale est $3 \cdot m$, l'ensemble des tableaux X_v dont la taille totale est kn et le graphe G dont la taille est $n + m$. Toute autre variable prend une place constante dans la mémoire. La complexité spatiale est donc $O(nk + m)$. □

En reprenant les démonstrations des théorèmes et à l'aide du lemme 4.4.1, on démontre les théorèmes suivants.

Théorème 4.4.2. *En utilisant l'algorithme 5 plutôt que l'algorithme 4 pour implémenter l'expérience, les complexités temporelle et spatiale de $Greedy_{FLAC}$ sont respectivement $O(k \cdot (n^2\log(n) + m\log(n) + mkn))$ et $O(nk + m)$.*

Théorème 4.4.3. *En utilisant l'algorithme 5 plutôt que l'algorithme 4 pour implémenter l'expérience, les complexités temporelle et spatiale de* $\text{Greedy}^{\triangleright}_{FLAC}$ *sont respectivement* $O(k \cdot (n^2 \log(n) + kn^3))$ *et* $O(nk + n^2)$.

4.5 Croissance des variables duales

Nous rappelons brièvement la technique d'ascension du dual, décrite dans le chapitre 2 et montrons que notre expérience de pensée est une manière d'utiliser cette technique.

Programme linéaire (PP2) : Description de DST par coupes

Minimiser	$\sum_{a \in A} \omega(a) \cdot x_a$				
s.c.	$\sum_{a \in \Gamma^-(S)} x_a$	$\geq$	1	pour	$S \subseteq V$ tel que $r \notin S$ et $S \cap X \neq \emptyset$
	x_a	$\in$	$\{0, 1\}$	pour	$a \in A$

Programme linéaire (DP3) : Description de DST par coupes (programme dual)

Maximiser	$\sum_{\substack{S \subseteq V \\ r \notin S \\ S \cap X \neq \emptyset}} y_S$				
s.c.	$\sum_{a \in \Gamma^-(S)} y_S$	$\leq$	$\omega(a)$	pour	$a \in A$
	y_S	$\geq$	0	pour	$S \subseteq V$ tel que $r \notin S$ et $S \cap X \neq \emptyset$

La contrainte d'un arc a est dite *saturée* si on ne peut plus augmenter les variables duales associées à cette contrainte. La technique d'ascension du dual consiste à choisir un ou plusieurs ensembles S, et d'augmenter les variables y_S jusqu'à saturation d'une contrainte d'un arc a. On ajoute a à la solution que l'on construit et on recommence jusqu'à ce que la solution soit réalisable.

La quantité d'eau dans un arc représente en réalité la valeur de la variable x_a. Les ensembles S dont on augmente les variables duales sont, pour chaque terminal x, l'ensemble S_x des nœuds qui peuvent atteindre x avec un chemin d'arcs saturés. Ces variables sont toutes augmentées en même temps uniformément. Dans l'expérience de pensée, quand deux terminaux x_1 et x_2 envoient du flot dans le même arc (u, v), le débit de ce flot est doublé. Du point de vue du programme dual, les ensembles S_{x_1} et S_{x_2} contiennent tous deux le nœud v, donc la contrainte relative à l'arc (u, v) approche de la saturation deux fois plus vite.

La différence avec les heuristiques dans [Won84, Mel07] est principalement la définition des ensembles dont on augmente la variable duale. Si FLAC en utilise à tout instant un par terminal, ces autres heuristiques regroupent les ensembles qui se rejoignent. Ainsi, le débit des arcs est toujours 1, quel que soit le nombre de terminaux auxquels ils sont reliés.

4.6 Évaluation des performances

Nous effectuons dans cette section deux évaluations séparées : celle de la qualité des solutions renvoyées et celle des temps de calculs. La reproductibilité des expériences présentées ci-après est précisée dans l'annexe B.

4.6.1 Évaluation de la qualité des solutions renvoyées

Afin d'évaluer les algorithmes Greedy_{FLAC} et $\text{Greedy}^{\triangleright}_{FLAC}$, nous les avons testés sur différents jeux d'instances que nous avons créés à partir d'instances non orientées.

Il existe un ensemble disponible en ligne répertoriant environ un millier d'instances du problème de l'arbre de Steiner (UST) classées en différents groupes. Il s'agit de la bibliothèque SteinLib [KMV01]. Chaque groupe d'instances fait référence à une application du problème ou à un type particulier de graphes, permettant d'évaluer des algorithmes exacts ou d'approximation sur ce problème. À chacune de ces instances (sauf rares exceptions) est associée la valeur d'une solution optimale. Les solutions elles-mêmes ne sont toutefois pas disponibles. Les divers groupes de SteinLib sont décrits plus en détails dans l'annexe C.

À l'inverse, il n'existe aucune liste d'instances pour le problème de l'arborescence de Steiner. La plupart des algorithmes traitant de ce problème doivent donc soit générer leurs propres instances, soit transformer les instances non orientées en instances orientées. Lorsque c'est le second choix qui est fait, la solution proposée consiste généralement à remplacer chaque arête par deux arcs opposés de même poids. Si une telle méthode fournit aisément et rapidement une instance orientée de même solution optimale, on peut critiquer ce procédé en remarquant qu'il restreint fortement les instances de l'algorithme aux applications modélisées par le problème de l'arbre de Steiner. Il n'est pas certain qu'un réseau de télécommunication puisse être modélisé par un graphe où toute transmission a un poids égal à la transmission opposée. Les graphes non orientés semblent donc trop restrictifs pour tester des algorithmes prévus pour des instances plus générales.

Nous proposons ici, en plus d'une étude sur les graphes biorientés où nos résultats pourront être comparés avec ceux d'autres algorithmes de la littérature, deux autres manières de transformer les instances non orientées en instances orientées : la première conçoit des instances sans circuits et l'autre des instances fortement connexes non biorientées.

4.6.2 Description des instances générées

Les 3 classes d'instances ont été générées à partir des instances de SteinLib des groupes suivants : WRP3, WRP4, ALUE, ALUT, DIW, DXMA, GAP, MSM, TAQ, LIN, SP, X, MC, I080 à I640, ES10FST à ES10000FST, B,C,D,E, P6E et P6Z. Les dix premiers groupes contiennent environ 270 instances dont le graphe est une grilles contenant éventuellement des trous. Le groupe X contient 3 instances complètes. Les groupes ES10FST à ES10000FST contiennent environ 200 instances rectilinéaires. Tous les autres groupes contiennent environ 520 instances peu denses. L'annexe C contient une brève description de chaque groupe. D'autres groupes comme P4E et P4Z n'ont pas été utilisés car les fichiers décrivant les instances étaient erronés ou incomplets.

Classe A : Instances biorientées. Comme expliqué précédement, ces instances sont générées en remplaçant chaque arête du graphe par deux arcs opposés de même poids. La racine est choisie aléatoirement parmi tous les terminaux. À partir de toute solution réalisable de l'instance non orientée, on peut construire une arborescence de même poids dans l'instance générée, et inversement.

Nous avons généré une instance biorientée pour chaque instance non orientée dont nous disposions. La classe A contient donc 999 instances.

Classe B : Instances sans circuits. Pour transformer une instance non orientée $\mathcal{I}$ en instance de cette classe, il faut premièrement calculer, à l'aide d'un algorithme exact, une solution optimale T^* de $\mathcal{I}$. Les instances non orientées de SteinLib fournissent les valeurs des solutions optimales et non pas les solutions optimales. Il a donc fallu les calculer.

À partir de cette solution T^*, on choisit un nœud qui sera la racine de notre instance orientée. Puis une arborescence est créée en orientant depuis la racine toutes les arêtes de T^* vers les feuilles. Ensuite on parcourt le reste des nœuds du graphe en largeur depuis la racine. Pour chaque nœud visité, et chaque arête reliée à ce nœud qui n'est pas déjà orientée, celle-ci est orientée du nœud vers ses voisins.

Toute solution réalisable de $\mathcal{I}$ est potentiellement transformée en arborescence de même poids dans l'instance générée. Puisque T^* a été orienté de sorte à générer une arborescence de même poids, cette arborescence est optimale.

Nous avons généré une instance sans circuits pour 353 des instances non orientées. Pour les instances restantes, c'est le calcul de la solution optimale qui n'a pas encore été réalisé.

Classe C : Instances fortement connexes. Pour transformer une instance non orientée $\mathcal{I}$ en instance de cette classe, il faut, comme dans la classe B, premièrement calculer à l'aide d'un algorithme exact une solution optimale T^* de $\mathcal{I}$, et choisir un nœud de cet arbre comme racine et orienter T^* à partir de cette racine. On oriente ensuite aléatoirement tout arc restant de l'instance.

Enfin, pour obtenir comme souhaité une instance fortement connexe, on ajoute un arc aléatoirement d'un nœud u vers un nœud v auquel il n'est pas déjà connecté. Le poids de cet arc est le poids d'un plus court chemin reliant u à v dans $\mathcal{I}$. On recommence cette opération jusqu'à ce qu'il existe un chemin entre tout couple de nœuds dans le graphe généré.

On vérifie de même qu'aucune solution réalisable de poids plus petit que T^* n'a été générée dans cette instance. Puisque T^* a été orienté de sorte à générer une arborescence de même poids, cette arborescence est optimale.

Nous avons généré autant d'instances fortement connexes que d'instances sans circuits, soit 353 instances.

4.6.3 Description des expériences et résultats

Nous avons comparé six algorithmes d'approximation. Ces algorithmes ont été implémentés avec le langage Java. Le code source est disponible en ligne à l'adresse `https://github.com/mouton5000/DSTAlgoEvaluation`.

Les deux premiers algorithmes testés sont Greedy_{FLAC} et $\text{Greedy}^{\triangleright}_{FLAC}$ qui ont été implémentés comme décrit dans la section 4.4. Le troisième est l'algorithme des plus courts chemins, SHP, qui a été implémenté en utilisant l'algorithme de Dijkstra pour calculer en même temps

tous les plus courts chemins reliant la racine à chaque terminal : c'est l'algorithme le plus rapide qui existe aujourd'hui. Nous avons également testé une variante de l'algorithme des plus courts chemins, SHP 2, qui calcule un plus court chemin entre la racine et tous les terminaux, ajoute ce chemin à la solution et met les poids de tous ses arcs à 0 avant de chercher un second plus court chemin. Cette modification des poids favorise la mutualisation des chemins. Cet algorithme reste toutefois une k-approximation. Le cinquième algorithme de cette liste est la plus rapide des heuristiques d'ascension du dual, DuAs, qui a été implémenté comme décrit dans l'article [Won84]. Enfin, nous nous comparons à l'algorithme de Charikar, avec un paramètre d'entrée égal à 2, que nous nommons CH_2. Il a été implémenté avec l'algorithme de Roos modifié, décrit dans [HHT06]. Nous n'avons pas testé l'algorithme de Charikar avec un paramètre d'entrée plus élevé à cause du temps de calcul nécessaire à son exécution.

Remarque 11. L'article [HHT06] décrit une meilleure méthode que l'algorithme des plus courts chemins et que l'algorithme de Roos. Cependant, nous n'avons pas été en mesure de vérifier leurs dires. Leur algorithme fonctionne, comme pour celui de Charikar avec, en entrée, un paramètre de profondeur. Or ils testent leur algorithme avec ce paramètre fixé à 2. D'après leur implémentation, si le paramètre vaut 2, l'algorithme doit renvoyer la meilleure solution parmi l'algorithme des plus courts chemins et celui de Roos, ce qui est incohérent avec leur évaluation des performances. Il ne fait donc pas partie des algorithmes testés dans cette évaluation.

Nous avons effectué deux tests différents.

Distance relative entre le poids des solutions renvoyées. Les algorithmes ont été testés sur les trois différentes classes. On compare chaque algorithme à Greedy_{FLAC} indépendamment les uns des autres.

Nous avons mesuré pour chaque test deux paramètres. Premièrement la distance relative entre les poids des solutions renvoyées par les algorithmes : connaissant le poids ω de la solution renvoyée par Greedy_{FLAC} et le poids ω_2 de la solution renvoyée par l'autre algorithme, la distance relative δ vaut $\frac{\omega-\omega_2}{\omega_2}$.

Le tableau 4.8 indique les résultats obtenus, sachant que nous avons retiré de chaque test les instances de la classe A pour lesquelles la mémoire requise pour l'algorithme comparé avec Greedy_{FLAC} dépassait les capacités de la machine de test. Ainsi :

- pour $\text{Greedy}^{\triangleright}_{FLAC}$, la classe A ne contient que 471 instances ;
- pour SHP, elle contient les 999 instances ;
- pour SHP 2, elle ne contient que 996 instances ;
- pour DuAs, elle ne contient que 977 instances ;
- pour CH_2, elle ne contient que 923 instances ;

On peut remarquer que Greedy_{FLAC} et SHP ont toujours donné un résultat, quelle que soit l'instance. De plus, pour les classes B et C, tous les algorithmes ont renvoyé un résultat pour les 353 instances de chaque classe.

De cette évaluation, nous observons que :

- $\text{Greedy}^{\triangleright}_{FLAC}$ est strictement plus utile que Greedy_{FLAC} dans moins de 10% des cas et d'après les colonnes $\delta = 0$ et $|\delta| < 5\%, 10\%, 15\%$, les poids des solutions renvoyées par les deux algorithmes sont souvent égaux ou proches ;
- concernant la classe A, l'algorithme des plus courts chemins SHP 2 est le concurrent le plus efficace avec 33% des instances renvoyant un meilleur poids ;

Algorithme comparé		$\delta < 0$	$\delta = 0$	$\delta > 0$	$\lvert\delta\rvert < 5\%$	$\lvert\delta\rvert < 10\%$	$\lvert\delta\rvert < 15\%$
$\text{Greedy}^{\triangleright}_{FLAC}$	A	54%	34%	10%	83%	98%	99%
	B	17%	71%	10%	94%	98%	99%
	C	16%	76%	6%	95%	98%	99%
SHP	A	84%	14%	0%	16%	20%	33%
	B	95%	4%	0%	11%	25%	43%
	C	84%	13%	1%	26%	40%	53%
SHP 2	A	59%	7%	33%	61%	78%	82%
	B	71%	20%	7%	47%	66%	73%
	C	59%	35%	4%	57%	71%	76%
DuAs	A	87%	3%	8%	44%	53%	62%
	B	49%	28%	21%	62%	77%	89%
	C	58%	26%	14%	55%	66%	73%
CH_2	A	90%	8%	0%	27%	36%	45%
	B	81%	17%	0%	26%	35%	49%
	C	82%	17%	0%	23%	32%	44%

Table 4.8 – *Comparaison des poids ω renvoyé par l'algorithme FLAC et ω_2 renvoyé par l'algorithme comparé sur chaque classe de test. Les colonnes de gauche indiquent le nombre d'instances pour lesquelles ω est plus petit, égal ou plus grand que ω_2. Les trois colonnes de droite donnent une idée de l'écart séparant les poids.*

- concernant les classes B et C, c'est DuAs qui se trouve être le meilleur concurrent ;
- les algorithmes SHP et CH_2 ne renvoient jamais ou très rarement une meilleure solution que Greedy_{FLAC} et, d'après les trois dernières colonnes, la différence relative est supérieure à 15%, donc la qualité des solutions renvoyées par Greedy_{FLAC} est meilleure ;

Le temps de calcul de $\text{Greedy}^{\triangleright}_{FLAC}$ est bien plus élevé que celui de Greedy_{FLAC}, car il travaille dans le graphe orienté complet des plus courts chemins. D'après cette constatation, et d'après le tableau 4.8, il vaut mieux, en pratique, utiliser Greedy_{FLAC} que $\text{Greedy}^{\triangleright}_{FLAC}$. De ce fait, cet algorithme n'apparait plus dans les tests à venir. Cela permet entre autres de comparer les algorithmes sur plus d'instances dans le cas de la classe A.

Si Greedy_{FLAC} donne de meilleurs résultats en pratique, contrairement à ce que laissent suggérer leurs rapports d'approximation, c'est parce que l'instance des plus courts chemins offre à $\text{Greedy}^{\triangleright}_{FLAC}$ des raccourcis pour relier la racine aux terminaux, qui n'existent pas dans le graphe original. Greedy_{FLAC} ne peut donc les employer et cela favorise la mutualisation d'arcs, même si dans certains cas cela favorise aussi des blocages dus à un flot dégénéré et un arc marqué.

D'après les données du tableau 4.8, Greedy_{FLAC} semble renvoyer de meilleurs résultats que les autres algorithmes. Mais ce tableau n'indique pas si les algorithmes sont proches ou non de la solution optimale.

Comparaison des marges d'erreur des algorithmes. Chaque algorithme, excepté Greedy$^{\triangleright}_{FLAC}$, a été testé sur les trois classes d'instances. Nous avons retiré de ce test les instances de la classe A pour lesquelles la mémoire requise pour un des algorithmes dépassait les capacités de la machine de test, elle ne contient donc que 918 instances pour ce test. Nous avons mesuré pour chaque test la marge d'erreur : connaissant le poids ω^* de la solution optimale de l'instance et le poids ω de la solution renvoyée par l'algorithme, la marge d'erreur e vaut $\frac{\omega-\omega^*}{\omega^*}$.

Pour chaque groupe, nous avons tracé 2 séries de courbes. Pour chaque algorithme, nous avons tracé pour chaque valeur possible (en %) le nombre d'instances pour lesquelles nous avons mesuré une marge d'erreur inférieure à cette valeur. Plus la courbe s'élève rapidement, plus l'algorithme a résolu d'instances avec une faible marge d'erreur [1].

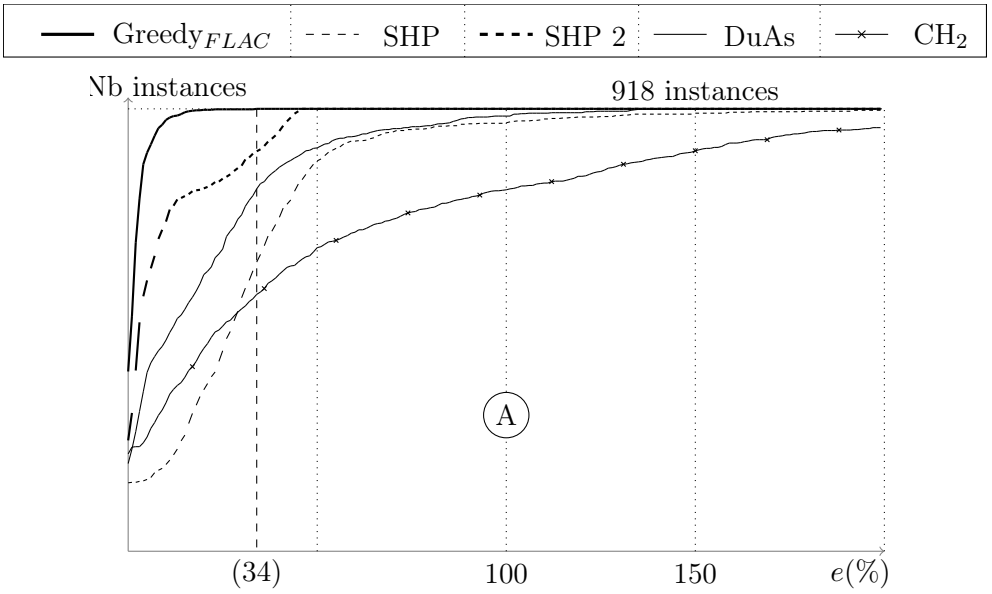


Figure 4.9 – *Fonction de répartition des marges d'erreurs de chaque algorithme sur toutes les instances de la classe A. Il est précisé en abscisse entre parenthèse la marge d'erreur maximale de Greedy$_{FLAC}$: l'écart entre une solution renvoyée par Greedy$_{FLAC}$ et une solution optimale ne dépasse pas 34%.*

Résultats sur les instances de la classe A. Les résultats sont présentés en figure 4.9. Un point important sur cette figure que la courbe de Greedy$_{FLAC}$ reste constamment au dessus des quatre autres courbes, ce qui signifie que, pour une valeur fixée, le nombre d'instance résolues avec une marge d'erreur inférieure à cette valeur est supérieur pour Greedy$_{FLAC}$ comparé aux autres algorithmes. On peut observer en outre que la marge d'erreur maximale pour Greedy$_{FLAC}$ est inférieure à celles des autres algorithmes. Le second algorithme après Greedy$_{FLAC}$ est SHP 2. Ces résultats sont cohérents avec ceux donnés dans le tableau 4.8.

1. Ce procédé a le défaut de mélanger toutes les instances. Il est impossible de savoir si un algorithme résout plus facilement certaines instances que d'autres. C'est pourquoi nous décrivons en annexe C les mêmes expériences sur des groupes plus restreints d'instances rassemblées par catégories dans le benchmark SteinLib.

Dans le détail présenté en annexe C, on observe les résultats généraux suivants :

- Pour les groupes I080 à I640, Greedy$_{FLAC}$ est l'algorithme le plus performant de tous. Sa marge d'erreur est inférieure à 5% pour au moins neuf dixième des instances tandis que celle des autres algorithmes dépasse 15% pour la moitié des instances. Un comportement similaire est observé pour les groupes MC et SP.
- Pour les groupes B,C,D et E, les algorithmes les plus performants sont Greedy$_{FLAC}$, SHP 2 puis DuAs. Les deux premières courbes sont presque confondues. On remarque que pour un peu moins d'une dizaine d'instances, la marge d'erreur de Greedy$_{FLAC}$ double (jusqu'à 20% environ), alors que celle de SHP 2 reste en dessous de 10 à 15%. Un comportement similaire est observé pour les groupes X, P6E et P6Z.
- Pour les groupes d'instances rectilinéaires, ESXFST, les algorithmes les plus performants sont Greedy$_{FLAC}$, SHP 2 et DuAs, et renvoient des résultats similaires (moins de 10% de marge d'erreur maximale quel que soit l'algorithme).
- Pour les groupes WRP3, WRP4, ALUE, ALUT, DIW, DXMA, GAP, MSM, TAQ et LIN, c'est-à-dire les grilles contenant des trous, l'algorithme SHP 2 est plus performant que l'algorithme Greedy$_{FLAC}$. L'écart entre les marges d'erreur des deux algorithmes ne dépasse jamais 10%, et la marge d'erreur maximale de Greedy$_{FLAC}$ est 20%. Tous les autres algorithmes renvoient de très bons résultats sur les groupes WRP3 et WRP4 (les courbes de SHP et Greedy$_{FLAC}$ sont confondues), mais sont dépassés sur tous les autres groupes.

On peut déduire de ces constatations que les causes qui donnent à Greedy$_{FLAC}$ de bons résultats sur la courbe intégrant toutes les instances étudiées sont les suivantes :

- Greedy$_{FLAC}$ est toujours l'algorithme le plus performant ou le second algorithme le plus performant si on observe les groupes de manière plus détaillées ;
- Greedy$_{FLAC}$ donne de très bons résultats sur les 400 instances des groupes I080 à I640 ;
- sur les autres instances, lorsque Greedy$_{FLAC}$ n'est pas le meilleur algorithme, ses résultats sont proches de ceux de l'algorithme le plus performant.

Résultats sur les instances de la classe B. Les résultats sont présentés en figure 4.10. Encore une fois, on observe une cohérence avec le tableau 4.8 : Greedy$_{FLAC}$ présente de meilleurs résultats, suivi par DuAs, puis par SHP 2. Si on compare les classes A et B, on observe nettement sur cette figure un échange entre les courbes des algorithmes DuAs et SHP 2. Ceci s'explique par la disparition, dans la classe B, des circuits présents dans la classe A. Cette perte des circuits entraîne la suppression de nombreux chemins reliant la racine aux terminaux et donc la suppression de nombreuses solutions réalisables. De plus, l'orientation des arcs n'est pas arbitraire, les chemins proviennent tous de la racine. Ainsi, il est plus simple pour un algorithme d'ascension du dual de relier la racine et les terminaux par des solutions de faible poids. La solution renvoyée par Greedy$_{FLAC}$ ou DuAs est donc meilleure dans la classe B que dans la classe A. À l'inverse, cette perte de chemins peut défavoriser l'algorithme des plus courts chemins. En effet, si parmi les chemins supprimés se trouvaient des plus courts chemins, SHP 2 se voit dans l'obligation d'emprunter d'autres chemins éventuellement disjoints les uns des autres et de dégrader la qualité de la solution renvoyée. La solution renvoyée est donc de moins bonne qualité dans la classe B que dans la classe A.

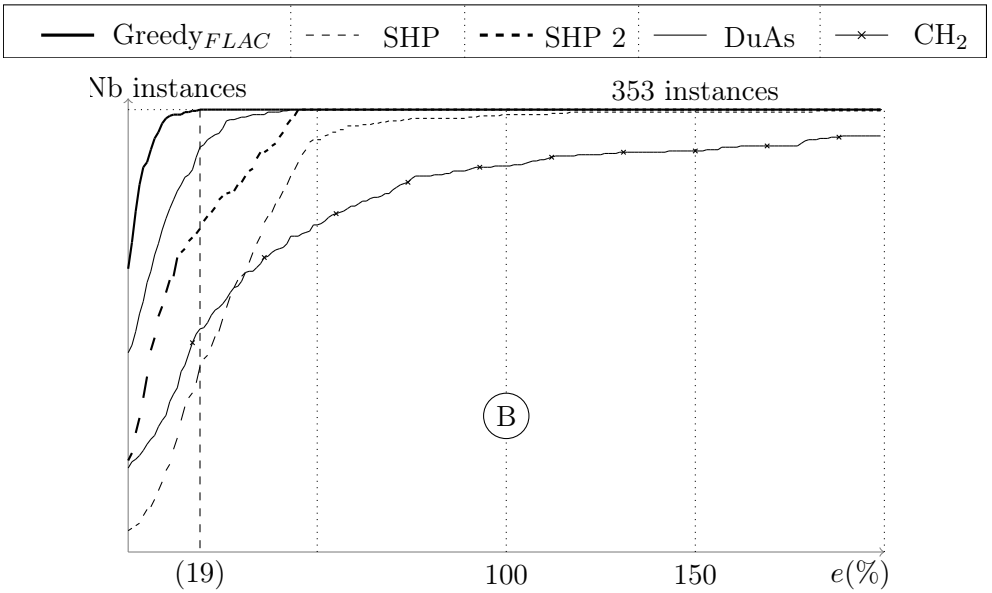


FIGURE 4.10 – *Fonction de répartition des marges d'erreurs de chaque algorithme sur toutes les instances de la classe B. Il est précisé en abscisse entre parenthèse la marge d'erreur maximale de Greedy$_{FLAC}$: l'écart entre une solution renvoyée par Greedy$_{FLAC}$ et une solution optimale ne dépasse pas 19%.*

Dans le détail présenté en annexe C, on observe les résultats généraux suivants :

- Pour les groupes I080 à I640, Greedy$_{FLAC}$ est l'algorithme le plus performant de tous. On remarque une amélioration de DuAs comparé à la classe A, mais ses performances restent en dessous de celles de Greedy$_{FLAC}$.
- Pour toutes les instances des groupes MC et SP et tous les algorithmes, à quelques exceptions près, une solution optimale a été renvoyée.
- Pour les groupes B et D, DuAs est plus performant que Greedy$_{FLAC}$ et SHP 2, renvoyant plus souvent une solution optimale. Pour le groupe C, Greedy$_{FLAC}$ renvoie toujours une solution optimale, ce qui n'est pas le cas de DuAs ni de SHP 2.
- Les résultats de Greedy$_{FLAC}$, SHP 2 et DuAs sur les groupes P6E et P6Z sont comparables. Leur marges d'erreurs sont très souvent inférieures à 5%.
- Pour les groupes WRP3, WRP4, ALUE, ALUT, DIW, DXMA, GAP, MSM, TAQ, et LIN, les algorithmes Greedy$_{FLAC}$, DuAs et SHP 2 donnent de meilleurs résultats pour la classe B que pour la classe A. On peut remarquer que ces instances, qui sont des grilles biorientées dans la classe A sont transformées ici en graphes sans circuits peu denses.
- Pour les groupes ESXFST, si on compare les classes A et B, Greedy$_{FLAC}$ renvoie sensiblement les mêmes résultats, DuAs renvoie de meilleurs résultats et SHP 2 renvoie de moins bons résultats. Tout comme précédemment, les graphes rectilinéaires deviennent des graphes sans circuits peu denses, ce qui explique l'amélioration des résultats de DuAs et la stagnation des résultats de Greedy$_{FLAC}$.

On peut déduire de ce détail que les causes qui donnent à Greedy$_{FLAC}$ de bons résultats sur la courbe intégrant toutes les instances étudiées sont, premièrement, les bons résultats sur

les groupes I080 à I640, qui constituent environ deux tiers du jeux de test de la classe B, et secondement, le faible écart séparant les résultats de Greedy$_{FLAC}$ et ceux de DuAs quand ces derniers sont meilleurs. S'il était possible de transformer toutes les instances de la bibliothèque en instance de la classe B, il serait fort probable que l'on observe un rapprochement des trois courbes DuAs, SHP 2 et Greedy$_{FLAC}$.

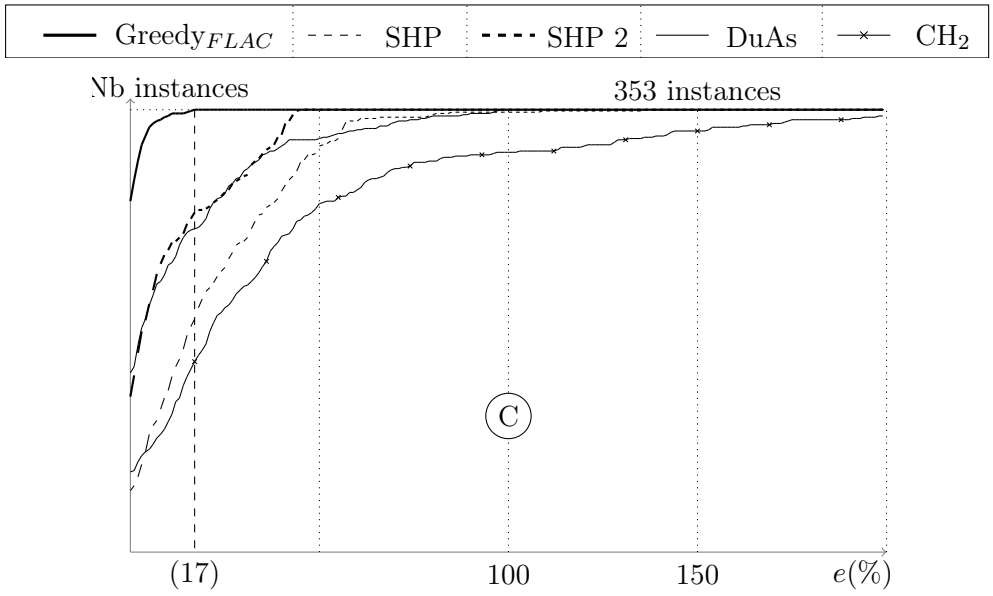


Figure 4.11 – *Fonction de répartition des marges d'erreurs de chaque algorithme sur toutes les instances de la classe C. Il est précisé en abscisse entre parenthèse la marge d'erreur maximale de Greedy$_{FLAC}$: l'écart entre une solution renvoyée par Greedy$_{FLAC}$ et une solution optimale ne dépasse pas 17%.*

Résultats sur les instances de la classe C. Les résultats sont présentés en figure 4.11. De même que pour les classes A et B, on observe une cohérence avec le tableau 4.8 : Greedy$_{FLAC}$ présente de meilleurs résultats, suivi par DuAs et SHP 2. En comparant ces courbes avec celles des classes A et B, on remarque que Greedy$_{FLAC}$ a de meilleurs résultats que dans la classe A et équivalents à ceux de la classe B. SHP 2 a des résultats moins bons que ceux de la classe A et équivalents à ceux de la classe B. En orientant arbitrairement les arcs du graphes, nous avons également supprimé des plus court chemins empruntés par SHP 2 dans la classe A, comme c'était déjà le cas dans la classe B. Enfin, DuAs a des résultats meilleurs que dans la classe A et moins bon que dans la classe B. L'orientation arbitraire a pour effet de supprimer une partie des solutions réalisables, ce qui favorise DuAs. Mais, dans certains cas, cette orientation n'est pas aussi efficace que l'orientation de la classe B. Puisqu'elle est arbitraire, il peut arriver que DuAs emprunte de nombreuses voies erronées dans le graphe avant d'atteindre la racine. Greedy$_{FLAC}$ est moins soumis à ce phénomène grâce à l'accélération du flot dans un arc relié à plusieurs terminaux. C'est pour cela que l'on observe, pour DuAs, une marge d'erreur dépassant les 100% sur une petite portion des instances.

Dans le détail présenté en annexe C, on observe les résultats généraux suivants :
- Pour les groupes I080 à I640, Greedy$_{FLAC}$ est l'algorithme le plus performant de tous.
- Pour l'unique instance du groupe MC de la classe C, Greedy$_{FLAC}$ renvoie une solution optimale, ce qui n'est pas le cas des autres algorithmes.
- Pour toutes les instances des groupes SP et tous les algorithmes, à quelques exceptions près, une solution optimale a été renvoyée.
- Pour le groupe B, les algorithmes les plus performants sont DuAs puis Greedy$_{FLAC}$ puis SHP 2. Pour les groupes C et D, ces trois algorithmes renvoient toujours une solution optimale, sauf à quelques exception près.
- Les résultats de Greedy$_{FLAC}$, SHP 2 sur les groupes P6E et P6Z sont comparables, avec des marges d'erreurs souvent inférieures à 5%. C'est DuAs qui est le plus performant.
- Pour les groupes WRP3, WRP4, tous les algorithmes renvoient une solution optimale ou très proche de l'optimale.
- Pour les groupes ALUE, ALUT, DIW, DXMA, GAP, MSM, TAQ, et LIN, les algorithmes les plus performants sont Greedy$_{FLAC}$ puis SHP 2 puis DuAs.
- Pour les groupes ESXFST, Greedy$_{FLAC}$ et DuAs ont pour toutes les instances des marges d'erreurs très faibles (moins de 3%). Celles de SHP 2 sont un peu plus élevées.

Les causes qui donnent à Greedy$_{FLAC}$ de bons résultats sur la courbe intégrant toutes les instances étudiées sont donc les mêmes que pour la classe B.

4.6.4 Évaluation des temps de calcul

Pour cette évaluation, nous avons généré des graphes aléatoires plutôt que d'utiliser les instances des classes A,B et C. Nous cherchons ici à connaître l'impact du nombre de nœuds, du nombre d'arcs et du nombre de terminaux sur le temps de calcul. Ces paramètres ne sont pas maîtrisés sur les instances de SteinLib, qui sont trop disparates. Pour chaque instance, nous avons testé tous les algorithmes, à l'exception de Greedy$^{\triangleright}_{FLAC}$ dont le temps de calcul dépasse de très loin celui des autres.

Pour générer chaque graphe, nous disposions de 3 paramètres : le nombre de nœuds n, la probabilité p de relier deux arcs et la densité de terminaux $\frac{k}{n}$. Pour chaque génération, nous avons ajouté n nœuds à une instance vide, le premier est la racine et les k suivants sont les terminaux, puis, pour chaque couple (u, v) de nœuds, nous avons ajouté l'arc (u, v) avec une probabilité p. Le poids de cet arc est choisi aléatoirement uniformément entre 0 et un maximum, lui-même choisi entre 1 et 100. On peut remarquer que la probabilité p est égale à l'espérance de $\frac{m}{n^2}$. Nous avons généré 10500 instances de 50 nœuds, 5250 instances de 100 nœuds et 1050 instances de 250 nœuds. Le tableau 4.12 indique les résultats obtenus.

On observe que SHP est l'algorithme le plus rapide, suivi par Greedy$_{FLAC}$ qui reste compétitif, d'après ses temps moyens et ses écarts-types, suffisamment faibles pour rarement dépasser quelques centaines de millisecondes sur des graphes denses de 250 nœuds.

Les algorithmes SHP 2 and CH_2 sont plus lents que les autres, à cause de k applications distinctes de l'algorithme de Dijkstra pour calculer des plus courts chemins entre des nœuds. On peut remarquer également le fort écart-type de DuAs, c'est dû à la forte part d'aléatoire contenue dans cet algorithme. En effet, son exécution dépend de l'ordre dans lequel il traite les terminaux.

n	p	$\frac{k}{n}$	Greedy$_{FLAC}$		SHP		SHP 2		DuAs		CH_2	
			moy.	écart-t.	moy.	écart-t.	moy.	écart-t.	moy.	écart-t.	moy.	écart-t.
50	0.3	0.3	< 1		< 1		1	0.5	2	1.9	2	0.8
		0.5	< 1		< 1		1	0.6	2	2.2	3	1.0
		0.8	1	0.5	< 1		2	0.6	2	2.5	5	1.3
	0.5	0.3	1	0.7	< 1		1	0.6	3	3.4	2	0.7
		0.5	1	0.6	< 1		2	0.7	3	3.1	3	1.0
		0.8	1	0.7	< 1		3	1.0	3	2.9	6	1.6
	0.8	0.3	1	0.7	< 1		2	0.6	4	4.5	2	0.8
		0.5	1	0.8	< 1		3	0.8	4	4.9	4	1.5
		0.8	1	0.8	< 1		5	0.7	4	4.2	7	1.8
100	0.3	0.3	2	1.0	< 1		8	1.1	15	20.9	11	2.3
		0.5	3	1.2	< 1		13	1.8	14	14.0	19	3.1
		0.8	4	1.5	< 1		21	3.5	15	14.9	31	4.0
	0.5	0.3	4	1.5	1	0.5	13	2.1	20	20.9	16	2.7
		0.5	5	1.5	1	0.4	21	3.0	17	22.8	26	3.7
		0.8	6	1.6	1	0.5	34	5.2	19	24.2	44	5.4
	0.8	0.3	6	1.7	2	0.5	20	2.1	48	56.4	25	2.8
		0.5	7	2.0	2	0.6	34	3.5	33	39.3	41	3.3
		0.8	8	2.2	2	0.6	53	6.2	27	35.4	66	6.7
250	0.3	0.3	31	14.7	7	0.8	343	39.1	515	921.3	354	39.3
		0.5	37	14.4	7	0.7	513	74.9	542	728.8	615	74.4
		0.8	46	18.5	7	0.8	807	136.2	398	441.0	1021	203.4
	0.5	0.3	60	28.3	13	1.3	639	46.5	948	1217.1	587	73.2
		0.5	75	36.7	13	1.2	1073	103.7	777	1137.8	980	133.5
		0.8	82	33.6	13	1.5	1638	201.2	781	1111.2	1551	299.2
	0.8	0.3	95	46.1	22	2.0	1116	80.6	1763	2789.1	867	166.2
		0.5	128	68.3	22	2.1	1819	193.3	1697	2510.1	1511	216.0
		0.8	127	65.5	23	4.0	2859	289.0	1226	1903.6	2331	463.3

Table 4.12 – *Temps de calcul moyen en millisecondes (colonne de gauche) et écarts-types (colonne de droite) pour chaque classe de graphes, dépendant du nombre de nœuds, de la probabilité de relier deux nœuds dans l'instance et de la densité de terminaux.*

4.7 Synthèse des résultats

Dans ce chapitre, nous avons décrit un algorithme, FLAC, utilisant la technique d'ascension du dual pour trouver des arbres de faible densité. À l'aide de FLAC, nous avons décrit deux algorithmes d'approximation pour le problème de Steiner dont les propriétés sont données dans le tableau 4.13.

Algorithme	**Rapport d'approximation**	**Complexité temporelle**
$\text{Greedy}^{\triangleright}_{FLAC}$	$4\sqrt{2k}$	$O(n^3k^2)$
Greedy_{FLAC}	k	$O(nmk^2)$

Table 4.13 – *Algorithmes d'approximation décrits dans ce chapitre.*

Si, en théorie, $\text{Greedy}^{\triangleright}_{FLAC}$ a un meilleur d'approximation que Greedy_{FLAC}, il semble que Greedy_{FLAC} soit plus performant dans la pratique. En effet, ces deux algorithmes renvoient des solutions de poids proches, mais Greedy_{FLAC} est bien plus rapide que $\text{Greedy}^{\triangleright}_{FLAC}$. Toutefois, $\text{Greedy}^{\triangleright}_{FLAC}$ est une première étape visant à réduire le rapport d'approximation de Greedy_{FLAC} pour atteindre le rapport $O(k^{\epsilon})$ de l'algorithme de Charikar ou dépasser ce rapport est ainsi améliorer le meilleur rapport d'approximation connu pour le problème de Steiner.

Comparé à d'autres algorithmes couramment utilisés dans la pratique, comme par exemple l'algorithme des plus courts chemins, Greedy_{FLAC} renvoie rapidement des solutions de poids moins élevé.

Les résultats de ce chapitre seront publiés à la conférence COCOA 2014 [WW].

Chapitre 5.

Cas des graphes sans circuits structurés en paliers

On rappelle que la *profondeur* H d'une instance, définie en page 20 du chapitre 2, est la distance maximale, en nombre d'arcs, entre la racine et un terminal. On peut construire une $H\log(k)$-approximation pour cette instance [Eve07], en appliquant l'algorithme de Charikar avec un paramètre d'entrée égal à H [CCCD98].

Dans le cas d'une instance où tous les poids sont unitaires, k et H ont un fort impact sur le rapport d'approximation que l'on peut obtenir.

En effet, prenons l'exemple d'une instance de profondeur 2. Puisque les poids sont unitaires, le poids de toute solution réalisable est nécessairement compris entre k et $2k$. Le rapport d'approximation de tout algorithme ne peut donc dépasser 2.

De même, dans une instance de profondeur H, le poids d'une solution est compris entre k et $H \cdot k$. Le rapport d'un algorithme d'approximation ne peut donc dépasser H. Ce rapport est donc meilleur que celui que l'on connait dans le cas général.

Nous montrons dans ce chapitre, que dans le cas particulier des graphes sans circuits structurés en paliers, il existe un algorithme d'approximation polynomial dont le rapport est meilleur que H.

5.1 Description des instances étudiées

Toutes les instances que nous cherchons à résoudre dans ce chapitre ont des poids unitaires. On note $\mathbb{1}$ la fonction constante égale à 1. Nous étudions dans un premier temps des instances orientées sans circuits particulières. Nous reviendrons sur le cas général dans la section 5.5.

Dans la suite du chapitre, on s'intéresse aux instances dite *à paliers*.

Définition 5. Une instance $\mathcal{I} = (G = (V, A), r, X, \mathbb{1})$ est une instance à paliers si l'ensemble des nœuds est partitionné en $H + 1$ ensembles, appelés *paliers*, numérotés de H à 0 tels que tout arc ayant pour origine un nœud du palier i a pour destination un nœud du palier $i - 1$. La racine r est l'unique nœud du palier H et les terminaux sont les seuls nœuds du palier 0.

Dans la suite, on considère comme donnée une instance $\mathcal{I} = (G = (V, A), r, X, \mathbb{1})$ à paliers.

5.2 Résolution par le problème de couverture par ensembles

Le problème de couverture par ensembles demande de trouver une couverture de poids minimal d'un ensemble d'éléments X, appelé *univers*, à l'aide d'un sous-ensemble S des parties de X, pondérées par une fonctions de poids $\omega : S \to \mathbb{R}^+$.

La $\log(k)$-approximation gloutonne classique de la couverture par ensembles consiste à choisir l'ensemble couvrant le plus d'éléments, puis à temporairement supprimer ces éléments de l'univers, et à recommencer jusqu'à ce que celui-ci soit vide.

Utilisons cet algorithme pour résoudre notre instance $\mathcal{I}$. On nomme S_1 les noeuds du palier 1, prédécesseurs des terminaux. Le palier S_1 et X forment une instance du problème de couverture par ensemble, où un nœud v de S_1 couvre un terminal si ce dernier est successeur de v dans $\mathcal{I}$.

Soit $\mathcal{GL}_1$ l'algorithme suivant : on construit une couverture $c \subset S_1$ des terminaux à l'aide de la $\log(k)$-approximation pour le problème de couverture par ensembles. Puis on relie chaque nœud c à la racine par un chemin quelconque de H nœuds.

Théorème 5.2.1. *$\mathcal{GL}_1$ est une $((H-1)\log(k)+1)$-approximation.*

Démonstration. Soit T une solution réalisable renvoyée par $\mathcal{GL}_1$, T^* une solution optimale, et c et c^* les couvertures de X par des nœuds de S_1 respectivement incluses dans T et T^* [1]. Alors, comme l'illustre la figure 5.1, $\omega(T^*) \geq H-2+|c^*|+k$ et $\omega(T) \leq (H-1)|c|+k$. Soit c° une couverture optimale de X par des nœuds de S_1, alors $|c^\circ| \leq |c| \leq \log(k)|c^\circ| \leq \log(k)|c^*|$. On peut donc déduire les inégalités suivantes :

$$\begin{aligned}
\frac{\omega(T)}{\omega(T^*)} &\leq \frac{(H-1)|c|+k}{(H-2)+|c^*|+k} \\
&\leq \frac{(H-1)\log(k)|c^*|}{(H-2)+|c^*|+k} + \frac{k}{(H-2)+|c^*|+k} \\
&\leq (H-1)\log(k) + \frac{k}{H+k-1} \\
&\leq (H-1)\log(k)+1
\end{aligned}$$

□

5.3 Amélioration du rapport d'approximation

Si le rapport d'approximation de l'algorithme précédent est si élevé, c'est à cause du poids induit par la nécessité de relier la racine aux sommets de S_1 qui sont dans c. Pour réduire ce rapport d'approximation, il faut donc parvenir à réduire ce poids. En contrepartie, l'impact du poids des chemins situés entre les nœuds de S_1 et les terminaux, qui jusqu'à présent était négligeable, va augmenter. Il existe toutefois un compromis réduisant sensiblement le rapport.

1. c^* n'est pas nécessairement une couverture optimale des terminaux par les ensembles de S_1. En particulier, il est possible que $|c| < |c^*|$.

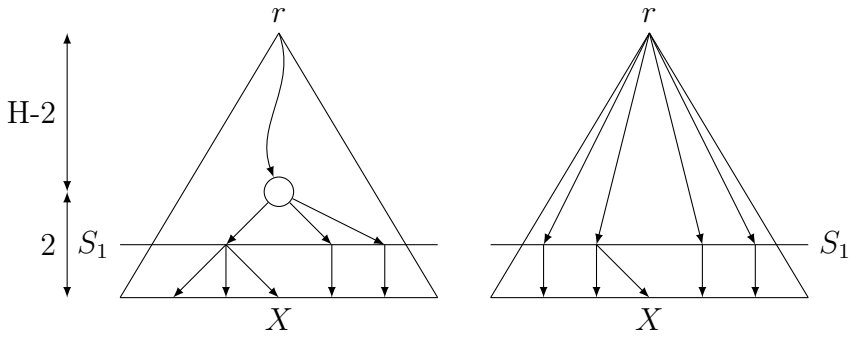


FIGURE 5.1 – *À gauche, la meilleure solution optimale qui puisse exister pour le problème : un chemin de longueur $H-2$ relie la racine à la couverture c^* des terminaux. À droite, la pire solution approchée qui puisse exister pour ce problème où il est impossible de relier la couverture c à la racine autrement que par des chemins disjoints.*

Soit S_h le palier h. On considère qu'un nœud de S_h couvre un terminal x s'il existe un chemin de cet ensemble vers le terminal x. Les ensembles S_h et les terminaux X définissent une nouvelle instance de couverture par ensemble.

Nous étendons $\mathcal{GL}_1$ en un algorithme $\mathcal{GL}_h$, décrit par l'algorithme 6.

Algorithme 6 Algorithme $\mathcal{GL}_h$

ENTRÉES : Une instance $\mathcal{I} = (G = (V, A), r, X, \mathbb{1})$ à $H+1$ paliers, et $h \in [\![1; H]\!]$.
SORTIES : T_h, une solution réalisable de $\mathcal{I}$

1: $T_h \leftarrow \emptyset$
2: Trouver, à l'aide de la $\log(k)$-approximation pour le problème de couverture par ensembles, une couverture $c_h \subset S_h$ couvrant X
3: **Pour** chaque nœud $v \in c_h$ **Faire**
4: Ajouter à T_h un chemin reliant la racine à v
5: **Pour** chaque terminal x **Faire**
6: $v \leftarrow$ un nœud de c_h couvrant x
7: Ajouter à T_h un chemin reliant v à x
8: **Renvoyer** T_h

On construit à partir de tous les algorithmes $\mathcal{GL}_h$ un algorithme $\mathcal{GL}$, décrit par l'algorithme 7, qui teste toutes les solutions renvoyées par ces algorithmes et renvoie la meilleure.

Algorithme 7 Algorithme $\mathcal{GL}$

ENTRÉES : Une instance $\mathcal{I} = (G = (V, A), r, X, \mathbb{1})$ à $H+1$ paliers.
SORTIES : T, une solution réalisable de $\mathcal{I}$

1: $T \leftarrow \emptyset$
2: **Pour** $1 \leq h \leq H$ **Faire**
3: $T_h \leftarrow \mathcal{GL}_h(\mathcal{I})$
4: **Si** $\omega(T) \geq \omega(T_h)$ **Alors**
5: $T \leftarrow T_h$
6: **Renvoyer** T

Nous démontrons maintenant que le rapport d'approximation de cet algorithme dépend de la valeur de H vis-à-vis d'une borne égale à $\frac{(k-1)\log(k)}{k-\log(k)}$.

Remarque 12. La valeur de cette borne n'est jamais plus élevée que $2\log(k)$.

Lemme 5.3.1. *Soit T^* une solution optimale de $\mathcal{I}$ et c_h^* l'ensemble des nœuds de $S_h \cap T^*$, pour $h \in [\![1; H]\!]$, alors $\omega(T^*) \geq (H - h - 1) + |c_h^*| \cdot h + k$.*

Démonstration. Soit $h \in [\![1; H]\!]$.

T^* est constitué successivement des nœuds de $c_H^* = \{r\}, c_{H-1}^*, \ldots, c_1^*$ et X. Chaque nœud de T^*, excepté r, a un et un seul arc entrant, sinon T^* ne serait pas une solution optimale.

$$\omega(T^*) = \sum_{i=1}^{H-1} |c_i^*| + k$$
$$\omega(T^*) = \sum_{i=h+1}^{H-1} |c_i^*| + \sum_{i=1}^{h} |c_i^*| + k$$

Chaque nœud non terminal de T^* a au moins un fils, sinon il pourrait être retiré de T^* et cette arborescence ne serait pas optimale. Donc $|c_H^*| = 1 \leq |c_{H-1}^*| \leq \cdots \leq |c_1^*| \leq |X| = k$.

$$\omega(T^*) \geq \sum_{i=h+1}^{H-1} 1 + \sum_{i=1}^{h} c_h^* + k$$
$$\omega(T^*) \geq (H - h - 1) + c_h^* \cdot h + k$$

□

La figure 5.2 illustre le lemme 5.3.1 et montre la meilleure solution optimale possible. Elle montre également la pire solution que peut renvoyer $\mathcal{GL}_h$.

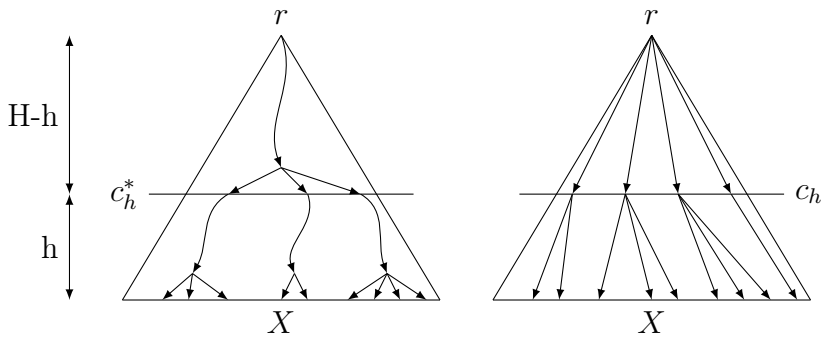


FIGURE 5.2 – *À gauche est représentée la meilleure solution optimale que l'on puisse espérer. À droite la pire solution possible renvoyée par $\mathcal{GL}_h$.*

Théorème 5.3.2. *Si* $\frac{(k-1)\log(k)}{k-\log(k)} < H$ *, alors* $\mathcal{GL}$ *est à la fois une* $O(\sqrt{H \cdot \log(k)})$*-approximation et une* $O(\sqrt{k \cdot \log(k)})$*-approximation.*

Démonstration. Soit $h \in [\![1; H]\!]$. Soit T_h une solution réalisable renvoyée par $\mathcal{GL}_h$, et c_h la couverture de X par des nœuds de S_h qui a permis de la construire. Soit c° une couverture optimale de X par des nœuds de S_h. Du plus, soit T^* une solution optimale et $c_h^* = S_h \cap T^*$. Puisque c_h est construit avec la $\log(k)$-approximation pour le problème de couverture par ensembles, on vérifie toujours $|c_h| \leq \log(k)|c^\circ|$. Puisque c_h^* est une couverture de X par des nœuds de S_h, $|c^\circ| \leq |c_h^*|$.

D'après le lemme 5.3.1 :

$$\omega(T^*) \geq (H - h - 1) + |c_h^*| \cdot h + k \tag{5.1}$$

Puisque $\mathcal{GL}_h$ relie arbitrairement tous les nœuds de c_h à la racine r et aux nœuds de X, il est possible, comme cela est montré en figure 5.2, que tous ces chemins soient disjoints.

$$\begin{aligned}\omega(T_h) &\leq (H-h)|c_h| + h \cdot k \\ &\leq (H-h)\log(k)|c^\circ| + h \cdot k \\ &\leq (H-h)\log(k)|c_h^*| + h \cdot k \end{aligned} \tag{5.2}$$

On déduit des inéquations (5.1) et (5.2) une borne supérieure du rapport $\frac{\omega(T^*)}{\omega(T_h)}$:

$$\begin{aligned}\frac{\omega(T^*)}{\omega(T_h)} &\leq \frac{(H-h)|c_h| + h \cdot k}{(H-h-1) + |c_h^*| \cdot h + k} \\ \frac{\omega(T^*)}{\omega(T_h)} &\leq \frac{H-h}{h}\log(k) + h\frac{k}{H+k-1}\end{aligned}$$

Soit $\tau(h)$ la fonction obtenue. Si on dérive τ pour rechercher un minimum h_0 de $\tau(h)$, on obtient :

$$\begin{aligned}\tau'(h_0) &= -\frac{H}{h_0^2}\log(k) + \frac{k}{H+k-1} = 0 \\ h_0 &= \sqrt{\frac{H(H+k-1)\log(k)}{k}}\end{aligned}$$

On vérifie l'égalité et les inégalités suivantes :

$$\begin{aligned}\tau(h_0) &= 2\sqrt{\frac{k \cdot H}{H+k-1}\log(k)} - \log(k) \\ \tau(h_0) &\leq 2\sqrt{H\log(k)} \\ \tau(h_0) &\leq 2\sqrt{k\log(k)}\end{aligned}$$

On vérifie bien que h_0 est un minimum, puisque la valeur de la fonction dérivée est strictement négative si $h < h_0$ et strictement positive si $h > h_0$.

Puisque h_0 est un réel et que notre algorithme ne parcourt que des valeurs entières de h, inspectons $\tau(\lfloor h_0 \rfloor)$. Cette valeur est plus grande que $\tau(h_0)$.

$$\begin{aligned}
\tau(\lfloor h_0 \rfloor) - \tau(h_0) &\leq \tau(h_0 - 1) - \tau(h_0) \\
&\leq (\frac{H - (h_0 - 1)}{h_0 - 1} - \frac{H - h_0}{h_0}) \log(k) - \frac{k}{H + k - 1} \\
&\leq (\frac{H - (h_0 - 1)}{h_0 - 1} - \frac{H - h_0}{h_0}) \log(k) - 1 \\
&\leq \frac{H \log(k)}{h_0(h_0 - 1)} - 1 \\
&\leq \frac{1}{1 - 1/h_0} \frac{H \log(k)}{h_0^2} - 1 \\
&\leq \frac{1}{1 - 1/h_0} \frac{k}{H + k - 1} - 1 \\
&\leq \frac{1}{1 - 1/h_0} - 1
\end{aligned}$$

On peut supposer sans perte de généralité que $\sqrt{H \log(k)} \geq 2$, donc $h_0 \geq 2$ et donc $\frac{1}{1-1/h_0} \leq 2$, ainsi $\tau(\lfloor h_0 \rfloor) - \tau(h_0) \leq 1$.
Par conséquent, les deux inégalités suivantes sont vérifiées :

$$\begin{aligned}
\min_{h \in [\![1;H]\!]} (\tau(h)) &\leq 1 + \tau(h_0) \leq 1 + 2\sqrt{H \log(k)} \\
\min_{h \in [\![1;H]\!]} (\tau(h)) &\leq 1 + 2\sqrt{k \log(k)}.
\end{aligned}$$

□

Théorème 5.3.3. *Si $H \leq \frac{(k-1)\log(k)}{k - \log(k)}$, alors $\mathcal{GL}$ est à la fois une $O(H)$-approximation et une $O(\log(k))$-approximation.*

Démonstration. Si H est plus petit que cette borne, alors h_0, décrit dans la preuve du théorème précédent, est plus grand que H.

Puisque $\tau(h)$ est une fonction décroissante pour $h < h_0$, alors $\tau(H)$ est minimum parmi les valeurs $\tau(h), h \leq H$.

$$\begin{aligned}
\frac{\omega(T^*)}{\omega(T)} &\leq \tau(H) \\
\frac{\omega(T^*)}{\omega(T)} &\leq \frac{H \cdot k}{H + k - 1} \leq H
\end{aligned}$$

Comme cela est précisé avec la remarque 12, $H \leq \frac{(k-1)\log(k)}{k - \log(k)} = O(\log(k))$, les rapports d'approximations sont vérifiés. □

5.4 Extension : couvrir avec deux paliers ou plus

On peut chercher à avoir non pas un mais deux paliers S_1 et S_2 de nœuds couvrant les terminaux. Cette fois-ci, on va parcourir tous les couples de paliers, et, pour chaque couple, trouver une couverture c_2 des terminaux par le second palier, et une couverture c_1 de c_2 par le premier palier. On ne peut plus utiliser l'algorithme glouton de couverture par ensembles pour construire c_1 et c_2, car ces deux couvertures doivent être construites en même temps. On est ici en présence non pas d'un problème de couverture par ensembles (soit un problème de Steiner dans un graphe à 3 paliers) mais d'un problème de Steiner dans un graphe à 4 paliers : la racine, S_1, S_2 et les terminaux.

Rechercher une bonne couverture revient donc à utiliser le meilleur algorithme d'approximation que l'on connaisse pour le problème de Steiner dans ce cas là. Or, pour un graphe de hauteur fixée égale à 2, on peut utiliser l'algorithme de Charikar, qui nous donne une $(2 - \log(k))$-approximation [CCCD98, Eve07]. Encore une fois, il faut considérer les trois couvertures suivantes : la couverture optimale dans le graphe à 4 paliers, celle d'une solution optimale T^* dans $\mathcal{I}$ et celle de notre algorithme d'approximation.

Une étude similaire à la précédente montre que cet algorithme a un rapport d'approximation $O(\sqrt[3]{H(2 * \log(k))^2})$ ou $O(\sqrt[3]{k(2 * \log(k))^2})$ à condition que H soit suffisamment grand.

De même, on peut démontrer qu'avec une couverture à l paliers on obtient un rapport $O(\sqrt[l+1]{H(l * \log(k))^l})$ ou $O(\sqrt[l+1]{k(l * \log(k))^l})$.

5.5 Adaptation au cas général

Soit $\mathcal{I} = (G = (V, A), r, X, \omega)$ une instance du problème de Steiner dont tous les poids des arcs sont unitaires. Nous avons déjà évoqué au chapitre 2, avec le théorème 2.1.1, une réduction transformant $\mathcal{I}$ en une instance à paliers. Mais les poids de cette instance ne sont pas tous unitaires, car certains arcs sont de poids nul.

La force de l'algorithme $\mathcal{GL}$ provient essentiellement des poids unitaires, car une solution utilisant de nombreux arcs coûte nécessairement cher. Nous ne connaissons pas aujourd'hui de moyen d'adapter simplement notre algorithme quand le graphe contient des arcs de poids nul sans détériorer le rapport d'approximation.

5.6 Synthèse des résultats

Dans ce chapitre, nous avons décrit un algorithme d'approximation polynomial, nommé $\mathcal{GL}$, pour le cas des instances à $H + 1$ paliers où tous les poids sont unitaires.

Le rapport d'approximation de cet algorithme dépend de la valeur de H vis-à-vis de k.

- Si $H \leq \frac{(k-1)\log(k)}{k-\log(k)}$, alors $\mathcal{GL}$ est à la fois une $O(H)$-approximation et une $O(\log(k))$-approximation.
- Si $H \geq \frac{(k-1)\log(k)}{k-\log(k)}$, alors $\mathcal{GL}$ est à la fois une $O(\sqrt{H\log(k)})$-approximation et une $O(\sqrt{k\log(k)})$-approximation.

Chapitre 6.

Approximation exponentielle pour le cas général

Ce chapitre présente une série d'algorithmes d'approximation exponentiel pour le problème de Steiner basé sur une approximation développée pour le problème de couverture par ensembles [CKW09]. L'idée est d'utiliser conjointement un algorithme de résolution exacte et un algorithme d'approximation polynomial. Un paramètre q, variant de 1 à k, gère la part de terminaux couverts par chaque algorithme et détermine le temps de calcul total.

Dans tout ce chapitre, on considère comme donnée une instance $\mathcal{I} = (G, r, X, \omega)$. On désignera par T^* une solution optimale de cette instance.

6.1 Séparation des terminaux

À partir d'un algorithme exact pour DST, il est possible de construire une 2-approximation exponentielle. Séparons l'ensemble des terminaux en deux ensembles disjoints X_1 et X_2. Soient T_1^* et T_2^* des solutions optimales couvrant respectivement X_1 et X_2. Étant toutes deux enracinées en r, l'union de ces solutions forme une solution réalisable pour $\mathcal{I}$. Puisque T^* couvre X_1 et X_2, $\omega(T_1^*) \leq \omega(T^*)$ et $\omega(T_2^*) \leq \omega(T^*)$. Donc $\omega(T_1^* \cup T_2^*) \leq \omega(T_1^*) + \omega(T_2^*) \leq 2 \cdot \omega(T^*)$. Cet algorithme prend un temps $t(|X_1|) + t(|X_2|)$ où $t(k)$ est le temps d'exécution d'un algorithme exact pour le problème de Steiner dans une instance possédant k terminaux.

De manière similaire, il est possible d'obtenir une q-approximation en temps $q \cdot T(\frac{k}{q})$ en partitionnant de façon équilibrée l'ensemble des terminaux en q sous-ensembles. Découper en k ensembles reviendrait à relier la racine aux terminaux avec des plus courts chemins. Il est donc important de trouver un compromis entre la séparation, limitant le temps de calcul, et le rapport d'approximation.

Remarque 13. Si la solution optimale est constituée de deux sous-arbres disjoints couvrant respectivement X_1 et X_2, alors partitionner les terminaux en ces deux sous-ensembles permettrait de calculer une solution optimale en un temps réduit. La manière dont on partitionne les terminaux a donc également son importance : si une bipartition aléatoire permet de garantir une 2-approximation, il est clairement possible d'obtenir mieux en choisissant un meilleur partitionnement.

Enfin, si un algorithme d'approximation (polynomial ou exponentiel) donne un rapport q_1 sur l'instance (G, r, X_1, ω) et q_2 sur l'instance (G, r, X_2, ω), alors l'union des deux solutions approchées donnera une $(q_1 + q_2)$-approximation sur (G, r, X, ω).

6.2 Approximation exponentielle pour la couverture par ensembles

Le problème de couverture par ensembles demande de trouver une couverture de poids minimal d'un ensemble de k éléments X, appelé *univers*, à l'aide d'un sous-ensemble S de parties de X, pondérées par une fonction de poids $\omega : S \to \mathbb{R}^+$.

Théorème 6.2.1 ([CKW09])**.** *Soit $q \in [\![1; k]\!]$ un entier, il existe un algorithme qui est une $(1 + \log(q))-$ approximation dont les complexités en temps et en espace sont $O^*(2^{\frac{k}{q}})$.*

Cet algorithme commence par couvrir une partie de l'univers avec un algorithme d'approximation glouton. Cet algorithme détermine la densité de chaque ensemble comme étant le rapport entre le poids de l'ensemble et le nombre d'éléments qu'il couvre. Il choisit ensuite l'ensemble de plus petite densité et recommence jusqu'à avoir couvert tout l'univers.

Dès que l'algorithme glouton a couvert au moins $k - \lceil \frac{k}{q} \rceil$ éléments, on couvre le reste avec un algorithme exact.

L'intérêt de l'algorithme glouton est qu'il va à la fois donner un bon rapport d'approximation, mais également effectuer une bonne découpe de X. En effet, il est conçu de sorte à construire petit à petit une solution approchée couvrant les éléments les plus intéressants (ceux que l'on peut couvrir en payant le moins).

Une fois un certain nombre d'éléments couverts, c'est l'algorithme exact qui prendra le relais, assurant une certaine qualité dans la manière de couvrir le reste de l'univers.

6.3 Généralisation au problème de Steiner

Si on transforme une instance $\mathcal{I}$ du problème de couverture par ensemble en instance $\mathcal{I}'$ de DST avec la réduction présentée en page 23 du chapitre 2, alors il est équivalent d'appliquer l'algorithme glouton du problème de couverture par ensembles à $\mathcal{I}$ et l'algorithme de Charikar à $\mathcal{I}'$: les deux renvoient le même résultat. On peut donc voir l'algorithme de Charikar comme une généralisation de l'algorithme glouton du problème de couverture par ensembles.

Il est donc naturel de penser que l'approximation exponentielle pour la couverture par ensembles est généralisable au problème de Steiner en utilisant l'algorithme de Charikar à la place de l'algorithme glouton.

L'idée consiste à utiliser l'algorithme de Charikar jusqu'à avoir couvert $k - \lceil \frac{k}{q} \rceil$ terminaux puis laisser le relais à un algorithme exact. Le temps de calcul ne doit pas excéder $O(t(\lceil \frac{k}{q} \rceil))$ où $t(k)$ est le temps de calcul d'un algorithme exact pour le problème de Steiner dans une instance possédant k terminaux.

Nous allons réutiliser dans ce chapitre les notions de densité et d'approximation partielle introduites dans le chapitre 2 respectivement par les définitions 2 et 3 en page 18.

La densité $d(T)$ d'une arborescence T enracinée en r est définie comme le rapport entre son poids $\omega(T)$ et le nombre de terminaux couverts $|X(T)| = k(T)$. Un algorithme $\mathcal{A}$ est

appelé $f(k)$-approximation partielle si, quelque soit l'instance auquel on l'applique, il renvoie une arborescence T dont la densité est au plus $f(k)$ fois celle d'une solution optimale.

En répétant $\mathcal{A}$ jusqu'à couvrir tous les terminaux, nous pouvons construire une approximation dont le rapport est donné par le lemme 2.2.2, décrit en page 19. Nous en donnons ci-après une variante qui nous servira dans la suite du chapitre.

Lemme 6.3.1 (Variante du lemme 2.2.2)**.** *Si $\mathcal{A}$ est une $f(k)$-approximation partielle, si $f(u)/u$ est décroissant, et si T_i est l'arbre renvoyé par la i^e répétition de l'algorithme $\mathcal{A}$, alors* $\sum_{j=1}^{i} \omega(T_j) \leq \int_{k-\sum_{j\leq i} k(T_j)}^{k} \frac{f(u)}{u} du \cdot \omega(T^*)$.

Démonstration. Prouvons ce lemme par récurrence sur i.

On suit la même démonstration que dans [CCCD98] pour le lemme 2.2.2. Puisque $\mathcal{A}$ est une $f(k)$-approximation partielle :

$$d(T_1) \leq f(k) \cdot d(T^*)$$
$$\omega(T_1) \leq \frac{k(T_1)}{k} f(k) \cdot \omega(T^*)$$

Par décroissance de $\frac{f(k)}{k}$, on vérifie :

$$\omega(T_1) \leq \int_{k-k(T_1)}^{k} \frac{f(u)}{u} du \cdot \omega(T^*)$$

Supposons la propriété prouvée pour i et montrons qu'elle est vraie pour $i+1$. Posons T_2^* une solution optimale de l'instance $(G, r, X \setminus \bigcup_{j\leq i} X(T_j), \omega)$. Puisque $\mathcal{A}$ est une $f(k)$-approximation partielle :

$$d(T_{i+1}) \leq f(k - \sum_{j\leq i} k(T_j)) \cdot d(T_2^*)$$
$$\omega(T_{i+1}) \leq \frac{k(T_{i+1})}{k - \sum_{j\leq i} k(T_j)} f(k - \sum_{j\leq i} k(T_j)) \cdot \omega(T_2^*)$$

Par décroissance de $\frac{f(k)}{k}$, on vérifie :

$$\omega(T_{i+1}) \leq \int_{k-\sum_{j\leq i+1} k(T_j)}^{k-\sum_{j\leq i} k(T_j)} \frac{f(u)}{u} du \cdot \omega(T_2^*)$$

Puisque T^* couvre également les terminaux de $X \setminus \bigcup_{j \leq i} X(T_j)$, alors $\omega(T_2^*) \leq \omega(T^*)$.

$$\omega(T_{i+1}) \leq \int_{k - \sum_{j \leq i+1} k(T_j)}^{k - \sum_{j \leq i} k(T_j)} \frac{f(u)}{u} du \cdot \omega(T^*)$$

Donc $\omega(T_1) + ... + \omega(T_{i+1}) \leq \int_{k - \sum_{j \leq i} k(T_{j+1})}^{k} \frac{f(u)}{u} du \cdot d(T^*)$.

Ceci conclut la récurrence et prouve le lemme. □

Afin de maîtriser le rapport d'approximation de notre algorithme, nous ne devons pas couvrir plus que le nombre de terminaux souhaité, soit $k - \lceil \frac{k}{q} \rceil$, puisque les autres seront couverts par l'algorithme exact.

On rappelle qu'on souhaite que le temps d'exécution de notre algorithme ne dépasse pas $t(\lceil \frac{k}{q} \rceil)$ où $t(k)$ est le temps d'exécution d'un algorithme exact pour le problème de Steiner dans une instance possédant k terminaux.

On répète l'algorithme $\mathcal{A}$ jusqu'à ce qu'il reste moins de $\lfloor \frac{k}{q} \rfloor$ terminaux à couvrir, c'est-à-dire qu'au moins $k - \lceil \frac{k}{q} \rceil$ terminaux soient couverts. Soit T le dernier arbre renvoyé par $\mathcal{A}$ et C_T l'union des autres arbres précédemment renvoyés par $\mathcal{A}$. On va montrer qu'il est possible de compléter C_T en un arbre C'_T couvrant exactement $k - \lceil \frac{k}{q} \rceil$ terminaux sans dépasser le temps de calcul escompté. Nous ne complétons pas C_T de la même manière selon que $k - k(C_T)$ soit plus grand ou plus petit que $2\lceil \frac{k}{q} \rceil$. L'algorithme 8 décrit cette approximation.

L'algorithme de Charikar est un algorithme paramétré par un entier l que l'on note, dans la suite du chapitre, CH_l. Lorsqu'il est utilisé pour couvrir tous les terminaux, il fournit, en temps $O(n^l k^{2l})$, une approximation de rapport $(l^3 k^{\frac{1}{l}})$ pour le problème de Steiner [CCCD98].

On rappelle que la *profondeur* H d'une instance, définie en page 20 du chapitre 2, est la distance maximale, en nombre d'arcs, entre la racine et un terminal. Nous distinguerons, dans ce chapitre, le cas où l'entier l choisi est la profondeur H de l'instance (le rapport d'approximation de l'algorithme CH_H est $H \log(k)$ [Eve07]) et le cas où l est plus petit que cette profondeur.

Nous utiliserons cet algorithme comme approximation partielle pour construire C_T et T.

Remarque 14. Il est possible d'utiliser tout algorithme connu fournissant une approximation partielle pour le problème de Steiner, comme l'algorithme des plus courts chemins ou les algorithmes Greedy_{FLAC} et $\mathrm{Greedy}^{\triangleright}_{FLAC}$ décrits dans le chapitre 4. Nous profiterions ainsi de leur efficacité dans la pratique et améliorerions, grâce à l'algorithme exact la qualité des solutions renvoyées.

L'algorithme 8 peut renvoyer deux solutions différentes selon que $k - k(C_T) \leq 2\lceil \frac{k}{q} \rceil$ ou $k - k(C_T) > 2\lceil \frac{k}{q} \rceil$. Nous allons donc démontrer le rapport d'approximation de l'algorithme dans chacun de ces cas, puis en déduire le rapport d'approximation dans le cas général.

Algorithme 8 Algorithme d'approximation exponentiel pour le problème de Steiner

Entrées : Une instance $\mathcal{I} = (G = (V, A), r, X, \omega)$, un paramètre $q \in [\![1; k]\!]$, une $f(k)$-approximation partielle $\mathcal{A}$ telle que $\frac{f(k)}{k}$ soit une fonction décroissante et un algorithme exact $\mathcal{E}$.

Sorties : une solution réalisable de $\mathcal{I}$.

1: $C_T \leftarrow \emptyset$
2: $T \leftarrow \emptyset$
3: **Tant que** $C_T \cup T$ ne couvre pas $k - \lceil \frac{k}{q} \rceil$ terminaux **Faire**
4: $\quad C_T \leftarrow C_T \cup T$
5: $\quad T \leftarrow \mathcal{A}(G, r, X \backslash X(C_T), \omega)$
6: **Si** $k - k(C_T) \leq 2\lceil \frac{k}{q} \rceil$ **Alors**
7: $\quad$ Couper arbitrairement $X \backslash X(C_T)$ en deux ensembles : X', de taille $k - k(C_T) - \lceil \frac{k}{q} \rceil$, et X'', de taille $\lceil \frac{k}{q} \rceil$
8: $\quad T' \leftarrow \mathcal{E}(G, r, X', \omega)$
9: $\quad T'' \leftarrow \mathcal{E}(G, r, X'', \omega)$
10: **Sinon**
11: $\quad T' \leftarrow$ un sous-arbre quelconque de T tel que $C_T \cup T'$ couvre exactement $k - \lceil \frac{k}{q} \rceil$ terminaux.
12: $\quad T'' \leftarrow \mathcal{E}(G, r, X \backslash X(C_T \cup T'), \omega)$
13: **Renvoyer** $C_T \cup T' \cup T''$

6.3.1 Cas 1 : $k - k(C_T) \leq 2\lceil \frac{k}{q} \rceil$

Dans ce cas, nous utilisons les lignes 8 et 9 de l'algorithme 8 pour couvrir les terminaux non couverts par C_T. Les arborescences T' et T" sont des solutions optimales qui couvrent chacune une moitié arbitraire des terminaux non couverts par C_T.

Calcul du rapport d'approximation.

Théorème 6.3.2. *Dans une instance de profondeur H, si on utilise l'algorithme CH_H pour construire C_T et si $k - k(C_T) \leq 2\lceil \frac{k}{q} \rceil$, le rapport d'approximation de l'algorithme 8 est $(H-1)\log(q) + 2$.*

Démonstration. Puisque nous utilisons l'algorithme CH_H dans une instance de profondeur H, alors $f(k) = H - 1$ [CCCD98, Eve07].

D'après le lemme 6.3.1 :

$$\omega(C_T) \leq \int_{k-k(C_T)}^{k} \frac{H-1}{u} du \cdot \omega(T^*)$$

$$\omega(C_T) \leq \int_{k-k(C_T)-k(T')}^{k} \frac{H-1}{u} du \cdot \omega(T^*)$$

D'après la ligne 8 de l'algorithme 8, $k(T') = k - k(C_T) - \lceil \frac{k}{q} \rceil$.

$$\omega(C_T) \leq \int_{\lceil \frac{k}{q} \rceil}^{k} \frac{H-1}{u} du \cdot \omega(T^*)$$

$$\omega(C_T) \leq (H-1)(\log(k) - \log(\lceil \frac{k}{q} \rceil)) \cdot \omega(T^*)$$

$$\omega(C_T) \leq (H-1)\log(q) \cdot \omega(T^*)$$

Sachant que T' et T'' sont des solutions optimales couvrant respectivement X' et X'', et que T^* est une solution couvrant ces terminaux, alors $\omega(T') \leq \omega(T^*)$ et $\omega(T'') \leq \omega(T^*)$. On en déduit donc que :

$$\omega(C_T \cup T' \cup T'') \leq ((H-1)\log(q) + 2) \cdot \omega(T^*)$$

□

Dans une instance de profondeur H supérieure au paramètre d'entrée l de l'algorithme CH_l, la valeur de $f(k)$ n'est pas la même, mais on peut démontrer de manière similaire que le rapport devient $l^3(k^{\frac{1}{l}} - (\frac{k}{q})^{\frac{1}{l}}) + 2$.

Complexités temporelles et spatiales. Nous utilisons partiellement l'algorithme CH_l, puis deux fois de suite un algorithme exact pour couvrir au plus $\lceil \frac{k}{q} \rceil$ terminaux. La complexité temporelle de l'algorithme 8 est donc $O(n^l k^{2l} + t(\lceil \frac{k}{q} \rceil))$ où $t(k)$ est le temps nécessaire à l'algorithme exact pour résoudre une instance avec k terminaux.

Sa complexité spatiale est $O(n + m + k + s(\lceil \frac{k}{q} \rceil)))$ où $s(k)$ est l'espace nécessaire à l'algorithme exact pour résoudre une instance avec k terminaux.

6.3.2 Cas 2 : $k - k(C_T) > 2\lceil \frac{k}{q} \rceil$

Dans ce cas, nous utilisons les lignes 11 et 12 de l'algorithme 8 pour couvrir les terminaux non couverts par C_T. Dans ce cas, T' est construit à partir de l'arborescence T, la dernière renvoyée par l'algorithme CH_l. T' est un sous-arbre arbitraire de T tel que $C_T \cup T'$ couvre exactement $k - \lceil \frac{k}{q} \rceil$ terminaux. Et T" est construit à l'aide d'un algorithme exact couvrait le reste des terminaux.

Nous allons dans un premier temps montrer que le rapport de l'approximation partielle qui a construit $C_T \cup T$ n'est dégradé que d'un facteur constant égal à 2 lorsque l'on remplace T par T'. Nous serons ensuite en mesure de démontrer le rapport d'approximation de l'algorithme 8 à l'aide d'une preuve similaire à celle du cas 1.

Propriété de T'.

Lemme 6.3.3. $d(T') \leq \frac{k(T)}{k(T')} d(T)$.

Démonstration. T' est un sous-arbre de T donc :

$$\begin{aligned}\omega(T') &\leq \omega(T)\\ \omega(T')\frac{k(T')}{k(T')} &\leq \omega(T)\frac{k(T)}{k(T)}\\ d(T')k(T') &\leq d(T)k(T)\end{aligned}$$

□

On sait que $\mathcal{A}$ est une $f(k)$-approximation partielle. Soit $\mathcal{A}'$ l'algorithme qui renvoie successivement tous les arbres de C_T puis T' à la place de T.

Lemme 6.3.4. *$\mathcal{A}'$ est une $2f(k)$-approximation partielle.*

Démonstration. Soit T_i le i^{e} arbre renvoyé par $\mathcal{A}'$ et X_i les terminaux couverts par T_i. Soit T_i^* une solution optimale couvrant tous les terminaux de $X\backslash\{X_1 \cup X_2 \cup \cdots \cup X_{i-1}\}$.

D'après la définition 3 de l'approximation partielle, nous devons montrer que $\frac{d(T_i)}{d(T_i^*)} \leq 2 \cdot f(X\backslash\{X_1 \cup X_2 \cup \cdots \cup X_{i-1}\})$.

Puisque $\mathcal{A}$ est une $f(k)$-approximation partielle, alors c'est aussi une $2f(k)$-approximation partielle. La propriété est donc démontrée pour tout arbre de C_T. Il ne reste donc plus qu'à la montrer pour l'arbre T'.

Soit T_p^* une solution optimale couvrant tous les terminaux de $X\backslash X(C_T)$ $\mathcal{A}$ étant une $f(k)$ approximation partielle, on vérifie :

$$d(T) \leq f(|X\backslash X(C_T)|)d(T_p^*)$$

D'après le lemme 6.3.3 :

$$d(T') \leq \frac{k(T)}{k(T')}f(|X\backslash X(C_T)|)d(T_p^*) \tag{6.1}$$

Puisque T contient entre $k - k(C_T) - \lceil\frac{k}{q}\rceil + 1$ et $k - k(C_T)$ terminaux, et que T' en contient exactement $k - k(C_T) - \lceil\frac{k}{q}\rceil$:

$$\frac{k(T)}{k(T')} \leq \frac{k - k(C_T)}{k - k(C_T) - \lceil\frac{k}{q}\rceil}$$

Or la fonction $\frac{x}{x-x_0}$ est décroissante sur l'intervalle $]x_0, +\infty[$. Puisque $k - k(C_T) > 2\lceil\frac{k}{q}\rceil$ alors :

$$\begin{aligned}\frac{k(T)}{k(T')} &\leq \frac{2\lceil\frac{k}{q}\rceil}{2\lceil\frac{k}{q}\rceil - \lceil\frac{k}{q}\rceil}\\ \frac{k(T)}{k(T')} &\leq 2\end{aligned}$$

Donc, d'après l'inégalité (6.1) :

$$d(T') \leq 2f(|X\backslash X(C_T)|)d(T_p^*)$$

Donc $\mathcal{A}'$ est une $2f(k)$-approximation partielle. □

Nous avons désormais tous les outils nécessaires au calcul du rapport d'approximation de l'algorithme 8 dans le cas 2.

Calcul du rapport d'approximation.

Théorème 6.3.5. *Dans une instance de profondeur H, si on utilise l'algorithme CH_H pour construire C_T et si $k - k(C_T) > 2\lceil\frac{k}{q}\rceil$, le rapport d'approximation de l'algorithme 8 est $2(H-1)\log(q)+1$.*

Démonstration. Puisque nous utilisons l'algorithme CH_H dans une instance de profondeur H, alors $f(k) = H - 1$ [CCCD98, Eve07].

D'après les lemmes 6.3.1 et 6.3.4 :

$$\omega(C_T \cup T') \leq \int_{k-k(C_T)-k(T')}^{k} 2\frac{H-1}{u}du \cdot \omega(T^*)$$

D'après la ligne 11, $k(T') = k - k(C_T) - \lceil\frac{k}{q}\rceil$.

$$\omega(C_T \cup T') \leq \int_{\lceil\frac{k}{q}\rceil}^{k} 2\frac{H-1}{u}du \cdot \omega(T^*)$$

$$\omega(C_T \cup T') \leq 2(H-1)(\log(k) - \log(\lceil\frac{k}{q}\rceil)) \cdot \omega(T^*)$$

$$\omega(C_T \cup T') \leq 2(H-1)\log(q) \cdot \omega(T^*)$$

Sachant que T'' est une solution optimale couvrant $X \backslash X(C_T \cup T')$, et que T^* est une solution couvrant ces terminaux, alors $\omega(T'') \leq \omega(T^*)$. On en déduit donc que :

$$\omega(C_T \cup T' \cup T'') \leq (2(H-1)\log(q)+1) \cdot \omega(T^*)$$

□

Dans une instance de profondeur H supérieure au paramètre d'entrée l de l'algorithme CH_l, la valeur de $f(k)$ n'est pas la même, mais on peut démontrer de manière similaire que le rapport devient $2l^3(k^{\frac{1}{l}} - (\frac{k}{q})^{\frac{1}{l}}) + 1$.

Complexités temporelle et spatiale Nous utilisons partiellement l'algorithme CH_l, puis, en temps linéaire, nous construisons T' à partir de T et, enfin, nous construisons T" avec un algorithme exact pour couvrir $\lceil\frac{k}{q}\rceil$ terminaux. La complexité temporelle de l'algorithme 8 est donc $O(n^l k^{2l} + n + t(\lceil\frac{k}{q}\rceil)) = O(n^l k^{2l} + t(\lceil\frac{k}{q}\rceil))$ où $t(k)$ est le temps nécessaire à un algorithme exact pour résoudre une instance avec k terminaux.

Sa complexité spatiale est $O(n + m + k + s(\lceil\frac{k}{q}\rceil)))$ où $s(k)$ est l'espace nécessaire à un algorithme exact pour résoudre une instance avec k terminaux.

Nous avons démontré dans cette sous-section et la précédente quels étaient les rapports d'approximation de l'algorithme 8 dans les cas où $k - k(C_T) \leq 2\lceil\frac{k}{q}\rceil$ et$k - k(C_T) > 2\lceil\frac{k}{q}\rceil$. Nous pouvons donc maintenant en déduire le rapport d'approximation dans le cas général.

6.3.3 Cas général

Rapport d'approximation.

Théorème 6.3.6. *Dans une instance de profondeur H, si on utilise l'algorithme CH_H pour construire C_T, le rapport d'approximation de l'algorithme 8 est $2(H-1)\log(q)+2$.*

Démonstration. D'après les théorèmes 6.3.2 et 6.3.5. □

Complexités temporelle et spatiale La complexité temporelle de l'algorithme 8 est $O(n^H k^{2H} + t(\lceil \frac{k}{q} \rceil))$ où $t(k)$ est le temps nécessaire à un algorithme exact pour résoudre une instance avec k terminaux. Si H est un paramètre constant, alors t l'emporte asymptotiquement. À l'inverse, il faut rester prudent si H est fonction de k.

Sa complexité spatiale est $O(n+m+k+s(\lceil \frac{k}{q} \rceil))$ où $s(k)$ est l'espace nécessaire à un algorithme exact pour résoudre une instance avec k terminaux.

Si le problème de Steiner est FPT vis-à-vis d'un paramètre, et si H est constant, cet algorithme est alors FPT pour ce paramètre (c'est le cas par exemple pour le paramètre $\frac{k}{q}$).

6.4 Synthèse des résultats

Dans ce chapitre, nous avons étendu une approximation exponentielle pour le problème de couverture par ensembles en une approximation exponentielle pour le problème de Steiner.

Cet algorithme utilise conjointement l'algorithme d'approximation glouton de Charikar et un algorithme exact. Un paramètre $q \in [\![1;k]\!]$ détermine la part de l'algorithme exact dans le processus (plus il est grand, moins il y a de terminaux couverts par cet algorithme). Un paramètre l détermine la précision de l'algorithme glouton.

Le rapport d'approximation de cet algorithme est $2(H-1)\log(q)+2$ quand l est la profondeur H de l'instance et $2l^3(k^{\frac{1}{l}} - (\frac{k}{q})^{\frac{1}{l}}) + 2$ si $l < H$. La complexité temporelle de cet algorithme est $O(n^l k^{2l} + t(\lceil \frac{k}{q} \rceil))$. Sa complexité spatiale est $O(n+m+k+s(\lceil \frac{k}{q} \rceil)))$.

À la place de l'algorithme de Charikar, il est tout à fait possible d'utiliser un autre algorithme d'approximation glouton, comme par exemple l'algorithme Greedy_{FLAC}, décrit dans le chapitre 4. On dispose ainsi d'un algorithme efficace en pratique dont le rapport d'approximation et le temps de calcul peut être paramétré par q.

Conclusion

Dans cette partie, nous avons exploré trois techniques d'approximation pour le problème de Steiner :

- l'algorithme FLAC trouve, en utilisant la technique d'ascension du dual, des arbres de faible densité et peut être employé pour construire des algorithmes d'approximation, Greedy$_{FLAC}$ et Greedy$^{\triangleright}_{FLAC}$, de rapports $O(k)$ et $O(\sqrt{k})$;
- un algorithme d'approximation de rapport $\sqrt{H \log(k)}$ dans les graphes structurés en paliers de poids unitaire et de profondeur H ;
- un algorithme d'approximation exponentiel, obtenu en utilisant conjointement un algorithme d'approximation glouton et un algorithme exact, et paramétré par la proportion de terminaux que doit couvrir l'algorithme exact.

Jusqu'à présent, le seul algorithme qui s'imposait, en pratique, pour la résolution du problème de Steiner était l'algorithme des plus courts chemins. Dans cette partie, nous avons introduit Greedy$_{FLAC}$ qui, étant donné ses performances et sa rapidité d'exécution peut se substituer à l'utilisation de l'algorithme des plus courts chemins.

Le meilleur rapport d'approximation connu quand la profondeur H de l'instance est constante est $H \log(k)$. Dans les graphes structurés en paliers, nos résultats montrent que, si la profondeur augmente au-delà de $2 \log(k)$, le problème peut être approché avec un rapport strictement inférieur. Ceci semble indiquer que, dans le cas général, le rapport d'approximation croît avec H, tant qu'il est inférieur à $\log(k)$ puis décroît, ce qui montrerait que le meilleur rapport d'approximation serait $O(\log^2(k))$.

Enfin, nous avons introduit pour la première fois une approximation exponentielle pour le problème. Sa qualité dépend de celle de l'algorithme d'approximation glouton qu'elle utilise. Ceci ouvre des opportunités pour des améliorations ultérieures.

Il existe une quatrième technique que nous n'avons pas encore employé : contraindre artificiellement le problème. Nous limitons ainsi le nombre de solutions réalisables et pouvons trouver une solution approchée ou optimale parmi les solutions restantes. Si la contrainte est assez forte pour permettre de trouver une bonne solution en temps polynomial, et assez lâche pour que l'ensemble des solutions restantes contienne au moins une bonne approximation de la solution optimale, alors nous pouvons construire un nouvel algorithme d'approximation pour le problème de Steiner.

La troisième partie étudie une contrainte différente : limiter le nombre de nœuds de branchement des solutions réalisables.

Troisième partie

Problèmes de Steiner à branchements contraints

Cette partie introduit et étudie deux problèmes de Steiner possédant chacun une contrainte supplémentaire sur les nœuds de branchement des solutions réalisables. L'objectif est de restreindre l'ensemble des solutions réalisables de l'instance d'origine de sorte à pouvoir en extraire une solution optimale contrainte, et de déterminer si cette solution optimale est une bonne approximation de la solution optimale de l'instance sans contrainte. Le premier chapitre présente les contraintes et les trois suivants étudient les résultats d'approximation ou de résolution exact de ces problèmes.

Chapitre 7.

Présentation des problèmes étudiés

Afin de réduire le nombre de solutions réalisables, nous exploitons une contrainte de branchement dans deux situations différentes. Nous donnons dans ce chapitre les définitions des deux problèmes contraints, puis nous expliquons dans la dernière section l'origine des contraintes dans le cadre des réseaux tout-optiques.

Dans chacune des versions du problème de Steiner qui suivent, nous limitons le nombre de *nœuds de branchement* dans la solution. Un nœud de branchement dans une arborescence est un nœud qui a au moins 2 fils.

7.1 Limitation du nombre de nœuds de branchement

Nous décrivons dans cette section le problème de l'arborescence de Steiner avec un nombre limité de nœuds de branchement [1].

Problème : Problème de l'arborescence de Steiner avec un nombre limité de nœuds de branchement (DSTLB)

Instance :

- Un graphe orienté $G = (V, A)$.
- Un nœud *racine* $r \in V$.
- Un ensemble de k nœuds terminaux $X \subset V$.
- Une fonction de poids $\omega : A \to \mathbb{R}^+$.
- Un entier $p \in [\![0; k-1]\!]$.

Solution réalisable : Une arborescence T enracinée en r, couvrant tous les terminaux et ne possédant pas plus de p nœuds de branchement.

Optimisation : Minimiser le poids $\omega(T) = \sum_{a \in T} \omega(a)$.

---

1. *Directed Steiner Tree with Limited number of Branching nodes*, en anglais.

La contrainte $p \leq k-1$ rappelle qu'une solution optimale ne peut posséder plus de $k-1$ nœuds de branchement, sans quoi ce ne serait pas une arborescence. Un exemple est donné en figure 7.1.

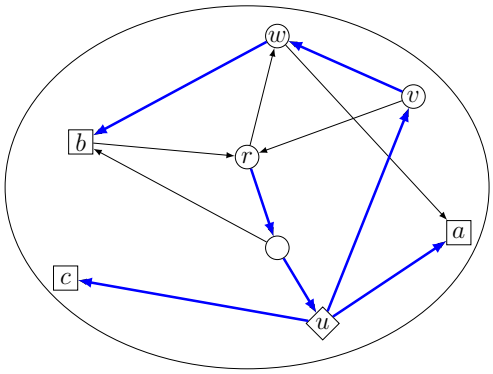


FIGURE 7.1 – *Exemple d'arborescence reliant la racine r à 3 terminaux a, b et c, avec $p = 1$ nœud de branchement autorisé (représenté ici par un losange) lorsque les poids sont tous égaux à 1.*

Remarque 15. La racine et les terminaux comptent eux aussi pour un nœud de branchement s'ils ont plusieurs fils. Par exemple, dans la figure 7.1, l'arc (r, w) ne peut être employé à la place du chemin $(u, v), (v, w)$ car alors on utiliserait deux nœuds de branchement au lieu d'un seul.

Cette version du problème de Steiner sera étudiée dans les chapitres 8 et 10. Nous le verrons, DSTLB est un problème difficile car prouver l'existence d'une solution réalisable d'une instance dans le cas général est un problème NP-complet même si les paramètres k et p sont fixés. Le chapitre 8 cherche donc les différents cas où il est possible de le résoudre ou de l'approcher. En particulier, il étudie en parallèle la version non orientée de DSTLB : le problème de l'arbre de Steiner avec un nombre limité de nœuds de branchement [2]. Dans cette version un nœud de branchement d'un arbre ou d'un graphe est un nœud qui possède 3 voisins.

Problème : Problème de l'arbre de Steiner avec un nombre limité de nœuds de branchement (USTLB)

Instance :

- Un graphe non orienté $G = (V, E)$.
- Un ensemble de k nœuds terminaux $X \subset V$.
- Une fonction de poids $\omega : E \to \mathbb{R}^+$.
- Un entier $p \in [\![0; k-2]\!]$.

Solution réalisable : Un arbre T couvrant tous les terminaux et ne possédant pas plus de p nœuds de branchements.

Optimisation : Minimiser le poids $\omega(T) = \sum_{e \in T} \omega(e)$.

---

2. *Undirected Steiner Tree with Limited number of Branching nodes*, en anglais

De façon analogue au cas orienté, la contrainte $p \leq k-2$ rappelle qu'une solution optimale ne peut posséder plus de $k-2$ nœuds de branchement, sans quoi ce ne serait pas un arbre.

7.2 Limitation du nombre de nœuds diffusants

Du fait de la difficulté du problème DSTLB, nous nous sommes tournés vers un problème plus simple, garantissant la présence d'au moins une solution réalisable.

L'algorithme de Charikar [CCCD98] recherche des solutions réalisables de hauteur fixée par un paramètre H. Cependant, une telle solution n'existe pas toujours, si chaque terminal est séparé de la racine par un chemin d'au moins $H+1$ arcs. Pour contourner ce problème, les algorithmes qui utilisent cette technique travaillent dans l'instance des plus courts chemins $\mathcal{I}^{\triangleright}$, vue dans le chapitre 2 en page 16. Rappelons que le théorème 2.1.2, donné en page 16, montre que cette instance a les mêmes propriétés d'approximation que $\mathcal{I}$.

Puisque l'instance $\mathcal{I}^{\triangleright}$ est constituée d'un graphe orienté complet, il est maintenant envisageable de trouver tout type de solution réalisable, y compris des arborescences de hauteur fixée et des arborescences dont le nombre de nœuds de branchement est fixé.

Nous supposons dans un premier temps que tous les terminaux sont des feuilles dans G et donc que tout arc sortant d'un terminal dans $\mathcal{I}^{\triangleright}$ a un poids infini. Si ce n'est pas le cas pour un terminal, on duplique le terminal, puis on relie l'original et la copie par un arc de poids nul, enfin on retire de X l'original et on y insère la copie. Dans le cas du problème de Steiner classique, cela ne change pas les poids des solutions optimales ou réalisables.

Nous définissons le problème de l'arborescence de Steiner avec un nombre limité de nœuds diffusants[3] ainsi :

Problème : Problème de l'arborescence de Steiner avec un nombre limité de nœuds diffusants (DSTLD)

Instance :

- Un graphe orienté $G = (V, A)$.
- Un nœud *racine* $r \in V$.
- Un ensemble de k nœuds *terminaux* $X \subset V$, tous feuilles de G.
- Une fonction de poids $\omega : A \to \mathbb{R}^+$.
- Un entier $d \in [\![1; k-1]\!]$.

Solution réalisable : Une arborescence T de $G^{\triangleright}$ enracinée en r, couvrant tous les terminaux et ne possédant pas plus de d nœuds de branchement.

Optimisation : Minimiser le poids $\omega^{\triangleright}(T) = \sum_{a \in T} \omega^{\triangleright}(a)$.

Nous reviendrons sur la notion de nœuds diffusants dans la section 7.3.

Le problème DSTLD est étudié dans les chapitre 9 et 10.

Nous nous sommes restreints aux instances où les arcs sortant des terminaux sont tous de poids infini. C'est pourquoi la valeur du paramètre d est au moins égale à 1. Si nous relâchons

3. *Directed Steiner Tree with Limited number of Diffusing nodes*, en anglais

cette hypothèse, alors le nombre de solutions réalisables augmente et le problème est plus difficile à résoudre. En particulier il n'est pas certain qu'il existe un algorithme FPT vis-à-vis du paramètre k si l'hypothèse est relâchée. À l'inverse, nous verrons dans le chapitre 10 qu'il existe un algorithme FPT en k si cette hypothèse est respectée.

7.3 Origine des contraintes

Dans les réseaux filaires utilisés pour les petites infrastructures (entreprises, campus, ...), il est courant de disposer de routeurs ou switchs capables d'effectuer une opération de *multicast* ou de *broadcast* : stocker une donnée entrante, et la retransmettre à un ou plusieurs destinataires. L'information transite ensuite par câble cuivré ou par fibre optique. Ce sont des architectures hybrides qui mélangent électronique et optique pour transmettre le signal.

Afin d'augmenter la qualité de service, il est possible de remplacer tout le matériel électronique par du matériel optique ; ce dernier accroît grandement la vitesse de diffusion de l'information dans le réseau. Ces réseaux, dits tout-optiques, cherchent alors à faire transiter l'information d'un expéditeur vers un destinataire en n'utilisant que du matériel optique, qu'il corresponde à une connexion ou à un routeur.

Nous nous intéressons au problème de satisfaire une requête multicast en minimisant la bande passante utilisée. Nous allons pointer dans un premier temps les différences entre une diffusion dans un réseau tout-optique et un réseau plus conventionnel, puis définir notre problématique.

Deux problèmes surviennent lorsque l'on veut transmettre les données par voie exclusivement lumineuse. Le premier est qu'un routeur optique standard ne peut effectuer d'opération ni de multicast ni de broadcast. Il se contente de recevoir un flux optique, et utilise le mécanisme MPLS [mpl] pour le dévier vers un seul destinataire, selon sa table de routage, en utilisant des informations codées directement sous forme optique (par exemple la longueur d'onde d'un signal secondaire émis avec le signal principal). Pour pallier ce manque, deux solutions existent.

- Un routeur dit *opto-électronique* est capable d'encoder temporairement le flux lumineux sous forme électronique pour le traiter et le copier autant de fois que nécessaire. Il introduit cependant un retard dans la transmission, et une consommation d'énergie accrue.
- Un routeur dit *séparateur* (ou *splitter*) est capable, par un jeu de miroirs, de séparer le flux lumineux en plusieurs copies pour plusieurs destinataires. Il introduit une atténuation du signal qu'il faudra réamplifier et nécessite une maintenance coûteuse.

Ces routeurs sont appelés *nœuds diffusants*. On prendra bien soin par la suite de faire la différence entre un nœud diffusant capable, à partir d'une seule donnée, de la retransmettre plusieurs fois, et un nœud non diffusant qui doit recevoir cette donnée autant de fois qu'il souhaite la renvoyer. La figure 7.2 illustre ces deux nœuds.

Le second problème des réseaux tout-optiques est qu'il faut prendre garde aux interférences qui peuvent survenir quand deux flux lumineux de longueurs d'onde proches ou égales sont à proximité. Afin de bien différencier les flux et éviter toute interférence, il est préférable d'utiliser des longueurs d'onde différentes quand deux flux utilisent la même connexion ou le même nœud. Mais tous les nœuds ou arcs ne peuvent traiter qu'un nombre limité de longueurs d'onde distinctes au même instant.

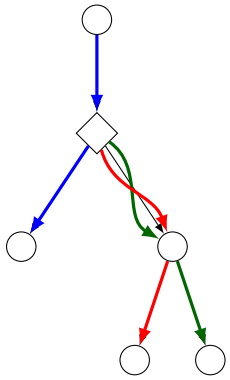


FIGURE 7.2 – *Exemple d'un nœud diffusant (représenté par un losange) et d'un nœud non diffusant. Le nœud diffusant reçoit la donnée une seule fois et la retransmet trois fois. Le nœud non diffusant doit la transmettre deux fois et donc la reçoit deux fois.*

Concernant les arcs, on mesurera le nombre de longueurs d'onde distinctes qui transitent dans chaque arc. C'est exactement équivalent à la charge de cet arc.

Concernant les nœuds, on mesurera aussi la charge des nœuds. De plus, on va les partager en deux catégories selon qu'ils possèdent ou non la capacité de convertir les longueurs d'onde. Un nœud a la capacité de *convertir les longueurs d'onde*, si, recevant en entrée une donnée encodée sous forme lumineuse avec une longueur d'onde donnée, il est capable de renvoyer en sortie la même donnée encodée sous forme lumineuse avec une autre longueur d'onde.

On notera en particulier le fait qu'un nœud diffusant qui ne possède pas cette propriété ne peut renvoyer un paquet deux fois sur le même arc s'il ne le reçoit qu'une fois. En effet, pour le transmettre deux fois sur le même arc, il doit le transmettre à l'aide de deux longueurs d'onde distinctes. S'il n'a pas la capacité de convertir les longueurs d'onde, alors la donnée qu'il reçoit en entrée reste encodée en sortie sur la même longueur d'onde.

On cherche à dimensionner le réseau : déterminer le poids des connexions, déterminer le nombre de nœuds diffusants et déterminer leur placement dans le réseau. Généralement, l'étude se fait dans un réseau non orienté alors que le placement des nœuds diffusant dans le réseau a déjà été décidé. La référence la plus ancienne au problème de diffusion multicast dans un réseau tout-optique peut être trouvée dans [MZQ98].

Si aucun nœud n'est diffusant ni ne possède la capacité de convertir les longueurs d'ondes, alors une solution possible est de calculer un arbre de Steiner à l'aide d'une r approximation polynomiale. On parcours ensuite cet arbre en suivant un cycle eulérien (en considérant que chaque arête est en réalité un couple de deux arcs symétriques). La solution renvoyée est une $2r$-approximation, dont la qualité peut être localement améliorée en insérant des nœuds diffusants [YDA03].

Dans le cas où les nœuds peuvent convertir les longueurs d'ondes et où les nœuds diffusants sont déjà placés, des heuristiques plus récentes être trouvées dans [RTBW09] pour répondre à une requête multicast.

Un autre résultat récent montre que si, pour une unique requête, on connait l'arbre de multicast par lequel se fera la diffusion, alors il est polynomial de placer les nœuds diffusants dans cet arbre pour optimiser le coût de cette diffusion [RCT$^+$12]. Les nœuds diffusants ont la capacité de convertir les longueurs d'onde. Les algorithmes décrits dans le chapitre 10 de cette partie s'inspirent de ces résultats.

Le problème de placement de nœuds diffusant dans un réseau tout-optique dans le but de satisfaire une ou plusieurs requêtes est introduit dans [AD00].

Nous expliquons dans la suite du chapitre pourquoi, dans le cas orienté, les problèmes DSTLB et DSTLD sont adaptés à la problématique de dimensionnement d'un réseau tout-optique.

7.3.1 DSTLB : Modélisation du problème avec une seule longueur d'onde

Dans cette version, seule une longueur d'onde peut être utilisée pour diffuser la donnée de la racine vers les terminaux. Soit un réseau tout-optique, ne possédant pas encore de nœud diffusant, dans lequel une requête de diffusion multicast doit être satisfaite. Nous avons à notre disposition un nombre fixé de nœuds diffusants, seuls une partie des nœuds du réseau pourront être remplacés par des nœuds diffusants. Où doit-on disposer ces nœuds diffusants pour pouvoir ensuite effectuer la diffusion de la donnée pour un poids optimal ?

On ne dispose que d'une seule longueur d'onde. Cette contrainte empêche tout arc ou tout nœud de servir au transit du paquet deux fois. Dans ce cas, les nœuds de branchements et les nœuds diffusants de la solution sont confondus. Une solution réalisable d'une instance de DSTLB permet de placer immédiatement les nœuds diffusants dans le réseau : il s'agit des nœuds de branchement de notre solution.

Les avantages d'une telle solution sont premièrement la possibilité de satisfaire dans le réseau plusieurs requêtes multicast en même temps, où chacune se voit attribuée une longueur d'onde distincte, et deuxièmement la possibilité de disposer d'un réseau où aucun nœud n'est capable de convertir les longueurs d'onde.

Le cas particulier de la racine. La racine dans DSTLB ne se distingue pas des autres nœuds : d'après la définition du problème, si elle a plusieurs successeurs dans la solution proposée, elle compte pour un des p nœuds diffusants que l'on peut placer dans le réseau. Cette décision peut paraître étonnante du point de vue de l'application. En effet, la racine n'a pas besoin de recevoir une donnée pour la transmettre à ses successeurs et peut donc transmettre son paquet à plusieurs successeurs sans enfreindre la contrainte d'unicité de la longueur d'onde. C'est la seule exception où un nœud peut être à la fois non diffusant et nœud de branchement, et on pourrait donc définir le problème en recherchant un arbre contenant au plus p nœuds de branchement *autres que la racine*.

Cependant introduire cette information dans le modèle ne changerait pas les résultats théoriques que nous présenterons dans le chapitre 8 et nous obligerait à tenir compte de cette exception dans toutes les démonstrations. Nous considérons donc par la suite que, si la racine a plusieurs successeurs, alors il s'agit d'un nœud diffusant.

7.3.2 DSTLD : Modélisation du problème sans contrainte sur les longueurs d'onde

Cette variante du problème de Steiner ne se soucie pas des contraintes de longueur d'onde dans le réseau. Toute connexion est en mesure de traiter autant de flux lumineux en même temps qu'elle le souhaite. De plus, nous supposons que tous les nœuds de notre réseau ont la

capacité de convertir les longueurs d'onde. À notre connaissance, les routeurs séparateurs (splitters) n'ont pas cette capacité, nous ne considérerons donc que des nœuds diffusants opto-électroniques. Ainsi un nœud diffusant recevant une donnée peut la retransmettre à un de ses successeurs deux fois, encodée avec deux longueurs d'onde distinctes.

Soit un réseau tout-optique, ne possédant pas encore de nœud diffusant, dans lequel une requête de diffusion multicast doit être satisfaite. Nous avons à notre disposition un nombre fixé de nœuds diffusants, seuls une partie des nœuds du réseau pourront être remplacés par des nœuds diffusants. Où doit-on disposer ces nœuds diffusants pour pouvoir ensuite effectuer la diffusion de la donnée pour un poids optimal ?

Lien entre DSTLD et la diffusion dans le réseau. Soit $\mathcal{I} = (G, r, X, \omega, d)$ une instance de DSTLD ; une fois calculée une solution réalisable T de l'instance des plus courts chemins $\mathcal{I}^{\triangleright}$, elle ne représente pas immédiatement la solution recherchée dans le réseau. Mais il est possible de définir un placement des nœuds diffusants et une diffusion multicast dans le graphe d'origine G à partir de T.

- Chaque nœud de branchement de T est un nœud diffusant dans G ;
- pour chaque arc (u, v) de T, choisir un plus court chemin dans G reliant u à v, et augmenter la charge de ce chemin de 1.

Ainsi pour envoyer la donnée de r vers les terminaux, il suffit de suivre pour chaque arc de T le plus court chemin correspondant dans G. Le poids de la diffusion et celui de la solution réalisable sont identiques.

La solution optimale d'une instance de DSTLD est-elle réellement une solution optimale du problème pratique de placement de nœuds diffusants et de diffusion multicast dans un réseau tout-optique ? Le fait de n'utiliser que des plus courts chemins est-il une bonne solution ?

Supposons que l'on connaisse un placement optimal et une diffusion optimale dans le réseau. Soit un chemin P utilisé par la donnée pour rejoindre, depuis la racine, un nœud diffusant ou un terminal, de sorte que P ne contienne aucun nœud diffusant (hormis éventuellement aux extrémités). Alors ce chemin est nécessairement un plus court chemin. En effet, si ce n'est pas le cas, il existe un chemin P' de poids plus petit reliant la racine à ce même nœud. Et si la donnée emprunte ce chemin, réduisant ainsi la charge des arcs de P de 1 et augmentant celle des arcs de P' de 0 ou 1 (cette charge n'augmente pas si l'arc est déjà utilisé et mène à un nœud diffusant) : le poids total de cette nouvelle solution est moins élevé, contredisant ainsi l'optimalité de notre diffusion.

De même, tout chemin reliant deux nœuds diffusants, ou un nœud diffusant et un terminal, qui ne contient que des nœuds non diffusants sauf aux extrémités, est un plus court chemin.

Revenons au graphe des plus courts chemins $G^{\triangleright}$. Pour chacun de ces plus courts chemins, sélectionnons l'arc dans $G^{\triangleright}$ reliant ses extrémités. Cette union d'arcs forme nécessairement un sous-graphe de $G^{\triangleright}$ de poids égal à celui de la diffusion, enraciné en r, contenant au plus d nœuds de branchement et couvrant tous les terminaux. Et si cette union n'est pas une arborescence, alors elle contient des cycles que l'on peut supprimer sans altérer le poids de la solution, car la diffusion à l'origine était de poids optimal.

Cas particulier de la racine. De même que pour DSTLB, la racine fait office d'exception. En effet, même si elle est sélectionnée comme nœud de branchement dans $G^{\triangleright}$, il n'y a pas d'intérêt à la désigner comme diffusante dans le réseau.

Cependant introduire cette information dans le modèle ne changerait pas les résultats théoriques que nous présenterons dans le chapitre 9 et nous obligerait à tenir compte de cette exception dans toutes les démonstrations. Nous considérons donc par la suite que, si la racine a plusieurs successeurs dans $G^{\triangleright}$, alors il s'agit d'un nœud diffusant.

Restriction aux instances où les terminaux sont des feuilles. Dans la section 7.2, nous avons précisé que nous nous restreignons aux instances de DSTLD pour lesquelles les terminaux sont des feuilles. Et, si cette hypothèse n'était pas respectée, alors on duplique chaque terminal, on relie l'original et la copie par un arc de poids nul, et on remplace l'original par la copie dans l'ensemble des terminaux X.

Cette hypothèse est vérifiée dans la pratique. Si un terminal doit à la fois recevoir une donnée et la retransmettre, il doit soit la recevoir deux fois, soit être un nœud diffusant (ou du moins avoir une technologie similaire à celle des nœuds diffusants). Si on duplique un terminal pour en faire une feuille, et si le terminal doit à la fois recevoir une donnée et la retransmettre, alors le nœud original doit soit la recevoir deux fois, soit être diffusant. En se restreignant aux instances qui vérifient cette hypothèse, on simplifie donc la description du problème.

Chapitre 8.

DST avec un nombre limité de nœuds de branchement

Nous rappelons ci-après la définition des problèmes de l'arbre et de l'arborescence de Steiner avec un nombre limité de nœuds de branchement [1], qui se différencient respectivement de UST et DST par l'ajout d'un paramètre p : le nombre de nœuds diffusants que l'on peut placer dans le graphe.

Problème : Problème de l'arborescence de Steiner avec un nombre limité de nœuds de branchement (DSTLB)

Instance :

- Un graphe orienté $G = (V, A)$.
- Un nœud *racine* $r \in V$.
- Un ensemble de k nœuds terminaux $X \subset V$.
- Une fonction de poids $\omega : A \to \mathbb{R}^+$.
- Un entier $p \in [\![0; k-1]\!]$.

Solution réalisable : Une arborescence T enracinée en r, couvrant tous les terminaux et ne possédant pas plus de p nœuds de branchement.

Optimisation : Minimiser le poids $\omega(T) = \sum_{a \in T} \omega(a)$.

---

1. *Undirected and Directed Steiner Tree with Limited number of Branching nodes*, en anglais

Problème : Problème de l'arbre de Steiner avec un nombre limité de nœuds de branchement (USTLB)

Instance :
- Un graphe non orienté $G = (V, E)$.
- Un ensemble de k nœuds terminaux $X \subset V$.
- Une fonction de poids $\omega : E \to \mathbb{R}^+$.
- Un entier $p \in [\![0; k-2]\!]$.

Solution réalisable : Un arbre T couvrant tous les terminaux et ne possédant pas plus de p nœuds de branchement.

Optimisation : Minimiser le poids $\omega(T) = \sum_{e \in T} \omega(e)$.

Nous allons nous focaliser dans ce chapitre sur les résultats de complexité paramétrée et d'approximation paramétrée pour ce problème vis-à-vis des paramètres k et/ou p. Le chapitre 10 vient compléter ces résultats par l'adjonction d'un troisième paramètre, n_s, le nombre de nœuds non terminaux de la solution optimale, à l'aide d'une technique d'énumération adaptable aux problème DST, DSTLD, DSTLB et toutes leurs versions non orientées.

Nous verrons dès la première section de ce chapitre que DSTLB est un problème difficile dans le cas général, même s'il est paramétré par k et p. C'est pourquoi, dans ce chapitre, nous étudions un panel de cas particuliers pour DSTLB, mais aussi pour USTLB, la version non orientée de DSTLB. Ces cas particuliers ne sont pas choisis au hasard. Le premier résultat de NP-complétude de ce chapitre provient d'une réduction depuis le problème de recherche de chemins disjoints entre des couples distincts de nœuds. Les autres cas sont ceux où cette réduction n'est plus valable : graphe non orienté, graphe planaire, graphe sans circuits.

Le tableau 8.1 résume les résultats de complexité paramétré vis-à-vis des deux paramètres k, p étudiés dans ce chapitre et vis-à-vis du paramètre n_s étudié dans le chapitre 10. Ces résultats se découpent en quatre parties qui constituent les deux sections suivantes de ce chapitre, et deux des sections du chapitre 10. Les autres résultats de complexité paramétrée se déduisent de ceux du tableaux, ou bien ne sont pas encore connus.

8.1 Cas général

Dans cette section, nous donnons deux preuves, démontrant la NP-complétude et l'inapproximabilité forte du problème DSTLB paramétré par k et p, ainsi que des problèmes DSTLB et USTLB paramétrés par p dans le cas d'un graphe planaire.

Dans la suite de cette section, nous désignerons par (k, p)-*instance* une instance contenant au plus k terminaux, n'autorisant pas plus de p nœuds de branchement dans la solution. De même, nous désignerons par $(k, p, \mathbb{1})$-*instance* une (k, p)-instance dont tous les poids sont unitaires.

Plan	Problème	k	p	n_s	Conditions	Trouver une solution réalisable	Minimisation
Section 8.1	DSTLB	×	×		$p \leq k-2$	$\notin$ XP	n^{ε}-Inappr.
	DSTLB		×		Graphe planaire	$\notin$ XP	n^{ε}-Inappr.
	USTLB		×		Graphe planaire	$\notin$ XP	n^{ε}-Inappr.
Section 8.3	DSTLB		×		Sans circuits	W[2]-Difficile	XP
Chapitre 10	DSTLB	×	×		Graphe planaire	XP	OUVERT
Section 10.3	USTLB	×	×		Cas général	XP	OUVERT
	USTLB	×	×		Largeur d'arbre bornée	XP	XP
Chapitre 10	DSTLB	×		×		FPT Prob.	FPT Prob.
Section 10.2	USTLB	×		×		FPT Prob.	FPT Prob.

Table 8.1 – *Résumé des résultats. Pour chaque résultat, on précise s'il s'agit de la version orientée ou non du problème, vis-à-vis de quels paramètres il est étudié, si une condition sur le graphe ou l'instance est posée. Les deux dernières colonnes précisent la classe de complexité paramétrée du problème de construction d'une solution réalisable, et de celui d'une solution optimale. La mention* Prob. *précise que l'algorithme est probabiliste.*

8.1.1 DSTLB paramétré par k et p avec $k-1 > p$

Nous allons montrer dans un premier temps que, si nous nous restreignons aux instances du problème DSTLB où $k-1 > p$, le problème consistant à trouver une solution réalisable de l'instance est NP-complet, même si les paramètres k et p sont fixés. Ce résultat sera étendu à un fort résultat d'inapproximabilité du problème DSTLB.

Les deux résultats proviennent d'une réduction depuis le problème de recherche de deux chemins nœuds-disjoints dans un graphe orienté [2].

Problème : Recherche de deux chemins nœuds-disjoints dans un graphe orienté (2-DVDP)

Instance :
- Un graphe orienté $G = (V, A)$
- Deux couples $(s_1, s'_1), (s_2, s'_2)$ de nœuds distincts

Solution réalisable : Un chemin P_1 reliant s_1 à s'_1 et un chemin P_2 reliant s_2 à s'_2 tels que P_1 et P_2 n'aient aucun nœud en commun.

Le problème 2-DVDP est NP-complet [FHW80].

2. 2-Directed Vertex Disjoint Paths, en anglais

NP-complétude de la recherche d'une solution réalisable

Théorème 8.1.1. *Soient k et p deux paramètres entiers fixés vérifiant $k \geq 2$ et $k-1 > p$. Rechercher une solution réalisable pour DSTLB est un problème NP-complet, même si on se restreint aux $(k, p, \mathbb{1})$-instances.*

On suppose jusqu'à la fin de cette section que k et p sont deux paramètres fixés tels que $k-1 > p$. On peut immédiatement remarquer que le problème étudié appartient à la classe NP puisqu'étant donné un sous-graphe de G, on peut en temps polynomial déterminer s'il s'agit d'une arborescence enracinée en r, couvrant tous les terminaux et ne possédant pas plus de p nœuds de branchement.

Nous commençons par présenter la réduction utilisée puis donnons la preuve du théorème.

Réduction. On étudie une instance $\mathcal{I} = (G = (V, A), s_1, s'_1, s_2, s'_2)$ de 2-DVDP. Nous allons transformer cette instance en une instance $\mathcal{I}' = (G' = (V', A'), r, X, \omega, p)$ de DSTLB telle que $|X| = k$. On construit l'instance $\mathcal{I}'$ comme suit :

- On ajoute la racine r en les reliant à s_1.
- On ajoute un premier terminal x_1, successeur de s'_1 et prédécesseur de s_2.
- On construit une chaîne constituée de p nœuds distincts dans cet ordre : $b^1, b^2, \ldots, b^p$. On relie s'_2 à b^1.
- On ajoute p terminaux, $x_2^1, x_2^2, \ldots, x_2^p$. On relie b^i à x_2^i.
- On ajoute les $k - (p+1) \leq 1$ terminaux restants, $x_2^{p+1}, x_2^{p+2}, \ldots, x_2^{k-1}$, comme successeurs de b^p.
- On définit la fonction de poids ω comme la fonction unité.

Un exemple de transformation d'une instance réalisable de 2-DVDP est donné en figure 8.2. Les flèches en pointillés représentent les arcs du graphe d'origine, dont on ne sait pas s'ils permettent deux chemins disjoints entre les couples (s_1, s'_1) et (s_2, s'_2).

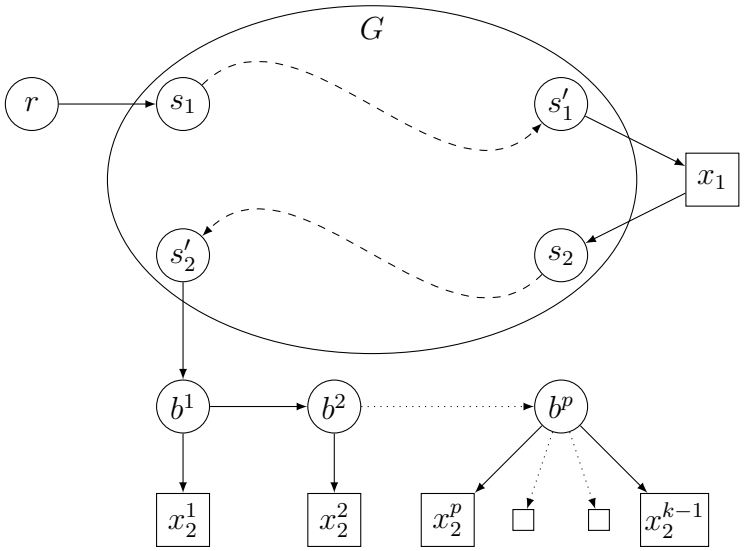


Figure 8.2 – *Exemple de graphe obtenu après transformation*

Démonstration du théorème 8.1.1. Nous allons démontrer que, dans la réduction précédente, il existe une solution réalisable de l'instance $\mathcal{I}$ si et seulement s'il existe une solution réalisable de l'instance $\mathcal{I}'$.

Lemme 8.1.2. *S'il existe un couple de chemins disjoints P_1 et P_2 solutions de l'instance $\mathcal{I}$, alors il existe une solution réalisable T de l'instance $\mathcal{I}'$.*

Démonstration. Le sous-graphe de G' constitué de r, P_1, x_1, P_2, et tous les nœuds b^i et leurs successeurs est une solution réalisable de l'instance $\mathcal{I}'$. □

Lemme 8.1.3. *Supposons qu'il existe une solution réalisable T de l'instance $\mathcal{I}'$, alors b^1, b^2, ..., b^p sont dans T et sont ses seuls nœuds de branchement.*

Démonstration. T est solution de $\mathcal{I}'$, donc il existe un chemin de r vers chaque terminal, en particulier x_2^1,x_2^2, ... et x_2^{k-1}. Donc T contient toute la chaîne reliant s_2' à b^p. Chaque nœud b^i pour $i < p$ a deux successeurs dans T, x_2^i et b^{i+1}. C'est donc un nœud de branchement. De même, puisque $k-1 > p$, le nœud b^p a au moins 2 successeurs dans T, $x_2^p, x_2^{p+1}, \cdots$ et x_2^{k-1}, et est lui aussi un nœud de branchement. □

Lemme 8.1.4. *Supposons qu'il existe une solution réalisable T de l'instance $\mathcal{I}'$, alors il existe une solution réalisable (P_1, P_2) de $\mathcal{I}$.*

Démonstration. Soit P le chemin reliant r à b^1 dans T. Ce chemin relie nécessairement s_1 à s_2'. D'après le lemme 8.1.3, T ne peut contenir aucun nœud de branchement différent de b^1, b^2, ..., b^p. Donc aucun nœud de P n'est un nœud de branchement, mis à part b^1.

En conséquence de quoi, premièrement, ce chemin est élémentaire, et deuxièmement, x_1 est un nœud de P. En effet, si P' est le chemin dans T reliant r à x_1, alors le dernier nœud commun à P et P' est un nœud de branchement, ce qui est exclu, à moins qu'il ne s'agisse de x_1. Puisque s_1' est le seul accès au terminal x_1, et s_2 le seul successeur de x_1, alors P contient s_1, s_1', s_2 et s_2' dans cet ordre.

Puisque P est élémentaire, les sous-chemins P_1 et P_2 de P reliant respectivement s_1 à s_1' et s_2 à s_2' sont nœuds-disjoints. Pour la même raison, ces sous-chemins ne peuvent contenir les arcs (r, s_1), (s_1', x_1) et (x_1, s_2). Enfin aucun autre arc en dehors des arcs de G ne peut être relié à s_2', P ne peut donc également pas les contenir. Ainsi, P_1 et P_2 ne sont constitués que d'arcs de A. Ce couple de chemins est donc une solution réalisable de l'instance $\mathcal{I}$. □

Démonstration du théorème 8.1.1. D'après les lemmes 8.1.2 et 8.1.4, il existe une réduction polynomiale de 2-DVDP vers DSTLB. □

Inapproximabilité

On s'intéresse maintenant au problème de minimisation, toujours sous condition que les paramètres k et p soient fixés et vérifient $k - 1 > p$.

On va modifier légèrement la réduction pour garantir la présence d'une solution réalisable T (et donc d'une solution optimale). Néanmoins son poids sera si élevé qu'un algorithme d'approximation ne pourra la renvoyer que si l'instance privée de T ne possède aucune solution réalisable et donc que le graphe ne contienne pas de chemins disjoints reliant les nœuds s_1 à s_1' et s_2 à s_2'.

Soit H un paramètre entier que nous fixerons plus tard. Une fois l'instance $\mathcal{I}'$ construite à partir de l'instance $\mathcal{I}$ et de la réduction de la partie précédente, on rajoute à $\mathcal{I}'$ les nœuds et arcs suivants : une chaîne C_1 de H arcs de r à x_1, une chaîne C_2 de H arcs de x_1 à x_2^1 et, pour tout $i \in [\![1; k-2]\!]$, une chaîne C_{i+2} de H arcs de x_2^i à x_2^{i+1}. On nomme toujours $\mathcal{I}'$ cette nouvelle instance.

Théorème 8.1.5. *Soient $\varepsilon < 1$, et k et p deux paramètres entiers fixés vérifiant $k \geq 2$ et $k-1 > p$. À moins que $P = NP$, il n'existe pas de n^ε-approximation polynomiale pour DSTLB où n est le nombre de nœuds de l'instance, même si on se restreint aux $(k, p, \mathbb{1})$-instances contenant au moins une solution réalisable calculable en temps polynomial.*

Démonstration. On désignera dans cette démonstration par n le nombre de nœuds dans l'instance $\mathcal{I}$ et n' le nombre de nœuds dans l'instance $\mathcal{I}'$.

Soit $\varepsilon < 1$ un réel positif. Supposons qu'il existe un algorithme d'approximation polynomial $\mathcal{A}$, qui renvoie une solution de poids au plus $(n')^\varepsilon$ fois le poids d'une solution optimale pour toute instance contenant n' nœuds, même si on se restreint aux $(k, p, \mathbb{1})$-instances contenant au moins une solution réalisable calculable en temps polynomial.

Nous montrons que l'existence d'un tel algorithme permet de résoudre le problème 2-DVDP en temps polynomial. Notons que l'instance $\mathcal{I}'$ contient $n' = k * H + n + p + 1$ nœuds.

Si dans $\mathcal{I}$ il existe une solution, alors dans $\mathcal{I}'$, la solution optimale est de taille au plus $n+k+p$ et l'algorithme $\mathcal{A}$ donne une solution de taille au plus $(k*H+n+p+1)^\varepsilon * (n+k+p)$.

S'il n'y a pas de solution dans $\mathcal{I}$, alors, d'après le lemme 8.1.4, il nous est impossible de trouver une solution réalisable de $\mathcal{I}'$ à moins d'utiliser les nœuds d'une des chaînes C_i, pour $i \in [\![1; k]\!]$. Au moins une de ces chaînes est utilisée complètement par toute solution T, sinon on pourrait simplement retirer ces nœuds de T pour trouver une solution réalisable n'utilisant pas les chaînes. La solution optimale est donc au moins de taille H et la solution renvoyée par $\mathcal{A}$ est au moins de taille H également.

Pour que $\mathcal{A}$ puisse décider s'il existe une solution pour l'instance $\mathcal{I}$, il suffit que $H > (k * H + n + p + 1)^\varepsilon * (n + k + p)$. Si $\mathcal{A}$ renvoie une solution de poids plus grand que H, la réponse est "Non", sinon, la réponse est "Oui".

Posons $\sigma = n + p + 1 + k$.

$$
\begin{aligned}
& & H &> (n+p+k)(n+p+1+k*H)^\varepsilon \\
\Leftarrow & & H &> \sigma \cdot (\sigma + \sigma * H)^\varepsilon \\
\Leftarrow & & H &> \sigma^{1+\varepsilon}\sigma \cdot (1+H)^\varepsilon \\
\Leftarrow & H > 1 \text{ et } & H &> \sigma^{1+\varepsilon}(H+H)^\varepsilon \\
\Leftarrow & H > 1 \text{ et } & H &> \sigma^{1+\varepsilon}(2H)^\varepsilon \\
\Leftarrow & H > 1 \text{ et } & H &> 2^\varepsilon\sigma^{1+\varepsilon}H^\varepsilon \\
\Leftarrow & H > 1 \text{ et } & H &> 2\sigma^{1+\varepsilon}H^\varepsilon \\
\Leftarrow & H > 1 \text{ et } & H^{1-\varepsilon} &> 2\sigma^{1+\varepsilon} \\
\Leftarrow & H > 1 \text{ et } & H &> 2^{\frac{1}{1-\varepsilon}}\sigma^{\frac{1+\varepsilon}{1-\varepsilon}} \\
\Leftarrow & & H &= \lceil 2^{\frac{1}{1-\varepsilon}}\sigma^{\frac{1+\varepsilon}{1-\varepsilon}} + 1 \rceil
\end{aligned}
$$

Posons $H = \lceil 2^{\frac{1}{1-\varepsilon}}\sigma^{\frac{1+\varepsilon}{1-\varepsilon}} + 1 \rceil$. Selon la réponse de l'algorithme $\mathcal{A}$, il est possible de savoir en temps polynomial si l'instance $\mathcal{I}$ a oui ou non une solution. Donc si $P \neq NP$ alors un tel algorithme d'approximation n'existe pas. □

Remarque 16. Sachant que toute solution admissible d'une instance de DSTLB à n sommets est une n-approximation, le théorème 8.1.5 prouve que n est le meilleur rapport d'approximation que l'on puisse espérer si les poids sont unitaires.

Remarque 17. Dans le cas du problème DSTLB restreint aux (k, p)-instances (avec des poids différents de 1 sur les arcs), le problème devient inapproximable à rapport quelconque. Si $f(G')$ est le rapport d'approximation dépendant des nœuds et des arcs de G', alors on pourrait remplacer chaque chaîne C_i par un arc a_i de poids $H = f(G') * (n + 2 + k + p) + 1$. Les nombres de nœuds et d'arcs du graphe seraient alors indépendants de H.

Le problème de recherche de deux chemins nœuds-disjoints est un problème NP-complet dans les graphe orientés. Toutefois, il existe de nombreux cas où ce n'est plus vrai, par exemple dans le cas d'un graphe planaire, d'un graphe sans circuit ou d'un graphe non orienté. Dans ces cas, la réduction de cette section ne peut être utilisée.

Dans la suite du chapitre, pour chacun de ces cas, nous allons déterminer les classes de complexité paramétrée auxquelles appartiennent les problèmes DSTLB et USTLB dans le cas où les instances sont restreintes aux graphes planaires. Nous terminerons ce chapitre en parlant du cas des instances sans circuits.

8.2 Restriction aux graphes planaires

Nous allons nous intéresser ici au cas des graphes planaires. Le problème de recherche de deux chemins disjoints n'est plus NP-complet si on restreint ses instances aux graphes planaires orientés ou non [Sch94b].

On étudie la complexité paramétrée des problèmes de l'existence d'une solution réalisable et de minimisation de DSTLB et USTLB vis-à-vis du seul paramètre p.

Nous démontrons des résultats sensiblement similaires à ceux de la sous-section 8.1.1

Dans la suite de cette section, nous désignerons par $(*, p)$-*instance* une instance n'autorisant pas plus de p nœuds de branchement dans la solution. De même, nous désignerons par $(*, p, \mathbb{1})$-*instance* une $(*, p)$-instance dont tous les poids sont unitaires.

NP-complétude de la recherche d'une solution réalisable

Théorème 8.2.1. *Soit p un paramètre entier fixé. Décider s'il existe une solution réalisable pour DSTLB ou USTLB est un problème NP-complet, même si on se restreint aux $(*, p, \mathbb{1})$-instances planaires.*

Démonstration. Soit G un graphe planaire orienté, et u un nœud de G. Décider s'il existe un chemin hamiltonien dans G est un problème NP-complet [GJ79, p199–200]. Donc la recherche d'un chemin hamiltonien ayant u pour origine est également un problème NP-complet. Soit T une arborescence avec p nœuds de branchement. Construisons une instance de DSTLB n'autorisant pas plus de p nœuds de branchement dans la solution comme suit :

- relions la racine de T à u par un arc,
- désignons toutes les feuilles de T comme terminales,
- désignons tous les nœuds de G comme terminaux,
- désignons u comme la racine de l'instance.

Alors clairement, toute solution réalisable couvre les feuilles de T, donc contient T en entier, et donc ses p nœuds de branchement sont dans T. En conclusion, cette solution contient

un chemin élémentaire passant par tous les nœuds de G, c'est-à-dire un chemin hamiltonien de G ayant u pour origine.

Donc décider s'il existe une solution réalisable pour DSTLB est un problème NP-complet, même si on se restreint $(*, p, \mathbb{1})$-instances planaires. Une démonstration similaire prouve le même résultat pour USTLB, sachant que le problème de décider s'il existe un chemin hamiltonien dans un graphe non orienté est également NP-complet [GJ79, p199–200]. □

Résultat d'inapproximabilité

Théorème 8.2.2. *Soit $\epsilon < 1$ un nombre réel et p un paramètre entier fixé. À moins que P = NP, il n'existe pas de n^{ε}-approximation polynomiale pour DSTLB ou USTLB où n est le nombre de nœuds de l'instance, même si on se restreint aux $(*, p, \mathbb{1})$-instances planaires et contenant au moins une solution réalisable calculable en temps polynomial.*

Le reste de cette partie est dédié à la démonstration de ce théorème. Nous allons prouver ce résultat dans le cas orienté, mais la preuve est similaire dans le cas non orienté.

Contrairement à la démonstration du théorème 8.1.5, nous ne pouvons nous contenter de rajouter une chaîne contenant de nombreux arcs et passant par tous les terminaux dans un ordre arbitraire. En effet, il n'est pas dit que rajouter une telle chaîne conserve la planarité du graphe. La démonstration qui suit propose une méthode de construction d'une solution réalisable permettant de prouver le théorème 8.2.2 en conservant la planarité du graphe.

Réduction. Le problème consistant à déterminer s'il existe dans un graphe orienté G un chemin hamiltonien ayant pour origine un nœud v de G est un problème NP-complet, même si on se restreint aux instances planaires 3-connexes [GJT76].

Soit $\mathcal{I} = (G = (V, A), v)$ une instance du problème de recherche d'un chemin hamiltonien, où G est un graphe orienté planaire 3-connexes. Soit p un paramètre entier fixé. Nous allons construire une instance $\mathcal{I}'_v = (G'_v, r, X, \omega, p)$ de DSTLB où G'_v est un graphe orienté planaire et ω la fonction unité. Ce graphe sera découpé en trois parties disjointes par les sommets. Un exemple est donné en figure 8.3. La première partie est un graphe $G' = (V' = V \cup W, A')$ construit à partir de G, où chaque arc de A est remplacé par un chemin en introduisant des nœuds intermédiaires. La deuxième partie est une arborescence binaire $\mathcal{B}$ enraciné en r contenant p nœuds de branchement et $p + 1$ feuilles. On ajoute un arc d'une de ces feuilles vers v avec un arc nommé a_v. L'ensemble des terminaux X de notre instance est l'ensemble des feuilles de $\mathcal{B}$ ainsi que tous les nœuds de V. La troisième et dernière partie de G'_v est un graphe H. On introduit enfin un entier h.

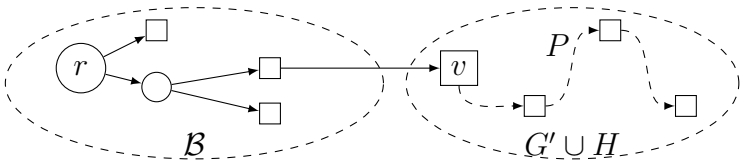


Figure 8.3 – *Exemple de réduction depuis un graphe G contenant 4 nœuds avec $p = 2$. Les nœuds de W $(= V' \backslash V)$, le graphe H et les arcs de G' n'apparaissent pas dans cette figure.*

Nous déterminerons les graphes G', H et l'entier h en fin de partie. Nous supposons pour l'instant qu'ils vérifient les trois assertions suivantes.

Hypothèse 1. Soient n, $n_{G'}$ et n_H le nombre de nœuds respectifs dans G, G' et H. Alors $n_{G'} - n \leq n^3$ et $n_{G'} - n + n_H \leq 4 \cdot n^3 \cdot h$.

Hypothèse 2. Il existe un chemin élémentaire P de $G' \cup H$ passant par tous les nœuds de G et ayant v pour origine.

Hypothèse 3. Tout chemin élémentaire de $G' \cup H$ passant par tous les nœuds de G, ayant v pour origine, et utilisant au moins un nœud de H en dehors de ses extrêmités, contient au moins h nœuds de H.

L'hypothèse 2 s'assure de l'existence d'une solution réalisable. Les hypothèses 1 et 3 vont mener au résultat d'inapproximabilité. Si h est suffisamment grand, alors l'hypothèse 3 s'assure qu'aucune solution réalisable contenant un nœud de H ne peut être renvoyée par un algorithme d'approximation polynomial sans dépasser le rapport n^ε fixé dans le théorème.

Démonstration du théorème 8.2.2. Nous démontrons maintenant, sous réserve que les hypothèses 1, 2 et 3 soient vérifiées, notre résultat d'inapproximabilité.

Démonstration du théorème 8.2.2. Soit $\epsilon < 1$, supposons un algorithme d'approximation $\mathcal{A}$ qui renvoie une solution de poids au plus n'^{ϵ} fois le poids d'une solution optimale pour toute instance contenant n' nœuds même si on se restreint aux $(*, p, \mathbb{1})$-instances planaires, et contenant au moins une solution réalisable calculable en temps polynomial.

Nous montrons que l'existence d'un tel algorithme d'approximation permet de résoudre l'instance $\mathcal{I}$ en temps polynomial en utilisant la réduction précédente.

Soit T^* une solution optimale de l'instance $\mathcal{I}'_v$. Cette solution existe car l'instance contient au moins une solution réalisable : $\mathcal{B} \cup P \cup a_v$, d'après l'hypothèse 2. Notons que l'instance $\mathcal{I}'_v$ contient $n' = n_{G'} + n_H + p + (p+1)$ nœuds.

S'il existe un chemin hamiltonien dans l'instance $\mathcal{I}$, alors T^* contient au plus $n_{G'} + 2p + 1$ nœuds : $n_{G'}$ nœuds dans G' et $2p+1$ nœuds dans $\mathcal{B}$. On en déduit que cette solution contient au plus $n_{G'} + 2p$ arcs. Donc toute solution approchée renvoyée par $\mathcal{A}$ a un poids au plus $(n_{G'} + 2p) \cdot n'^{\epsilon}$.

Supposons maintenant que l'instance $\mathcal{I}$ ne possède pas de solution. Alors, sans utiliser les nœuds de H, il est impossible de construire un chemin élémentaire passant par tous les nœuds de G d'origine v. Or, puisque $\mathcal{B}$ contient exactement p nœud de branchement, toute solution réalisable de $\mathcal{I}'_v$ contient un chemin élémentaire passant par tous les nœuds de G ayant v pour origine. En conclusion, sans H, il est impossible de trouver une solution réalisable de $\mathcal{I}'_v$. Donc toute solution renvoyée par $\mathcal{A}$ utilise des nœuds de H. Au moins un de ces nœuds n'est pas une extrémité du chemin élémentaire de cette solution, sans quoi nous pourrions tout simplement retirer ces nœuds et obtenir une solution réalisable n'utilisant aucun nœud de H. D'après l'hypothèse 3, cette solution contient au moins h nœuds de H. Donc elle a un poids au moins égal à h.

Pour que $\mathcal{A}$ puisse décider s'il existe une solution pour l'instance $\mathcal{I}$, il suffit que $h > (n_{G'} + 2p) \cdot ((n + 2p + 1) + n_{G'} + n_H - n)^{\epsilon}$. Si $\mathcal{A}$ renvoie une solution de poids plus grand que h alors la réponse est "Non", sinon la réponse est "Oui".

$$h > (n_{G'} + 2p) \cdot ((n + 2p + 1) + n_{G'} + n_H - n)^\epsilon$$
$$\Leftarrow \quad h > (n_{G'} + 2p) \cdot ((n^3 + 2p + 1) + n_{G'} + n_H - n)^\epsilon$$

D'après l'hypothèse 1, $n_{G'} - n + n_H \leq 4 \cdot n^3 \cdot h$.

$$\Leftarrow \quad h > (n_{G'} + 2p) \cdot ((n^3 + 2p + 1) + 4 \cdot n^3 \cdot h)^\epsilon$$
$$\Leftarrow \quad h > (n_{G'} + 2p) \cdot (1 + 4h)^\epsilon (n^3 + 2p + 1)^\epsilon$$

D'après l'hypothèse 1, $n_{G'} - n \leq n^3$.

$$\Leftarrow \quad h > (n^3 + n + 2p + 1)(n^3 + 2p + 1)^\epsilon \cdot (1 + 4h)^\epsilon$$
$$\Leftarrow \quad h > 1 \text{ et } h > (2n^3 + 2p + 1)^{1+\epsilon} (5h)^\epsilon$$
$$\Leftarrow \quad h > 1 \text{ et } h^{1-\varepsilon} > (2n^3 + 2p + 1)^{1+\epsilon} (5)^\varepsilon$$
$$\Leftarrow \quad h > 1 \text{ et } h > (2n^3 + 2p + 1)^{\frac{1+\epsilon}{1-\varepsilon}} 5^{\frac{\epsilon}{1-\varepsilon}}$$

Posons $h = \lceil 5^{\frac{\epsilon}{1-\epsilon}} (2n^3 + 2 \cdot p + 1)^{\frac{1+\epsilon}{1-\epsilon}} + 1 \rceil$. Selon la réponse de l'algorithme $\mathcal{A}$, il est possible de savoir en temps polynomial si l'instance $\mathcal{I}$ a oui ou non une solution. Donc si $P \neq NP$ alors un tel algorithme d'approximation n'existe pas. □

Existence de G' et H. Nous démontrons maintenant qu'il est possible de construire de tels graphes G' et H vérifiant les hypothèses 1, 2 et 3. Ainsi, la preuve du théorème 8.2.2 sera terminée.

G' est construit à partir de G en introduisant des nœuds intermédiaires et en divisant tout arc a en plusieurs arcs de A'. L'ensemble des nœuds intermédiaires rajoutés est W.

G est un graphe 3-connexe planaire. Nous sommes donc en mesure de le plonger dans le plan sous la forme d'un polygone convexe tel que v soit sur la face extérieure de ce polygone. Pour cela, nous pouvons par exemple utiliser la technique de [Ple99]. Pour tout nœud $w \in V$, on désigne les coordonnées de ce nœud dans le plan par x_w et y_w.

On dira qu'un nœud v est *à gauche* du graphe si, pour tout autre nœud w, $x_v \leq x_w$.

Lemme 8.2.3. *Il existe un angle α tel que la rotation d'angle α et de centre v tourne le graphe G dans le plan de telle façon que v soit à gauche du graphe et que chaque nœud $w \in V$ possède une unique abscisse x_w.*

Démonstration. Puisque G est un polygone convexe contenant v sur sa face externe, il existe deux angles α_m et α_M de $[0; 2\pi]$ tels que pout tout angle $\alpha \in [\alpha_m; \alpha_M]$, v est placé à gauche du graphe. Si nous supposons le lemme erroné, alors pour toute rotation d'angle compris dans $[\alpha_m; \alpha_M]$, il existe deux nœuds u et w avec même abscisse. Cependant, il existe tout au plus n^2 angles vérifiant cette propriété, donc tout autre angle $[\alpha_m; \alpha_M]$ permet de démontrer le lemme. □

Trions les nœuds par abscisse croissante : $x_v = x_{v_1} < x_{v_2} < x_{v_3} < ... < x_{v_n}$. On définit D_i, pour $i \in [2..n]$, comme étant la droite verticale d'abscisse $x_i = (x_{v_{i-1}} + x_{v_i})/2$ dans le plan. Pour tout arc $a = (t, u)$ de G croisant D_i, on ajoute un nœud w à W à l'intersection entre a et D_i et on remplace a par deux arcs (t, w) et (w, u). Un exemple est donné Figure 8.4.

Puisqu'aucun arc ajouté n'en croise un autre, le graphe obtenu G' reste planaire.

Nous pouvons désormais construire H. Commençons par prouver quelques lemmes utiles par la suite.

Lemme 8.2.4. *Pour tout nœud $v_i \in V$, $i \in [2..n]$, il est possible d'ajouter au graphe H un nœud $v_{i,l}$ sur la droite D_i et un arc $(v_{i,l}, v_i)$ de sorte que le graphe $G' \cup H$ reste planaire.*

Démonstration. Il faut remarquer tout d'abord que D_i contient au plus $|A|$ nœuds puisqu'il s'agit d'intersections avec les arcs de G : soient $\{v'_1, v'_2, \ldots, v'_{|D_i|}\}$ ces nœuds, classés de la plus haute ordonnée à la plus basse. Soit v'_j le nœud de D_i ayant la plus grande ordonnée qui soit également incident à un arc de G' passant 'sous v_i', ou relié à v_i. Si aucun arc de cette sorte n'existe, on ajoute $v_{i,l}$ n'importe où en-dessous de $v'_{|D_i|}$. Puisqu'il n'y a aucun nœud de D_i sous $v'_{|D_i|}$, aucun arc de G' ne peut croiser $(v_{i,l}, v_i)$, et le graphe $G' \cup H$ reste planaire.

Dans le cas contraire, aucun arc de G' passant au dessus de v_i n'est relié à un sommet de D_i se trouvant sous v'_j, sinon G ne serait pas planaire. Donc nous pouvons ajouter $v_{i,l}$ sur le segment reliant v'_{j-1} et v'_j si $j \neq 1$, ou ajouter $v_{i,l}$ au dessus de v'_1 si $j = 1$. Aucun arc de G' ne croise $(v_{i,l}, v_i)$, et le graphe $G' \cup H$ reste planaire. □

De même, pour tout nœud $v_i \in V$, $i \in [2..n]$ il est possible d'ajouter à H un nœud $v_{i,r}$ sur la droite D_{i+1} et un arc $(v_i, v_{i,r})$ de sorte que le graphe $G' \cup H$ reste planaire.

Enfin, pour tout $i \in [2..n]$, nous trions tous les nœuds de $G' \cup G$ d'abscisse x_i par ordonnée croissante. Pour tout couple (u, t) de nœuds consécutifs de même abscisse, on ajoute à H un chemin de h nœuds allant de u à t et un autre chemin allant de t à u partageant les mêmes h nœuds, de sorte que $G' \cup H$ reste planaire. Un exemple est donné Figure 8.4.

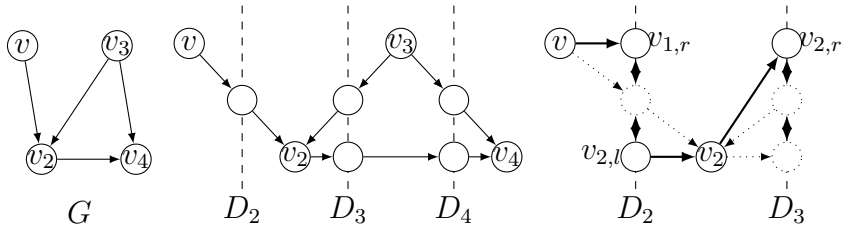


Figure 8.4 – *Exemple de construction de $G' \cup H$ à partir d'un graphe G contenant 4 nœuds (image de gauche). On construit en premier lieu W (image du centre) puis H. Seul un sous-graphe de H est dessiné sur l'image de droite. Chaque arc vertical est en réalité un chemin de h nœuds.*

Lemme 8.2.5. *G', H and h vérifient les hypothèses 1, 2 and 3.*

Démonstration. $n'_G - n$ et n_H sont le nombre d'arcs dans W et H. En d'autres termes, il s'agit du nombre de nœuds sur l'ensemble des droites D_i. Pour chaque droite D_i, nous créons au plus m nœuds dans G' et $2 + h \cdot (1 + m)$ nœuds dans H. Donc $n'_G - n \leq (n-1) \cdot m \leq n^3$ et $n'_G - n + n_H \leq (n-1)(2 + h(1+m) + m) = n(1+m)(h+1) + (n-1) - (1+m)(h+1)$.

Puisqu'un graphe connexe vérifie $n - 1 \leq m$ et puisque $h \geq 0$, on peut montrer que $(n-1) < (1+m)(h+1)$. Donc $n'_G + n_H - n \leq n(1+m)(h+1) < n \cdot (2m) \cdot (2h) < 4n^3 h$. L'hypothèse 1 est donc vérifiée. Si le graphe G n'est pas connexe, alors l'instance $\mathcal{I}$ n'a trivialement aucune solution.

Le chemin P commençant en v, puis allant de $v_{i-1,r}$ à $v_{i,l}$ et v_i en passant par D_i pour tout $i \in [2..n]$ passe par tous les nœuds de G. L'hypothèse 2 est donc vérifiée.

Soit P un chemin élémentaire passant par tous les nœuds de G et au moins un nœud H qui n'est pas une de ses extrémités. Puisque les seuls nœuds de H reliés à un nœud de G sont $v_{i-1,r}$ et $v_{i,l}$, $i \in [2..n]$, P contient un nœud $t = v_{i-1,r}$ ou $t = v_{i,l}$. Puisqu'aucun nœud de D_i ne permet à la fois d'arriver sur D_i et d'en repartir, P contient nécessairement un autre nœud t' de D_i, distinct de t, ainsi que tous les nœuds reliant t à t' ou t' à t sur D_i. Ainsi, P contient nécessairement h nœuds de D_i. L'hypothèse 3 est vérifiée. □

Puisqu'il existe deux graphes G' et H vérifiant les hypothèses 1, 2 and 3, alors la démonstration du théorème 8.2.2 de cette section est justifiée.

La dernière section de ce chapitre s'intéresse aux cas des graphes orientés sans circuits. Nous verrons que, contrairement aux cas précédents, ce problème n'est pas NP-complet.

8.3 Restriction aux graphes sans circuits

Le problème de recherche de deux chemins disjoints n'est plus NP-complet si on restreint ses instances aux graphes orientés sans circuits ou non [FHW80].

Dans cette section, nous étudions le problème DSTLB dans ce cas, paramétré par p. Elle contient un résultat montrant que ce problème est dans XP. Un second résultat montre qu'il n'est cependant pas FPT, sauf si la hiérarchie des classes de complexité paramétrée s'effondre, puisqu'il est au mieux dans la classe W[2].

8.3.1 Un algorithme XP en p pour DSTLB dans un graphe sans circuits

On se place dans une instance $\mathcal{I} = (G = (V, A), r, X, \omega, p)$ de DSTLB où G est un graphe sans circuits. L'idée directrice de cet algorithme est de tester chaque configuration possible d'au plus p nœuds de branchement dans G. Il existe $\sum_{i=1}^{p} n^i$ configurations. Nous allons montrer par la suite que, connaissant une configuration de nœuds de branchement, il est polynomial, dans un graphe sans circuits, de trouver une solution réalisable passant par tous les nœuds de cette configuration et n'utilisant que ces nœuds comme nœuds de branchement.

Trouver une solution optimale connaissant les nœuds de branchement à utiliser

Soit $\kappa = (v_1, v_2, \ldots, v_i)$ un ensemble de $i \leq p$ nœuds du graphe. Nous allons utiliser une instance du problème de flot de coût minimum pour trouver une arborescence de poids minimum passant par κ et n'utilisant que ces nœuds comme nœuds de branchement. On

rappelle ci-après la définition du problème de flot de coût minimum.

Problème : Problème de flot de coût minimum

Instance :

- Un graphe $G = (V, A)$.
- Une fonction de coût ω sur les arcs, à valeur positives ou nulles.
- Une fonction de capacité c sur les nœuds de V, à valeur positives ou nulles (ou même infinie).
- Deux nœuds S et S' appelés respectivement *sources* et *puits*.
- Un entier l.

Solution réalisable : Un flot entier de l unités traversant le graphe pour relier S à S' tout en respectant la contrainte de capacité sur les nœuds, c'est-à-dire une fonction F sur les arcs à valeurs entières respectant les contraintes suivantes :

- création du flot à la source : $\sum_{a \in \Gamma^+(S)} F(a) = l$;
- disparition du flot au puits : $\sum_{a \in \Gamma^-(S')} F(a) = l$;
- conservation du flot : pour tout nœud v de V, $\sum_{a \in \Gamma^-(v)} F(a) = \sum_{a \in \Gamma^+(v)} F(a)$;
- contrainte de capacité sur les nœuds : pour tout nœud v de V, $\sum_{a \in \Gamma^-(v)} F(a) \leq c(v)$.

Optimisation : Minimiser le coût du flot $\omega(F) = \sum_{a \in A} F(a) \cdot \omega(a)$.

Soit $\mathcal{I}_\kappa = (G_\kappa, \omega, c, S, S', l)$ l'instance du problème de flot de coût minimum que nous allons construire à partir de l'instance $\mathcal{I}$. Un exemple d'une telle transformation est donné en figure 8.5.

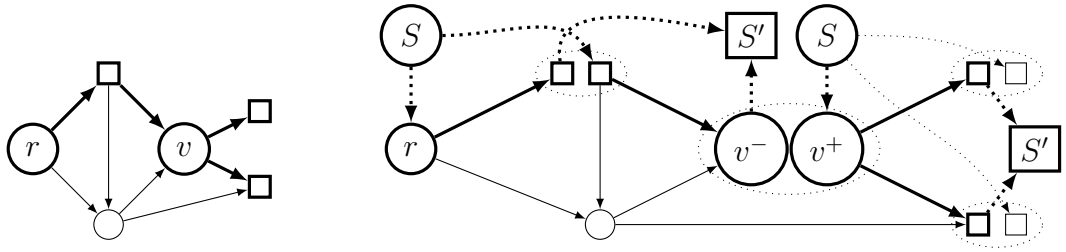


Figure 8.5 – *Exemple de transformation d'une instance $\mathcal{I}$ (sur la gauche) en une instance $\mathcal{I}_{\{v\}}$ (sur la droite). La valeur du paramètre l dans cette seconde instance est 4. Dans le but d'avoir une meilleure lisibilité, la source S et le puits S' sont dupliqués sur cet exemple. Les éléments en gras constituent une solution réalisable des deux instances. A droite, il s'agit de tous les arcs contenant une ou plusieurs unités de flot.*

La fonction de poids de l'instance $\mathcal{I}$ et la fonction de coût de l'instance $\mathcal{I}_\kappa$ sont nommées de manière identique par commodité. Cependant, il faut noter que certains arcs ou nœuds vont être modifiés ou ajoutés et que la fonction sera modifiée en conséquence.

Soit $\mathcal{D} = \kappa \cup X$. Ces nœuds sont l'ensemble des nœuds que notre arborescence doit couvrir. Pour construire $\mathcal{I}_\kappa$, on transforme $\mathcal{I}$ de la manière suivante :

- On dédouble chaque nœud v de $\mathcal{D}$ en deux nœuds v^-, de capacité 1, et v^+, de capacité infinie. L'ensemble des nœuds v^- est dénommé $\mathcal{D}^-$. L'ensemble des nœuds v^+ est dénommé $\mathcal{D}^+$.
- Pour chaque nœud v de D, on procède de la façon suivante. Chaque arc entrant en v est maintenant un arc entrant en v^-. Chaque arc sortant de v est maintenant un arc sortant de v^+. Le poids de chacun de ces arcs ne change pas.
- Tout autre nœud est de capacité 1.
- On ajoute les nœuds source S et puits S' au graphe.
- Si $r \notin \mathcal{D}$, on relie S à r. Sinon on relie S à r^-. Le poids de cet arc est nul.
- On relie S à tout nœud de $\mathcal{D}^+$. Le poids de chacun de ces arcs est nul.
- On relie tout nœud de $\mathcal{D}^-$ à S'. Le poids de chacun de ces arcs est nul.
- On fixe $l = |\mathcal{D}| \leq k + i$.

Nous allons montrer que cette instance du problème de flot de coût minimum, une fois résolue, fournit, dans le cas où une solution F existe, une arborescence T_κ par le biais des arcs de A de flot strictement positif. Cette arborescence couvre X et κ et n'utilise que ces derniers nœuds comme nœuds de branchement. Si aucune solution n'existe pour l'instance $\mathcal{I}_\kappa$ alors il n'existe pas de telle arborescence.

Supposons qu'il existe une solution F pour l'instance $\mathcal{I}_\kappa$. Soit T_κ le sous-graphe de G contenant r et tous les arcs de flot strictement positif. On peut remarquer que le poids de T_κ et le coût de F sont égaux.

Lemme 8.3.1. *Le flot F passe par tous les nœuds de $\mathcal{D}^-$.*

Démonstration. En effet, seuls les nœuds de $\mathcal{D}^-$ sont des prédécesseurs du puits. Ils ont tous une capacité égale à 1. Étant donné que $l = |\mathcal{D}|$ unités de flots sont envoyées depuis la source jusqu'au puits et que le flot est entier, chacun de ces nœuds reçoit une unité de flot. □

Lemme 8.3.2. *T_κ est une solution réalisable pour $\mathcal{I}$ couvrant κ. Tous ses nœuds de branchement sont dans κ.*

Démonstration. D'après le lemme 8.3.1 le flot F passe par tous les nœuds de $\mathcal{D}^-$, donc T_κ contient $\mathcal{D} = \kappa \cup X$. Il contient r par définition. Puisque $\mathcal{D}^+$ contient les seuls nœuds de $\mathcal{I}_\kappa$ de capacité infinie, excepté S et S', tous les nœuds de branchement de T_κ sont dans κ.

T_κ n'a qu'une composante connexe. Sinon ce sous-graphe contiendrait une composante connexe ne contenant pas r. Puisque le graphe est sans circuits, il existe un nœud de cette composante sans prédécesseur. Soit w ce nœud. F passe par tous les nœuds de $\mathcal{D}^-$, en particulier w^-. Ce flot est passé par un prédécesseur de w^- autre que S car le seul nœud de $\mathcal{D}^-$ auquel S peut être relié est éventuellement r^-, mais $w \neq r$. Ainsi, dans T_κ, w devrait avoir un prédécesseur et ne peut être racine d'une composante connexe.

Pour terminer, T_κ est une arborescence. Dans le cas contraire, ce sous-graphe contiendrait un cycle ou un circuit. Il ne peut y avoir de circuit, car G n'en possède pas. Il n'y a pas non plus de cycle car exceptés S et S', seuls les nœuds de $\mathcal{D}^+$ ont une capacité $c > 1$ dans $\mathcal{I}_\kappa$. Leur unique prédécesseur étant S, et le flot étant entier, aucun autre nœud ne peut avoir plusieurs prédécesseurs leur envoyant du flot. □

Lemme 8.3.3. *T_κ est une solution réalisable pour $\mathcal{I}$ de poids minimum parmi toutes celles couvrant les nœuds de κ et les utilisant comme nœuds de branchement.*

Démonstration. S'il existait une telle solution réalisable T' de poids moindre, alors on pourrait construire un flot F' de coût plus petit que celui de F en effectuant l'opération inverse de celle qui a construit T_κ : pour tout arc a de T', mettre le flot $F'(a)$ de cet arc à 1, et déduire le flot des arcs sortant de S et entrant en S' de la contrainte de conservation du flot. Puisque ces derniers arcs sont de poids 0, le poids de T' et le coût de F' sont identiques. Puisque c'est également le cas du poids de T_κ et du coût du F alors le coût de F' est strictement plus petit que celui de F ce qui est exclu. □

Complexité paramétrée de l'algorithme complet

Pour chaque ensemble κ, on construit l'arbre T_κ avec la méthode décrite dans la partie précédente. Une fois ceci fait, on renvoie l'arbre T_κ de poids minimum.

Théorème 8.3.4. *Cet algorithme renvoie une solution optimale pour l'instance $\mathcal{I}$ en temps XP vis-à-vis du paramètre p.*

Démonstration. Soit κ^* l'ensemble de nœuds de branchement pour lequel T_{κ^*} est de poids minimum. D'après le lemme 8.3.2, il s'agit d'une solution réalisable pour $\mathcal{I}$. Soit $T^\diamond$ une solution optimale pour $\mathcal{I}$ et $\kappa^\diamond$ ses nœuds de branchement. D'après les lemmes 8.3.2 et 8.3.3, $T_{\kappa^\diamond}$ est une solution réalisable de poids plus petit que $T^\diamond$.

Par définition de T_{κ^*}, son poids est plus petit que celui de $T_{\kappa^\diamond}$, donc que la solution optimale $T^\diamond$. En conclusion, T_{κ^*} est également une solution optimale pour $\mathcal{I}$. On itère sur $\binom{n}{1} + \binom{n}{2} + ... + \binom{n}{p} = O(n^p)$ différents ensembles κ.

Il est possible de résoudre une instance du problème de flot de coût minimum en temps $O(n^2m^3\log(n))$ si la capacité est associée aux arcs [GT89]. Nos instances définissent une capacité sur les nœuds. Nous pouvons toutefois utiliser cet algorithme en associant premièrement une capacité infinie sur tous les arcs du graphe, et en transformant tout nœud v de capacité c non infinie en un couple de nœud v_1 et v_2 reliés par un arc (v_1, v_2) de capacité c tel que tout arc entrant en v est maintenant un arc entrant en v_1 et tout arc sortant de v est maintenant un arc sortant de v_2. On vérifie aisément qu'à partir du flot obtenu, il est possible de calculer en temps $O(m+n)$ un flot de même coût pour notre instance. On a au plus doublé le nombre de nœud et ajouté un arc par nœud pour créer la seconde instance. La complexité de l'algorithme de [GT89] est donc dans notre cas $O((2n)^2(m+n)^3\log(2n))$. On en déduit donc que le temps total de l'algorithme est $O(n^{p+2}(m+n)^3\log(2n))$. □

8.3.2 W[2]-difficulté

Cette partie pointe le fait que DSTLB n'est pas FPT vis-à-vis du paramètre p, à moins que FPT = W[2]. En outre, sous la même contrainte, décider s'il existe une solution réalisable à ce problème n'est pas FPT en p.

Théorème 8.3.5. *Le problème de décider si une instance de DSTLB contient une solution réalisable ou non est $W[2]$-difficile vis-à-vis du paramètre p même si le graphe est sans circuits.*

Démonstration. Pour démontrer le théorème 2.3.3, page 23, nous avons utilisé une réduction depuis le problème de couverture par ensembles vers le problème de Steiner. Nous allons légèrement modifier cette réduction pour prouver notre théorème. Nous rappelons la définition du problème, paramétré par le poids de sa solution optimale. Etant donnés un univers U, un ensemble S de parties de U, et un entier N, le problème de couverture d'ensemble demande s'il existe N ensembles ou moins dans S couvrant U. Ce problème est $W[2]$-complet vis-à-vis du paramètre N [**?**]. On suppose dans la suite que $|S| > 1$.

Nous allons définir une réduction FPT de ce problème paramétré vers celui consistant à décider de l'existence ou non d'une solution admissible dans une instance $\mathcal{I} = (G = (V, A), r, X, \omega)$ sans circuits du problème DSTLB paramétré par p. Un exemple est donné Figure 8.6. Pour tout ensemble s dans S, nous ajoutons un *nœud-ensemble* s dans V, ainsi qu'un terminal t_s et relions s à t_s. On définit X_S comme l'ensemble des terminaux t_s. Pour tout élément de l'univers U, nous ajoutons un *terminal-élément* dans un ensemble X_e. On a donc défini $X = X_S \cup X_e$. On ajoute pour finir une racine r dans V. On relie r à tous les nœuds-ensembles, et on relie chaque nœud-ensemble à un terminal-élément si l'ensemble correspondant contient l'élément correspondant. On fixe le paramètre p à $N + 1$.

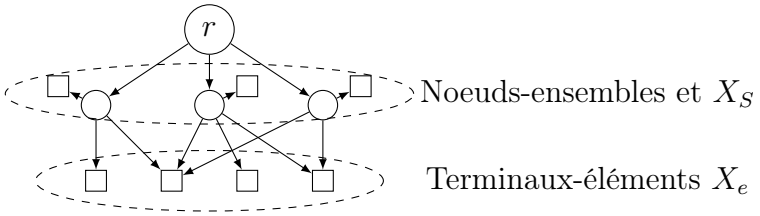


Figure 8.6 – *Exemple de réduction depuis une instance du problème de couverture par ensemble vérifiant $U = \{x_1, x_2, x_3, x_4\}$ et $S = \{\{x_1, x_2\}, \{x_2, x_3, x_4\}, \{x_2, x_4\}\}$*

Puisqu'une solution réalisable T de $\mathcal{I}$ couvre tous les terminaux, elle couvre X_S et contient donc tous les nœuds-ensembles. La racine est donc un nœud de branchement, sauf si $|S| = 1$ ce qui est exclu par hypothèse. De plus, le degré sortant de ces nœuds dans T est donc au moins 1. En conclusion, chaque nœud-ensemble servant à couvrir un terminal-élément dans T est un nœud de branchement.

S'il existe une couverture $c \subset S$ vérifiant $|c| \leq N$, l'arborescence utilisant chaque nœud-ensemble correspondant à un ensemble de c utilisé pour couvrir X_e contient donc $|c| + 1 \leq p$ nœuds de branchement. Inversement, s'il existe une arborescence T utilisant les nœuds-ensembles $c \subset V$ pour couvrir X_e, le sous-ensemble correspondant de S couvre tout l'univers U avec $|c| \leq N$ ensembles. Cette réduction FPT montre que le problème de décider si une instance de DSTLB contient une solution réalisable ou non est $W[2]$-difficile vis-à-vis du paramètre p même si le graphe est sans circuits. □

8.3.3 Complexité paramétrée vis-à-vis du paramètre k

Dans le cas orienté, le paramètre p est borné par $k - 1$. Si k est fixé, alors p est également fixé. L'algorithme présenté en début de cette section est donc également XP en k.

Savoir s'il existe un algorithme FPT en k pour DSTLB dans le cas d'un graphe orienté sans circuits est un problème ouvert.

8.4 Synthèse des résultats

Dans ce chapitre, nous avons présenté des résultats concernant les problèmes de l'arbre et de l'arborescence de Steiner avec un nombre limité de nœuds de branchement. Les résultats démontrés dans ce chapitre sont résumés dans le tableau 8.7.

Problème	k	p	n_s	Conditions	Trouver une solution réalisable	Minimisation
DSTLB	×	×		$p \leq k-2$	$\notin$ XP	n^{ε}-Inappr.
DSTLB		×		Graphe planaire	$\notin$ XP	n^{ε}-Inappr.
USTLB		×		Graphe planaire	$\notin$ XP	n^{ε}-Inappr.
DSTLB		×		Sans circuits	W[2]-Difficile	XP
DSTLB	×			Sans circuits	XP	XP

Table 8.7 – *Résultats démontrés de ce chapitre.*

Nous avons également mentionné le fait qu'il existe un algorithme d'approximation de rapport n pour les problèmes DSTLB et USTLB, à partir du moment où il est possible de construire au moins une solution réalisable en temps polynomial.

La dernier ligne du tableau 8.7 présente le cas orienté sans circuits paramétré par k. Les problèmes de décider si une instance orientée sans circuits possède une solution réalisable et de trouver une solution de poids minimum, si elle existe, dans cette instance sont des problèmes XP en k. Savoir si ces problèmes sont FPT vis-à-vis du paramètre k est une question ouverte.

Les résultats de ce chapitre ont été publiés à la conférence SIROCCO 2013 [WWBB13].

Chapitre 9.

DST avec un nombre limité de nœuds diffusants

Nous rappelons ci-après la définition du problème de Steiner avec un nombre limité de nœuds diffusants [1].

Problème : Problème de Steiner avec un nombre limité de nœuds diffusants (DSTLD)

Instance :
- Un graphe orienté $G = (V, A)$.
- Un nœud *racine* $r \in V$.
- Un ensemble de k nœuds *terminaux* $X \subset V$, tous feuilles de G.
- Une fonction de poids $\omega : A \to \mathbb{R}^+$.
- Un entier $d \in [\![1; k-1]\!]$.

Solution réalisable : Une arborescence T de l'instance des plus courts chemins $\mathcal{I}^{\triangleright} = (G^{\triangleright}, r, X, \omega^{\triangleright})$, enracinée en r, couvrant tous les terminaux et ne possédant pas plus de d nœuds de branchement.

Optimisation : Minimiser le poids $\omega^{\triangleright}(T) = \sum_{a \in T} \omega^{\triangleright}(a)$.

Tout nœud de branchement dans l'instance $\mathcal{I}^{\triangleright}$ est associé à un nœud diffusant dans l'instance $\mathcal{I}$ et qu'à tout arc d'une arborescence de $\mathcal{I}^{\triangleright}$ est associé un plus court chemin dans $\mathcal{I}$. Pour éviter toute ambigüité entre les instances, on parlera des nœuds diffusants d'une solution réalisable de $\mathcal{I}$ et des nœuds de branchement d'une arborescence de $\mathcal{I}^{\triangleright}$.

La section suivante introduit les résultats préliminaires associés à DSTLD (classes de complexité et de complexité paramétrée, et algorithme d'approximation trivial). Les deux dernières sections contiennent respectivement un résultat d'approximabilité pour le problème de Steiner conçu à l'aide de DSTLD, et un résultat d'inapproximabilité pour DSTLD. Nous verrons enfin dans le chapitre 10 un algorithme exact pour DSTLD, adaptable également à DST et DSTLB, et leurs versions non orientées.

1. *Directed Steiner Tree with Limited number of Diffusing nodes*, en anglais

On rappelle que l'on fait l'hypothèse que les terminaux des instances de DSTLD sont des feuilles. Ainsi les arcs sortant des terminaux dans $G^{\triangleright}$ sont de poids infinis. Si ce n'est pas le cas pour un terminal, on duplique le terminal, puis on relie l'original et la copie par un arc de poids nul, enfin on retire de X l'original et on y insère la copie.

9.1 Propriétés du problème

Cette section est dédiée à la description de propriétés du problème : classes de complexité, classes de complexité paramétrée et algorithme d'approximation polynomial.

9.1.1 Complexité et complexité paramétrée

Théorème 9.1.1. *DSTLD est NP-complet même si $k - d$ est un entier fixé.*

Démonstration. Ce problème appartient à la classe NP car connaissant une arborescence du graphe des plus courts chemins, il est possible de savoir en temps linéaire si elle respecte les contraintes d'une solution réalisable de l'instance.

Soit p un entier fixé. Soit une instance $\mathcal{I} = (G, r, X, \omega)$ de l'arborescence de Steiner à k terminaux. On rappelle qu'il a été démontré dans la preuve du théorème 2.1.2, en page 16 du chapitre 2, qu'il est équivalent, pour DST, de travailler avec $\mathcal{I}$ et avec $\mathcal{I}^{\triangleright}$.

Posons $d' = k$ et créons une instance $\mathcal{I}' = (G', r, X', \omega', d')$ de DSTLD en ajoutant p nouveaux terminaux à $\mathcal{I}^{\triangleright}$ et en les reliant à la racine avec un arc de poids nul. On complète avec des arcs reliant V à ces nouveaux terminaux et inversement pour obtenir $\mathcal{I}'^{\triangleright}$. Soit $k' = |X'|$, on peut remarquer que $p = k' - d'$.

Une solution réalisable T de $\mathcal{I}^{\triangleright}$ permet de construire une arborescence T' de $\mathcal{I}'^{\triangleright}$ de poids identique en ajoutant à T les arcs reliant r à $X' \backslash X$. L'arborescence T' contient au plus d' nœuds de branchement. En effet, une arborescence de Steiner couvrant k terminaux ne peut contenir plus de $k - 1$ nœuds de branchement, et les nœuds de branchement de T' sont ceux de T ainsi que la racine. C'est donc une solution réalisable de $\mathcal{I}'$. Inversement, d'une solution réalisable T' de $\mathcal{I}'$ peut être extraite une solution réalisable T de $\mathcal{I}^{\triangleright}$ de poids plus petit en retirant de T' les arcs menant et provenant de $X' \backslash X$.

On en conclut qu'il existe une réduction du problème de Steiner vers le problème DSTLD restreint aux instances où $k' - d' = p$. Ce dernier problème est donc NP-complet. □

En conclusion, le problème DSTLD n'est pas XP vis-à-vis du paramètres $k - d$.

Théorème 9.1.2. *DSTLD est W[2]-difficile en d.*

Cette preuve se démontre à l'aide d'une réduction similaire à celle qui est donnée dans le chapitre 8 pour le Théorème 8.3.5. Nous verrons en fin de ce chapitre une preuve d'inapproximabilité du problème DSTLD qui utilise cette réduction.

Théorème 9.1.3. *DSTLD est XP en d.*

Démonstration. Soit d un entier fixé, montrons qu'on peut résoudre toute instance $\mathcal{I}$ autorisant le placement d'au plus d nœuds diffusants en temps polynomial.

Soit T^* une solution optimale pour $\mathcal{I}$. On sait que cette arborescence ne possède pas plus de d nœuds de branchements. Soit κ cet ensemble de nœuds. Puisque le graphe des plus

courts chemins $G^{\triangleright}$ respecte l'inégalité triangulaire, il existe une arborescence optimale T_2^* qui contient exactement r, κ et X, et dont tous les nœuds de branchement sont dans κ.

Supprimons de $G^{\triangleright}$ tous les autres nœuds. Supprimons également tous les arcs sortant de r qui ne sont pas dans T_2^*. Alors T_2^* est une arborescence couvrante de poids minimal du graphe résultant.

Il est donc possible de trouver une solution optimale pour $\mathcal{I}$ en

- itérant sur tous les d-upplets κ de nœuds avec possible répétition
- itérant sur tous les arcs sortant de r, si $r \notin \kappa$
- calculant une arborescence couvrante du graphe $G^{\triangleright}$ restreint à $\{r\} \cup X \cup \kappa$, et à l'arc sortant de r choisi à l'étape précédente si $r \notin \kappa$
- comparant la solution aux solutions précédemment construites.

Il est possible de trouver une arborescence couvrante dans un graphe orienté avec un algorithme de complexité $O(m \log(n))$ [Tar77] où m et n sont respectivement les nombres d'arcs et de nœuds dans un graphe. Il existe donc un algorithme de complexité temporelle $O^*(n^d)$ pour DSTLD. □

9.1.2 Approximation polynomiale

Lemme 9.1.4. *Renvoyer l'étoile reliant la racine à tous les terminaux fournit une k-approximation pour DSTLD.*

Démonstration. La preuve est identique à celle de la k-approximation de l'algorithme des plus courts chemins pour DST : par définition du graphe des plus courts chemins, tout arc reliant la racine à un terminal x est de poids plus petit que le poids de la solution optimale car celle-ci contient un chemin de la racine vers x. L'étoile reliant la racine à tous les terminaux est donc de poids au plus k fois le poids de la solution optimale.

On peut noter que toute instance de DSTLD autorise le placement d'au moins un nœud diffusant dans le réseau : toute arborescence dans l'instance des plus courts chemins ne contenant qu'un nœud de branchement est une solution réalisable. Une étoile ne contient qu'un nœud de branchement et est donc une solution réalisable. □

9.2 Construction d'un algorithme d'approximation pour DST

Cette section décrit une technique pour construire des algorithmes d'approximation pour DST à l'aide de DSTLD.

Soit $\mathcal{I} = (G = (V, A), r, X, \omega)$ une instance de DST et $\mathcal{I}_d = (G, r, X, \omega, d)$ une instance de DSTLD. On peut remarquer que $\mathcal{I}^{\triangleright} = \mathcal{I}_d^{\triangleright}$ et que $\mathcal{I} = \mathcal{I}_{k-1}$. On va démontrer que construire une solution optimale de $\mathcal{I}_d$ permet d'approcher les solutions optimales de $\mathcal{I}$ avec un rapport $\left\lceil \frac{k-1}{d} \right\rceil$, en conséquence de quoi il est possible de construire une approximation pour le problème de Steiner. Puisque DSTLD est XP en d, c'est également le cas de cette approximation.

Il a été démontré dans la preuve du théorème 2.1.2, en page 16 du chapitre 2, que, pour DST, toute solution optimale de $\mathcal{I}^{\triangleright}$ a le même poids qu'une solution optimale de $\mathcal{I} = \mathcal{I}_{k-1}$. Ainsi, notre propriété est vérifiée pour $d = k - 1$.

Nous cherchons ici à réduire artificiellement le nombre de nœuds diffusants autorisés pour trouver une solution approchée de l'instance $\mathcal{I}$ de DST. Plus on réduit d, plus le temps

de calcul est court, mais en contre partie, le rapport d'approximation devient moins bon. Par cette technique il est possible de garantir n'importe quel rapport d'approximation pour le problème de Steiner, y compris un rapport qu'une approximation polynomiale ne peut atteindre, à condition que notre choix pour d dépende des paramètres de l'instance que l'on souhaite résoudre. En effet, si on fixe $d = \frac{k-1}{r(k)}$, alors on dispose d'une $\lceil r(k) \rceil$-approximation pour DST.

S'il est préférable de conserver un temps polynomial sans pour autant restreindre d à des valeurs constantes, il est toujours possible de résoudre l'instance $\mathcal{I}_d$ à l'aide d'un algorithme d'approximation polynomial. Si cet algorithme est de rapport $r(d,k)$, alors la solution renvoyée est une $\left(\left\lceil \frac{k-1}{d} \right\rceil \cdot r(d,k)\right)$-approximation pour le problème de Steiner. Nous y reviendrons dans une section ultérieure, dans laquelle nous étudions l'inapproximabilité de DSTLD.

Théorème 9.2.1. *Soient T^* une solution optimale pour $\mathcal{I}$ et T_d^* une solution optimale pour $\mathcal{I}_d$ alors $\frac{\omega^{\triangleright}(T_d^*)}{\omega(T^*)} \leq \left\lceil \frac{k-1}{d} \right\rceil$.*

Démonstration. En partant de la solution optimale T^*, on va construire une solution réalisable T_d de $\mathcal{I}_d$ vérifiant ce rapport. Et puisque $\omega^{\triangleright}(T_d^*) \leq \omega^{\triangleright}(T_d)$, le théorème sera prouvé.

Pour tout arc (u,v) de T^*, un plus court chemin reliant u à v est l'arc (u,v) lui-même. En effet, dans le cas contraire, on pourrait remplacer l'arc (u,v) par un plus court chemin reliant u et v pour obtenir une arborescence de poids plus petit. Donc $\omega^{\triangleright}(u,v) = \omega(u,v)$. Sélectionnons pour chaque arc (u,v) de T^* l'arc (u,v) dans $\mathcal{I}^{\triangleright}$. L'arborescence obtenue a donc le même poids que T^*. On la note également T^*. On vérifie $\omega(T^*) = \omega^{\triangleright}(T^*)$.

Pour un nœud u de T^*, soit $X(u)$ l'ensemble des terminaux contenus dans le sous-arbre de T^* enraciné en u. On cherche un nœud v de T^* tel que

- $|X(v)| \geq 1 + \left\lceil \frac{k-1}{d} \right\rceil$
- pour tout fils w de v, $|X(w)| \leq \left\lceil \frac{k-1}{d} \right\rceil$

Si un tel nœud n'existe pas, on pose $v = r$, et on a alors nécessairement $|X(r)| \leq \left\lceil \frac{k-1}{d} \right\rceil$.

On remplace le sous-arbre enraciné en v par une étoile $S(v)$ reliant v à tous les terminaux de $X(v)$. Puisque chaque arc est pondéré par le poids d'un plus court chemin dans G, le poids de chaque arc (v,x) de cette étoile ne peut excéder celui du chemin reliant v à x dans T^*. Soit $P(v,x)$ ce chemin. On vérifie alors

$$\omega^{\triangleright}(S(v)) \leq \sum_{x \in X(v)} \omega^{\triangleright}(P(v,x))$$

$$\omega^{\triangleright}(S(v)) \leq \sum_{w,\text{ fils de } v} \sum_{x \in X(w)} \omega^{\triangleright}(v,w) + \omega^{\triangleright}(P(w,x))$$

On rappelle que, par définition de v, pour tout fils w de v, $|X(w)| \leq \left\lceil \frac{k-1}{d} \right\rceil$. De plus chaque chemin $P(w,x)$ dans T^* est nécessairement de poids plus petit que l'union des chemins $\bigcup_{x \in X(w)} P(w,x)$.

$$\omega^{\triangleright}(S(v)) \leq \sum_{w,\text{ fils de } v} \left\lceil \frac{k-1}{d} \right\rceil \cdot \omega^{\triangleright}((v,w) \cup \bigcup_{x \in X(w)} P(w,x))$$

Soit $T^*(v)$ le sous-arbre de T^* enraciné en v.

$$\omega^{\triangleright}(S(v)) \leq \left\lceil \frac{k-1}{d} \right\rceil \cdot \omega^{\triangleright}(T^*(v))$$

Le nombre de nœuds de branchement dans le sous-arbre enraciné en v est réduit à 1, et le poids de ce sous-arbre n'a pu être multiplié par plus de $\lceil \frac{k-1}{d} \rceil$. On remplace temporairement cette nouvelle étoile par un terminal seul. On recommence jusqu'au moment où T^* est réduit à un seul terminal. On construit donc au plus d étoiles.

On va maintenant redéployer les terminaux en étoiles, dans l'ordre inverse de leur contraction. Ceci nous donne un arbre T_d de $\mathcal{I}_d^{\triangleright}$ possédant pas plus de d nœuds de branchement, un par étoile, c'est donc une solution réalisable de $\mathcal{I}_d$. Puisque le poids de chaque étoile est au plus $\lceil \frac{k-1}{d} \rceil$ fois celui des sous-arbres de T^* qu'elles remplacent, on a $\frac{\omega^{\triangleright}(T_d^*)}{\omega(T^*)} \leq \lceil \frac{k-1}{d} \rceil$. Donc T_d vérifie la propriété désirée, prouvant le théorème. □

Il n'y a pas de rapport d'approximation $\frac{d_2}{d_1}$ garanti entre les valeurs des solutions optimales de $\mathcal{I}_{d_1}$ et $\mathcal{I}_{d_2}$, comme le montre la figure 9.1. Ceci est dû au fait qu'il faut toujours partager les terminaux entre les nœuds que l'on choisit comme diffusants, et ce partage démultiplie le poids par un facteur qui dépend de k. Il n'est donc pas toujours possible de réduire le nombre de nœuds diffusants sans tenir compte du nombre de terminaux.

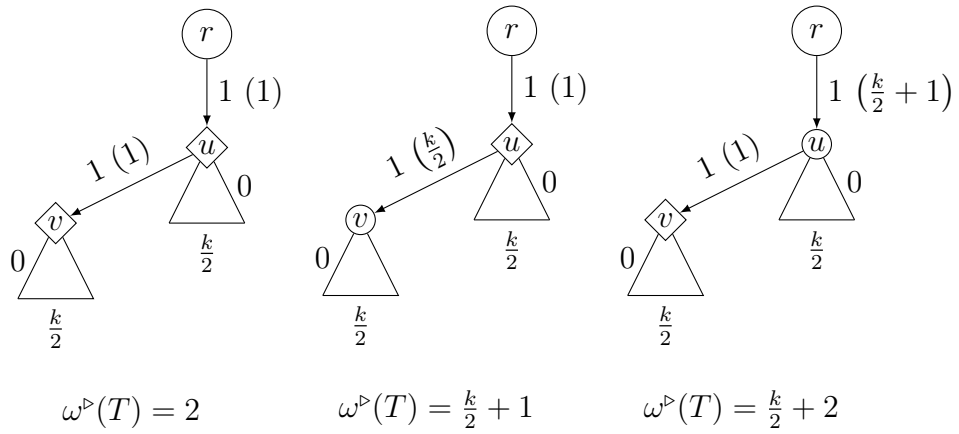


Figure 9.1 – *Nous avons représenté une arborescence T avec 2 nœuds de branchement. Le poids des arcs (r, u) et (u, v) vaut 1, les autres arcs sont de poids nul. Cette arborescence constitue notre instance $\mathcal{I}$. À gauche, on a représenté une solution réalisable de $\mathcal{I} = \mathcal{I}_2$. Au centre et à droite, on a représenté deux solutions réalisables de l'instance $\mathcal{I}_1$. On a mentionné entre parenthèses la charge des arcs (c'est-à-dire le nombre de fois qu'il faut payer le poids de ces arcs). Une solution optimale pour $\mathcal{I} = \mathcal{I}_2$ est le graphe tout entier. On pourrait penser en généralisant le théorème 9.2.1 qu'en retirant un nœud diffusant le poids de la solution optimale serait multiplié par 2. Cet exemple montre que, quel que soit le nœud diffusant choisi, alors le poids de la solution est multiplié par $\frac{k}{4}$. Il est aisé de construire à partir de là un exemple pour tout paramètre d_1 et d_2.*

Dans le chapitre 2, le lemme 2.2.4 en page 19 fait mention d'une technique similaire, consistant à contraindre la hauteur des solutions renvoyées. Chercher une solution de hauteur fixée égale à l et de poids optimal parmi les solutions de hauteur l permet de construire une $O(k^{\frac{1}{l}})$-approximation pour le problème de Steiner non contraint. La restriction par les nœuds diffusants que nous faisons dans cette section est donc moins efficace que la restriction par la hauteur des solutions, au sens où il faut autoriser beaucoup de nœuds diffusants dans la solution là où il n'est nécessaire que d'autoriser une faible hauteur de solution réalisable pour obtenir une solution de qualité équivalente. Cependant, comme nous l'avons fait remarqué en page 19, il est NP-difficile de construire une arborescence profondeur au plus l de poids

minimum (parmi toutes les solutions de profondeur au plus l) si $l \geq 2$ alors que restreindre le nombre de nœuds diffusant peut rendre le problème de Steiner polynomial puisque DSTLD est XP vis-à-vis du paramètre d.

Le résultat de cette section montre qu'en disposant d'un algorithme d'approximation de rapport $r(d,k)$ pour DSTLD, on dispose d'un algorithme d'approximation de rapport $\left(\left\lceil \frac{k-1}{d} \right\rceil \cdot r(d,k)\right)$ pour DST. Dans la section suivante, nous démontrons un résultat d'inapproximabilité pour DSTLD.

9.3 Résultat d'inapproximabilité pour DSTLD

Cette section est dédiée à la preuve du théorème suivant.

Théorème 9.3.1. *Si $NP \not\subset TIME(n^{O(\log\log(n))})$, pour toute constante $\varepsilon > 0$, il n'existe pas d'algorithme d'approximation pour DSTLD de rapport d'approximation meilleur que $1 + (\frac{1}{e} - \varepsilon) \cdot \frac{k}{d-1}$, où e est la constante d'Euler.*

Pour démontrer ce résultat, nous allons décrire une réduction depuis le problème de couverture maximum par ensembles [2].

Problème : Problème de couverture maximum par ensembles (max-SC)

Instance :
- Un univers X.
- Un sous-ensemble $\mathcal{S}$ des parties de X, appelés les *ensembles*.
- Un entier $d \in \mathbb{N}^*$.

Solution réalisable : Une couverture $C \in \mathcal{S}$ ne contenant pas plus de d ensembles.

Optimisation : Si $X(C)$ est l'ensemble des éléments couverts par C, maximiser $|X(C)|$.

Théorème 9.3.2 ([Fei98], propriété 5.2)**.** *Si $NP \not\subset TIME(n^{O(\log\log(n))})$, alors pour toute constante $\epsilon > 0$, le problème max-SC ne peut être approché polynomialement avec rapport $1 - \frac{1}{e} + \epsilon$, même si on se restreint aux instances où toute solution optimale couvre tout l'univers avec exactement d ensembles.*

Après avoir décrit la réduction, nous montrerons qu'il est possible d'associer à chaque solution réalisable de DSTLD une solution réalisable de max-SC et inversement. Nous démontrerons qu'il existe un lien entre les poids de ces solutions. Ce lien permettra de mettre en défaut le théorème 9.3.2 si nous supposons qu'une approximation de rapport $1+(\frac{1}{e}-\varepsilon)\cdot\frac{k}{d-1}$ existe pour DSTLD.

Réduction. Soit $\mathcal{I} = (X, \mathcal{S}, d-1)$ une instance de mac-SC dont toute solution optimale couvre l'univers avec exactement $d-1$ ensembles.

Dans l'instance $\mathcal{I}$, on ne considérera que les solutions réalisables maximales, c'est-à-dire celles qui utilisent exactement $d-1$ ensembles. Puisque toute solution optimale utilise

2. *Maximum Coverage*, en anglais

exactement $d-1$ ensembles, il est possible de compléter toute solution non maximale pour obtenir une solution strictement meilleure.

Nous allons définir une instance $\mathcal{I}' = (G = (V, A), r, X, \omega, d)$ de DSTLD.

L'ensemble V des nœuds est constitué d'une racine r, d'un nœud par ensemble s de $\mathcal{S}$ et d'un terminal par élément x de X. Pour plus de simplicité dans la démonstration, nous identifierons chaque ensemble s avec le nœud qui lui correspond dans G (on parlera du nœud-ensemble s), et chaque élément x de l'univers avec le terminal qui lui correspond (on parlera du terminal-élément x).

Soit $B > 0$ une constante que nous choisirons ultérieurement. La racine est reliée à tout nœud-ensemble s par un arc (r, s) de poids B et à tout terminal-élément x par un arc (r, x) de poids $B + 1$. Enfin, chaque nœud-ensemble est relié aux terminaux-éléments qu'il contient dans l'instance $\mathcal{I}$ par un arc de poids 1. Un exemple de réduction est donné en figure 9.2.

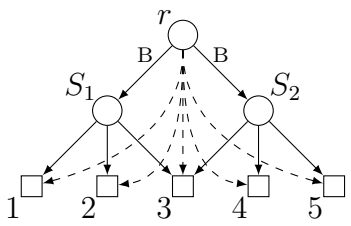


FIGURE 9.2 – *Un exemple de réduction avec l'instance $\mathcal{I}$ telle que $X = \{1, 2, 3, 4, 5\}$ et $\mathcal{S} = \{S_1 = \{1, 2, 3\}, S_2 = \{3, 4, 5\}\}$. Chacun des arcs en pointillés a un poids égal à $B + 1$.*

On peut remarquer que
- tout arc a un poids égal au poids du plus court chemin reliant ses extrémités dans G,
- il n'existe pas d'arc entre deux nœuds u et v si et seulement s'il n'existe pas de chemin dans G entre ces deux nœuds.

En conséquence de quoi, pour construire $G^{\triangleright}$, il suffit de compléter G avec des arcs de poids infini. Ainsi, on peut considérer toute arborescence de G ne contenant pas plus de d nœuds de branchement est une solution réalisable de $\mathcal{I}'$.

Construire une solution réalisable de $\mathcal{I}'$ de poids fini à partir d'une solution réalisable de $\mathcal{I}$. Nous allons maintenant démontrer qu'il est possible d'associer à toute solution réalisable de $\mathcal{I}$ une solution réalisable de $\mathcal{I}'$ dont on maîtrisera le poids. Un exemple d'une telle transformation est donné en figure 9.3.

Définition 6. Soit une solution réalisable maximale C de $\mathcal{I}$, alors on définit la solution $\mathcal{T}(C)$ de $\mathcal{I}'$ associée à C ainsi :
- on sélectionne la racine r ;
- on relie r à tous les nœuds-ensembles de C ;
- on relie à chaque terminal-élément couvert par C un des nœuds-ensembles de C qui le contient ;
- on relie directement r à tous les autres terminaux.

$\mathcal{T}(C)$ est une arborescence couvrant tous les terminaux de X et d nœuds de branchement : la racine et $d-1$ nœuds-ensembles. C'est donc une solution réalisable de $\mathcal{I}'$.

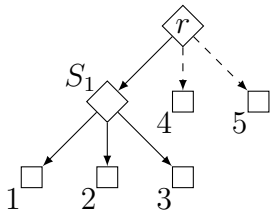


FIGURE 9.3 – $\mathcal{T}(\{1,2,3\})$ *à partir de l'instance de la figure 9.2 et* $d-1=1$. *Le poids de cet arbre est* $B+3+2(B+1)$.

On rappelle que $X(C)$ est l'ensemble des terminaux couverts par C. Le poids de $\mathcal{T}(C)$ est

$$\begin{aligned}\omega^{\triangleright}(\mathcal{T}(C)) &= (B+1)\cdot(k-|X(C)|)+B\cdot(d-1)+|X(C)|\\ \omega^{\triangleright}(\mathcal{T}(C)) &= B\cdot(d-1+k-|X(C)|)+k \qquad (9.1)\end{aligned}$$

Puisque C^* couvre exactement k terminaux avec $d-1$ ensembles, alors $\omega^{\triangleright}(\mathcal{T}(C^*)) = B\cdot(d-1)+k$.

Construire une solution réalisable de $\mathcal{I}$ à partir d'une solution réalisable de $\mathcal{I}'$ de poids fini. Nous allons démontrer, à l'inverse de la construction précédente, qu'il est possible d'associer à toute solution réalisable de $\mathcal{I}'$ de poids fini une solution réalisable de $\mathcal{I}$ dont on maîtrisera le poids.

Soit T une solution réalisable de $\mathcal{I}'$ de poids fini. Puisque $d \geq 2$, l'arborescence T ne contient pas plus de $d-1$ nœuds-ensembles qui soient nœuds de branchement. En effet, si exactement d nœuds-ensembles étaient nœuds de branchement, alors r ne pourrait être nœud de branchement. Donc r n'a qu'un seul successeur dans T. Puisqu'aucun nœud-ensemble n'a d'autre prédécesseur que r dans T et que T contient au moins d nœuds-ensembles, alors T est de poids infini, ce qui est exclu.

Définition 7. Soit T une solution réalisable de $\mathcal{I}'$ de poids fini, alors on définit la solution $\mathcal{C}(T)$ de $\mathcal{I}$ associée à T en choisissant tout nœud-ensemble qui est nœuds de branchement dans T. Si $\mathcal{C}(T)$ contient moins de $d-1$ ensembles, alors on complète $\mathcal{C}(T)$ de façon gloutonne, en ajoutant des ensembles qui couvrent de nouveaux terminaux.

Il y a au plus $d-1$ nœuds-ensembles qui sont nœuds de branchement dans T donc $\mathcal{C}(T)$ est une solution réalisable de $\mathcal{I}$. Puisque toute solution optimale de $\mathcal{I}$ contient $d-1$ ensembles, il est toujours possible de compléter une solution qui n'est pas maximale avec un ensemble couvrant des terminaux qui ne sont pas déjà couverts.

Lemme 9.3.3. *Soit T une solution réalisable de $\mathcal{I}'$ de poids fini, alors $\omega^{\triangleright}(\mathcal{T}(\mathcal{C}(T))) \leq \omega^{\triangleright}(T)$.*

Démonstration. Soient $S = \{s_1, s_2, \ldots, s_p\}$ les nœuds-ensembles de T qui sont nœuds de branchement, et X_S les éléments-terminaux couverts par S dans T (éventuellement, dans l'instance $\mathcal{I}$, la couverture S couvre plus d'éléments). Le poids de T est donc $\omega^{\triangleright}(T) = B\cdot(p+k-|X_S|)+k$. D'après l'équation (9.1), $\omega^{\triangleright}(\mathcal{T}(\mathcal{C}(T))) = B\cdot(d-1+k-|X(\mathcal{C}(T))|)+k$. On rappelle que $B>0$. On veut montrer que $\omega^{\triangleright}(T)-\omega^{\triangleright}(\mathcal{T}(\mathcal{C}(T))) \geq 0$, autrement dit :

$$\begin{aligned}B\cdot(p-|X_S|-(d-1)+|X(\mathcal{C}(T))|) &\geq 0\\ |X(\mathcal{C}(T))|-|X_S| &\geq (d-1)-p\end{aligned}$$

Chaque ensemble dans $\mathcal{C}(T)$ mais pas dans S a été ajouté pour obtenir une solution maximale. Tout ensemble ajouté de cette manière dans $\mathcal{C}(T)$ couvre au moins un nouvel élément, car toute solution optimale contient $d-1$ ensembles. Donc le nombre de terminaux dans $X(\mathcal{C}(T))$ mais pas dans X_S vaut au moins $(d-1)-p$. □

Lien entre les solutions optimales. Nous allons maintenant montrer que les deux constructions précédentes permettent de construire une solution optimale de $\mathcal{I}$ à partir d'une solution optimale de $\mathcal{I}'$ et inversement.

Lemme 9.3.4. *Si C^* est une solution optimale pour $\mathcal{I}$, alors $\mathcal{T}(C^*)$ est une solution optimale pour $\mathcal{I}'$.*

Démonstration. Supposons qu'il existe une solution T de poids strictement inférieur à $\mathcal{T}(C^*)$. D'après le lemme 9.3.3, $\omega^{\triangleright}(\mathcal{T}(\mathcal{C}(T))) \leq \omega^{\triangleright}(T) < \omega(\mathcal{T}(C^*))$. Donc, d'après l'équation 9.1 :

$$B \cdot (d-1+k-|X(\mathcal{C}(T))|) + k < B \cdot (d-1) + k$$
$$B \cdot (k-|X(\mathcal{C}(T))|) < 0$$

Or $B > 0$ donc :

$$k < |X(\mathcal{C}(T))|$$

Ce qui est exclu. Donc $\mathcal{T}(C^*)$ est une solution optimale pour $\mathcal{I}'$. □

Lemme 9.3.5. *Si T^* est une solution optimale pour $\mathcal{I}'$ alors $\mathcal{C}(T^*)$ est une solution optimale pour $\mathcal{I}$.*

Démonstration. T^* est une solution optimale. D'après le lemme 9.3.3, $\omega^{\triangleright}(\mathcal{T}(\mathcal{C}(T^*))) \leq \omega^{\triangleright}(T^*)$. Donc $\mathcal{T}(\mathcal{C}(T^*))$ est également une solution optimale : $\omega^{\triangleright}(\mathcal{T}(\mathcal{C}(T^*))) = \omega^{\triangleright}(T^*)$.

Supposons qu'il existe une solution C couvrant strictement plus de terminaux-éléments que $\mathcal{C}(T^*)$. On rappelle que $\mathcal{C}(T^*)$ est maximale. On suppose que C l'est aussi, sinon on pourrait compléter C pour trouver une solution réalisable couvrant plus de terminaux-éléments que C. D'après l'équation 9.1 :

$$\omega^{\triangleright}(\mathcal{T}(C)) = B \cdot (d-1+k-|X(C)|) + k$$
$$\omega^{\triangleright}(\mathcal{T}(\mathcal{C}(T^*))) = B \cdot (d-1+k-|X(\mathcal{C}(T^*))|) + k$$
$$\Rightarrow \quad \omega^{\triangleright}(\mathcal{T}(C)) - \omega^{\triangleright}(\mathcal{T}(\mathcal{C}(T^*))) = B \cdot (|X(\mathcal{C}(T^*)| - |X(C)|)$$
$$\Rightarrow \quad \omega^{\triangleright}(\mathcal{T}(C)) - \omega^{\triangleright}(\mathcal{T}(\mathcal{C}(T^*))) < 0$$
$$\Rightarrow \quad \omega^{\triangleright}(\mathcal{T}(C)) < \omega^{\triangleright}(\mathcal{T}(\mathcal{C}(T^*)))$$
$$\Rightarrow \quad \omega^{\triangleright}(\mathcal{T}(C)) < \omega^{\triangleright}(T^*)$$

Ce qui est exclu donc $\mathcal{C}(T^*)$ est une solution optimale pour $\mathcal{I}$. □

Démonstration du théorème 9.3.1. Nous allons démontrer que, dans la réduction précédente, s'il existe une approximation polynomiale de rapport $1 + (\frac{1}{e} - \varepsilon') \cdot \frac{k}{d-1}$ pour DSTLD alors il en existe une de rapport $(1 - \frac{1}{e} + \varepsilon)$ pour max-SC, où ε et ε' sont deux constantes réelles strictement positives.

Démonstration. Soit C^* une solution optimale de l'instance $\mathcal{I}$. Soit $\varepsilon' > 0$, supposons qu'il existe un algorithme d'approximation de rapport $\rho = 1 + (\frac{1}{e} - \varepsilon') \cdot \frac{k}{d-1}$ pour DSTLD. Alors, pour l'instance $\mathcal{I}'$, cet algorithme renvoie, d'après le lemme 9.3.4, une solution T de poids au plus $\rho \cdot \omega^{\triangleright}(\mathcal{T}(C^*))$.

$$\frac{\omega^{\triangleright}(T)}{\omega^{\triangleright}(\mathcal{T}(C^*))} \leq \rho$$

D'après le lemme 9.3.3 :

$$\Rightarrow \qquad \frac{\omega^{\triangleright}(\mathcal{T}(\mathcal{C}(T)))}{\omega^{\triangleright}(\mathcal{T}(C^*))} \leq \rho$$

D'après l'équation 9.1 :

$$\begin{aligned}
\Leftrightarrow \quad & \frac{B(d-1+k-|X(\mathcal{C}(T))|)+k}{B\cdot(d-1)+k} \leq \rho \\
\Leftrightarrow \quad & d-1+k-|X(\mathcal{C}(T))| + \frac{k}{B} \leq (d-1+\frac{k}{B})\cdot\rho \\
\Leftrightarrow \quad & d-1+k-|X(\mathcal{C}(T))| + \frac{k}{B} \leq (d-1)\cdot(1+(\frac{1}{e}-\varepsilon')\cdot\frac{k}{d-1}) + \frac{k}{B}\cdot\rho \\
\Leftrightarrow \quad & k\cdot(1-\frac{1}{e}+\varepsilon'+\frac{1-\rho}{B}) \leq |X(\mathcal{C}(T))|
\end{aligned}$$

C^* est une solution optimale de l'instance $\mathcal{I}$. Elle couvre dons tous les terminaux : $|X(C^*)| = k$.

$$\Leftrightarrow \qquad |X(C^*)|\cdot(1-\frac{1}{e}+\varepsilon'+\frac{1-\rho}{B}) \leq |X(\mathcal{C}(T))|$$

Posons désormais $B = 2 \cdot \frac{\rho-1}{\varepsilon'} > 0$. Alors $1 - \frac{1}{e} + \frac{\varepsilon'}{2} \leq \frac{X(C(T))}{X(C^*)}$. Et cela contredit donc le théorème 9.3.2, en posant $\varepsilon = \frac{\varepsilon'}{2}$, ce qui prouve le présent théorème. □

9.3.1 Conséquence d'un tel résulat

Dans la section 9.2, nous avons établi qu'il est possible de construire un algorithme d'approximation polynomial de rapport $r \cdot \lceil \frac{k-1}{d} \rceil$ pour DST si on dispose d'un algorithme d'approximation de rapport r pour DSTLD.

Ce fort résultat d'inapproximabilité démontré dans la présente section a pour conséquence l'impossibilité de construire, pour DST, une approximation de rapport meilleur que $O((\lceil \frac{k-1}{d} \rceil)^2)$ par cette technique.

Cependant, il faut quand même noter que, s'il existe une approximation de rapport $O(\frac{k}{d})$ pour DSTLD dont la complexité en temps dépendant de d reste raisonnable, il est alors envisageable de construire une approximation de rapport $r(k)$ pour DST en posant $d = \frac{k}{\sqrt{(r(k))}}$.

9.4 Synthèse des résultats

Dans ce chapitre, nous avons établie des propriétés du problème DSTLD vis-à-vis de la complexité paramétrée et de l'approximabilité polynomiale :

- le problème DSTLD n'est pas XP vis-à-vis du paramètre $k - d$;
- le problème DSTLD est XP et W[2]-difficile vis-à-vis du paramètre d ;
- à moins que $NP \subset TIME(n^{O(\log\log(n))})$, pour toute constante $\varepsilon > 0$, il n'existe pas d'algorithme d'approximation pour DSTLD de rapport d'approximation meilleur que $1 + (\frac{1}{e} - \varepsilon) \cdot \frac{k}{d-1}$.

Nous avons enfin démontré un résultat d'approximabilité pour le problème de Steiner, en utilisant le problème DSTLD pour construire un algorithme d'approximation XP vis-à-vis du paramètre d : soit $\mathcal{I} = (G = (V, A), r, X, \omega)$ une instance de DST, à partir d'une solution optimale de l'instance $\mathcal{I}_d = (G = (V, A), r, X, \omega, d)$ de DSTLD, il est possible de construire une solution réalisable de $\mathcal{I}$ approchant la solution optimale avec un rapport $\lceil \frac{k-1}{d} \rceil$.

Les résultats de ce chapitre ont été publiés à la conférence COCOON 2014 [WWBB14].

Chapitre 10.

Algorithmes paramétrés basés sur une énumération de patrons

Ce chapitre revient sur les problèmes USTLD, DSTLD, USTLB et DSTLB. Il donne une technique d'énumération de structures appelées *patrons* permettant d'accélérer le temps de recherche d'une solution réalisable ou optimale.

Ce chapitre se découpe en trois sections. La première section décrit un algorithme FPT en k et d pour DSTLD et USTLD. Nous observerons en parallèle que cet algorithme résout également DST et UST et vient compléter le tableau 10.1, qui indique les résultats connus liés à la complexité paramétrée (temporelle et spatiale) de ces problèmes vis-à-vis du paramètre kn, décrits en partie dans le chapitre 10. Les points qui nous intéressent sont les suivants :

- complexité temporelle FPT vis-à-vis du paramètre k,
- complexité spatiale polynomiale,
- construction et renvoi d'une solution optimale.

Complexité temporelle	Complexité spatiale	Renvoie $\omega(T^*)$	Renvoie T^*	Référence
$O^*(3^k)$	$O^*(2^k)$	OUI	OUI	[DW71]
$O^*((2+\varepsilon)^k)$, $\varepsilon > 0$	$O^*(2^k)$	OUI	OUI	[FKM$^+$07]
$O^*(2^k)$	$O^*(2^k)$	OUI	OUI	[BHKK07]
$O^*(5.96^k n^{O(\log(k))}$	P	OUI	OUI	[FGK08]
$O^*(2^k)$	P	OUI	NON [1]	[Ned09]
$O^*(2^k)$	P	OUI	OUI [1]	[Ned09]
$O^*(1 \cdot 3 \cdot 5 \cdots (2k-3))$	P	OUI	OUI	Théorème 10.1.8

Table 10.1 – *Vue d'ensemble des algorithmes paramétrés connus pour le problème de l'arbre de Steiner. On distingue les algorithmes ne renvoyant que la valeur d'une solution optimale T^* et ceux construisant et renvoyant une telle solution. Tous ces algorithmes permettent, avec les mêmes complexités temporelle et spatiale, de résoudre le problème de l'arborescence de Steiner.*

1. L'algorithme de Nederlof [Ned09], comme précisé dans le chapitre 1, ne peut à lui seul fournir une solution optimale, mais peut être répété pour en construire une.

Notre algorithme partage, avec l'algorithme de Nederlof [Ned09], la vérification des trois propriété. Il dispose en outre de deux avantages comparé à cet algorithme : premièrement, son temps de calcul ne dépend pas linéairement du poids des arcs de l'instance, et secondement, il conçoit au fur et à mesure des solutions réalisables et conserve celle de plus petit poids, donc il est possible de stopper l'algorithme et de renvoyer cette solution avant la fin du calcul.

Les deux autres sections sont dédiées à deux variantes de cet algorithme spécifiquement créées pour les problèmes DSTLB et USTLB. La première de ces deux variantes porte sur les paramètres k et n_s (le nombre de nœuds non terminaux dans la solution optimale). Nous montrons alors qu'il existe un algorithme FPT probabiliste pour résoudre ces instances de manière optimale. Enfin, la seconde variante s'intéresse aux paramètres k et p. Ce choix de paramètres est moins restrictif que fixer k et n_s. Cependant, il permet tout de même de construire un algorithme XP renvoyant une solution réalisable dans les cas où l'instance est non orientée ou orientée et planaire. De plus, il permet de construire un algorithme XP renvoyant une solution optimale dans le cas où l'instance est non orientée et est de largeur d'arbre bornée.

10.1 Algorithme exact FPT en k et d pour DSTLD

Nous allons dans cette section définir les bases de l'algorithme d'énumération de patrons. Il s'agit d'un algorithme pour DSTLD qui s'exécute en espace polynomial et en temps FPT vis-à-vis des paramètres k et d, et qui construit une solution optimale. La complexité temporelle de cet algorithme est $O(t_d \cdot d^k \cdot n \cdot (d+k))$ où t_d est égal au nombre d'arborescences enracinées contenant d nœuds non étiquetés. Ces arbres sont non planaires (ou *libres*), c'est-à-dire qu'il n'existe pas de structure ordonnant les fils d'un nœud dans ces arbres. Ainsi, le début de la suite t_d est le suivant : $1, 1, 2, 4, 9, 20, 48, 115, 286, 719, 1842, 4766, 12486, \ldots$. Cette séquence est celle d'identifiant A000081 sur le site des séquences d'entiers en ligne [2]. Elle est également présentée dans [FS09].

Puisque DSTLD est une généralisation de DST, cet algorithme peut également être utilisé pour résoudre ce dernier en temps FPT en k, tout en utilisant un espace polynomial, et en construisant puis en renvoyant une solution optimale.

Nous ne présentons dans cette section que la version orientée, mais cet algorithme est aussi valide pour le problème USTLD (les démonstrations sont similaires).

10.1.1 Description de l'algorithme

Soit $\mathcal{I} = (G = (V, A), r, X, \omega, d)$ une instance de DSTLD. Nous allons définir un ensemble de *patrons* et un étiquetage de ces patrons. Chaque patron étiqueté correspondra à une solution réalisable de telle sorte que la solution optimale soit en relation avec au moins un patron. Nous rechercherons, pour chaque patron, un étiquetage de poids optimal.

Les deux propriétés des patrons suivantes garantissent que notre algorithme est FPT vis-à-vis de k et d :

- leur nombre est FPT en k et d,
- trouver un étiquetage de poids optimal d'un patron peut se faire en temps polynomial.

2. `https://oeis.org/A000081`

Définition et propriétés des patrons

Définition 8. Un *patron* pour l'instance $\mathcal{I}$ est une arborescence enracinée en r contenant k feuilles étiquetées, chacune avec un unique terminal de X, et au plus d autres nœuds internes de branchement non étiquetés. La racine n'a qu'un seul fils (ainsi, un patron ne contient aucun nœud de degré deux). Les fils d'un même nœud ne sont pas ordonnés : il est possible de les inverser (avec leurs descendants) sans altérer le patron.

La figure 10.2 donne un exemple de patrons. Nous décrivons maintenant l'étiquetage d'un patron et la solution réalisable qui en résulte.

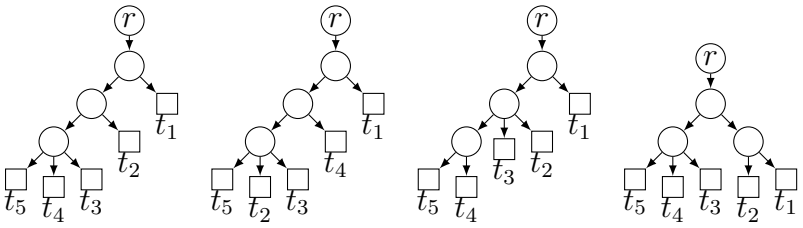


Figure 10.2 – *Quatre patrons distincts parmi les 120 existants pour une instance vérifiant $d = 3$ et $k = 5$. Si, dans le premier patron, on échange t_2 et t_4, on obtient le deuxième patron, mais, si on échange t_3 et t_4, le patron ne change pas car ils sont fils d'un même nœud.*

Définition 9. Soit P un patron de $\mathcal{I}$ et I_P ses nœuds internes non étiquetés. Un *étiquetage* est une fonction κ associant à chaque nœud de I_P un nœud de V et associant la racine et les terminaux à eux-mêmes.

Définition 10. Soit P un patron de $\mathcal{I}$ et κ un étiquetage de P. La solution réalisable $\kappa(P)$ associée à P est, dans le graphe des plus courts chemins $G^{\triangleright}$, l'ensemble des arcs $(\kappa(v_1), \kappa(v_2))$ pour tout arc (v_1, v_2) de P.

Remarque 18. $\kappa(P)$ peut éventuellement inclure des cycles. Dans ce cas, il est possible en temps polynomial de trouver dans $G^{\triangleright}$ une arborescence de poids plus petit qui soit incluse dans $\kappa(P)$.

Remarque 19. Un étiquetage autorise deux nœuds distincts du patrons à être étiquetés par le même nœud, voire par un terminal ou par r. Si deux extrémités d'un même arc (v_1, v_2) sont étiquetés par le même nœud, alors l'arc $(\kappa(v_1), \kappa(v_2))$ est tout simplement ignoré dans la définition 10.

Lemme 10.1.1. *Il existe au plus $t_d \cdot d^k$ patrons pour l'instance $\mathcal{I}$ où t_d est le nombre d'arborescences dont les fils de chaque nœud ne sont pas ordonnés et contenant d nœuds internes non étiquetés.*

Démonstration. Un patron pour $\mathcal{I}$ est une arborescence enracinée en r, dont les fils de chaque nœud ne sont pas ordonnés, et contenant d nœuds internes non étiquetés et k feuilles, chacune étiquetée par un unique terminal.

Donc il est possible de construire un ensemble contenant tous les patrons (ainsi que d'autres arbres) en énumérant toutes les arborescences dont les fils de chaque nœud ne sont pas ordonnés et contenant d nœuds non étiquetés, puis en insérant les k feuilles étiquetées.

Le nombre de tels arbres s'élève à $t_d \cdot d^k$. Il existe moins de patrons car ceux-ci n'ont pas de nœud non étiqueté avec moins de 2 fils. □

Dans le cas du problème DST, le nombre de nœuds diffusants autorisé est $d = k-1$. Dans ce cas particulier, il est possible de compter précisément le nombre de patrons.

Lemme 10.1.2. *Il existe* $1 \cdot 3 \cdot 5 \cdots 2k-3$ *patrons pour* $\mathcal{I}$ *si* $d = k-1$.

Démonstration. Si $d = k-1$, il existe autant de patrons que d'arbres binaires enracinés contenant k feuilles étiquetées (un patron est un tel arbre, complété avec une racine r ayant un seul fils). Il est possible d'énumérer ces arbres en adaptant l'algorithme de [RŐ5] : soit un patron pour une instance avec $k-1$ feuilles, il est possible de construire un unique patron à k feuilles en choisissant un nœud v, terminal ou nœud interne mais différent de la racine. On insère entre v et son père un nœud v'. On ajoute le dernier terminal en tant que fils de v'. Il y a exactement $2k-3$ choix pour le nœud v : s'il existe $f(k)$ patrons à k feuilles, alors $f(k) = f(k-1) \cdot (2k-3)$. Puisque $f(1) = f(2) = 1$, alors $f(k) = 1 \cdot 3 \cdot 5 \cdots 2k-3$. □

Une autre démonstration de ce résultat est disponible dans [FS09, page 129].

Programmation dynamique avec les patrons. Soit P un patron pour l'instance $\mathcal{I}$. On recherche l'étiquetage κ pour lequel $\kappa(P)$ a un poids minimum. On sait qu'il existe au moins une solution de poids fini qui consiste à étiqueter tous les nœuds avec r, renvoyant ainsi la k-approximation fournie par l'union des k plus courts chemins. Donc il existe une solution de poids minimum finie.

On associe à chaque nœud v de P deux tableaux :

- C_v est un tableau de taille n contenant pour chaque nœud $u \in V$ le poids minimum d'une solution induite par un étiquetage du sous-patron de P enraciné en v si $\kappa(v) = u$,
- D_v est une matrice de taille $n \cdot |ch(v)|$, où $ch(v)$ sont les fils de v dans P, contenant pour chaque nœud $u \in V$ et chaque fils $w \in ch(v)$ l'étiquette $\kappa(w)$ de w si v est étiqueté avec u et si le reste de l'étiquetage du sous-patron enraciné en v induit une solution de poids minimal.

Pour tout terminal t, on initialise le tableau C_t avec $C_t[t] = 0$ et $C_t[u] = +\infty$ pour $u \neq t$. Le tableau D_t est vide car tout terminal est une feuille du patron et n'a donc pas de fils.

Pour tout nœud interne v de P (y compris la racine), on calcule D_v et C_v pour tout nœud $u \in V$ et tout fils $w \in ch(v)$ avec les formules suivantes :

$$D_v[u,w] = \arg\min_{x \in V}(\omega^{\triangleright}(u,x) + C_w[x]) \tag{10.1}$$

$$C_v[u] = \sum_{w \in ch(v)} \min_{x \in V}(\omega^{\triangleright}(u,x) + C_w[x]) \tag{10.2}$$

La procédure d'étiquetage. La *procédure d'étiquetage* construit récursivement un étiquetage κ pour un patron P à partir des tableaux D_v. Connaissant un nœud v de P et un nœud u de V, la procédure d'étiquetage construit l'étiquetage du sous-patron de P enraciné en v comme suit. Elle affecte $\kappa(v) = u$ et, pour chaque fils $w \in ch(v)$, applique la procédure d'étiquetage avec, comme paramètres, w et $D_v[u,w]$. En appelant la procédure d'étiquetage avec r en tant que racine de P et étiquette d'elle-même, on construit un étiquetage pour P de façon récursive.

Algorithme pour trouver une solution optimale $\kappa(P)$. On utilise l'algorithme 9 pour renvoyer une solution optimale pour $\mathcal{I}$.

Algorithme 9 Algorithme d'énumération des patrons

1: Initialiser T^0 avec une solution réalisable quelconque.
2: **Pour** chaque patron P pour $\mathcal{I}$ **Faire**
3: Construire les tableaux C_v et D_v, pour tout nœud v de P dans l'ordre inverse d'un parcours en largeur
4: Utiliser la procédure d'étiquetage pour construire un étiquetage κ de P
5: $T \leftarrow \kappa(P)$
6: **Si** $\omega^{\triangleright}(T) < \omega^{\triangleright}(T^0)$ **Alors**
7: $T^0 \leftarrow T$
8: **Renvoyer** T^0

10.1.2 Validité de l'algorithme

La preuve de la validité de l'algorithme 9 se fait en trois temps : nous prouvons dans un premier temps qu'il existe un patron P^* et un étiquetage κ^* pour lesquels $\kappa^*(P^*)$ est une solution optimale de $\mathcal{I}$. Puis nous démontrons que, lorsque la variable P dans l'algorithme 9 prend la valeur P^*, la solution réalisable construite à la ligne 5 est de poids $C_r[r]$. Enfin, nous montrons que ce poids est optimal.

Lemme 10.1.3. *Il existe un patron P^* pour $\mathcal{I}$ et un étiquetage κ^* de ce patron pour lequel $\kappa^*(P^*)$ est une solution optimale de $\mathcal{I}$.*

Démonstration. Soit T^* une solution optimale de $\mathcal{I}$. Nous allons construire un patron étiqueté à partir de T^*. Cette solution contient déjà k feuilles terminales, et aucune autre feuille non terminale.

Si la racine de T^* n'a pas qu'un seul fils, on la duplique. Le nouveau nœud devient la racine de T^*, relié à l'ancien par un arc de poids nul.

Si T^* contient un nœud interne qui n'est pas un nœud de branchement, alors on contracte ce nœud. Puisque la fonction de poids $\omega^{\triangleright}$ vérifie l'inégalité triangulaire, le poids de la solution ne peut que diminuer. Cependant, puisque la solution est optimale, il n'est nullement modifié.

Si T^* contient strictement moins de $d \leq k-1$ nœuds internes de branchement, il existe un nœud v avec au moins 3 fils. On va dupliquer v en un nœud v' et relier v à v' avec un arc de poids nul. On choisit ensuite 2 fils quelconques de v qu'on déplace en tant que fils de v'. On recommence jusqu'à ce que T^* contienne d nœuds internes de branchement.

L'arbre résultant peut être défini comme un patron P^* et un étiquetage κ tels que $\kappa^*(P^*) = T^*$. □

On suppose qu'on a maintenant construit les tableaux C_v et D_v pour tout nœud du patron P^* à la ligne 3 de l'algorithme. On montre premièrement que la procédure d'étiquetage renvoie une solution de poids $C_r[r]$. Soit un nœud v de P^*, alors P^*_v est le sous-patron de P^* enraciné en v.

Lemme 10.1.4. *Si $C_v[u]$ n'est pas infini, alors la procédure d'étiquetage appelée sur le nœud v et l'étiquette u construit un étiquetage κ de P_v^* tel que le poids de $\kappa(P_v^*)$ est $C_v[u]$.*

Démonstration. On prouve ce lemme par récurrence sur la hauteur de v dans P^* (*i.e.* la longueur, en nombre d'arcs, du plus long chemin séparant v d'une feuille de P_v^*).

Si t est un nœud de hauteur 0, il s'agit d'une feuille (et d'un terminal), donc seule la valeur $C_t[t]$ est de poids fini : $C_t[t] = 0$. Le poids de $\kappa(P_t^*)$ est 0 puisque cette solution n'a pas d'arc.

Si v est maintenant un nœud interne, soit $u \in V$ tel que $C_v[u]$ est fini. D'après l'équation (10.2), $C_v[u] = \sum_{w \in ch(v)} \min(\omega^{\triangleright}(u, x) + C_w[x], x \in V)$. Puisque $C_v[u]$ est fini, pour tout $w \in ch(v)$, la valeur $C_w[x]$ choisie dans la formule précédente est finie. D'après l'équation (10.1), le nœud x choisi est $D_v[u, w]$. Donc $C_v[u] = \sum_{w \in ch(v)} (\omega^{\triangleright}(u, D_v[u, w]) + C_w[D_v[u, w]])$.

Par hypothèse de récurrence, si on appelle la procédure d'étiquetage avec le nœud w et l'étiquette $D_v[u, w]$, on construit un étiquetage κ de P_w^* tel que le poids de $\kappa(P_w^*)$ soit $C_w[D_v[u, w]]$. Par définition, la procédure d'étiquetage, quand on l'applique avec v et u, étiquette chaque nœud $w \in ch(v)$ avec $D_v[u, w]$, donc le poids de $\kappa(P_v^*)$ est $\sum_{w \in ch(v)} \omega^{\triangleright}(u, D_v[u, w]) + C_w[D_v[u, w]]$.

Ainsi la propriété est vérifiée pour v. Ceci conclut la récurrence, et prouve le lemme. □

Nous prouvons maintenant que $C_r[r]$ est le poids de T^*. Puisque le lemme 10.1.4 prouve que la procédure d'étiquetage construit une solution de poids $C_r[r]$, ces deux lemmes montrent que nous construisons une solution optimale.

Lemme 10.1.5. *$C_r[r]$ est le poids d'une solution optimale de $\mathcal{I}$.*

Démonstration. Nous allons montrer la propriété suivante par induction sur la hauteur de v dans P^* : "$C_v[\kappa^*(v)]$ est le poids de $\kappa^*(P_v^*)$". Si t est une feuille de P, $\kappa^*(t) = t$. Puisque $C_t[t] = 0$ et P_t^* ne contient aucun arc, la propriété est vérifiée pour t.

Soit v un nœud interne de P. $C_v[\kappa^*(v)] \leq \sum_{w \in ch(v)} \omega^{\triangleright}(\kappa^*(v), \kappa^*(w)) + C_w[\kappa^*(w)]$, d'après l'équation (10.2). Par hypothèse de récurrence, pour tout fils $w \in ch(v)$, $C_w[\kappa^*(w)]$ est le poids de $\kappa^*(P_w^*)$. Donc $C_v[\kappa^*(v)]$ ne peut excéder le poids de $\kappa^*(P_v^*)$.

Supposons maintenant que $C_v[\kappa^*(v)]$ soit strictement plus petit que ce poids : la procédure d'étiquetage construirait alors un étiquetage κ' pour P_v^* de poids strictement plus petit que $\kappa^*(P_v^*)$, d'après le lemme 10.1.4. Donc, si on construit l'étiquetage κ avec κ' pour tous les nœuds de P_v^* et κ^* pour tous les autres nœuds de P, on construit une solution réalisable $\kappa(P^*)$ de poids strictement plus petit que celui de $\kappa^*(P^*)$, ce qui, d'après le lemme 10.1.3, contredirait l'optimalité de T^*. Ainsi, la propriété est prouvée pour v.

Ceci conclut la récurrence, ce qui implique que la propriété est vérifiée pour tout noeud de P, et que $C_r[r]$ est le poids de $\kappa^*(P_r^*) = T^*$. □

Théorème 10.1.6. *L'algorithme 9 renvoie une solution optimale pour $\mathcal{I}$.*

Démonstration. D'après les lemmes 10.1.4 et 10.1.5, la procédure d'étiquetage renvoie une solution optimale quand on l'applique sur P^*. Par conséquent, l'algorithme 9 renvoie une solution optimale pour $\mathcal{I}$. □

Théorème 10.1.7. *La complexité temporelle de l'algorithme 9 est $O(t_d \cdot d^k \cdot n \cdot (d+k))$ où t_d est le nombre d'arborescences dont les fils de chaque nœud ne sont pas ordonnés et contenant d nœuds internes non étiquetés. Sa complexité spatiale est $O((d+k) \cdot n + n^2)$.*

Démonstration. D'après le lemme 10.1.1, il existe au plus $t_d \cdot d^k$ patrons pour $\mathcal{I}$. Après sélection d'un patron lors d'une itération donnée, il est possible d'entamer l'itération suivante (et donc construire le patron de cette itération) en au plus $d+k$ opérations.

Pour chaque patron P, un programme dynamique construit à la ligne 3 les tableaux C_v et D_v en temps $O((n^2+n) \cdot ch(v))$ pour chaque nœud v de P à l'aide des équations (10.1) et (10.2). Puisque P contient $d+k$ arcs, cet algorithme effectue au plus $O((n^2+n) \cdot (d+k))$ opérations. La procédure d'étiquetage à la ligne 4 est appliquée récursivement une seule fois par nœud et peut construire pendant son exécution la solution $T = \kappa(P)$ à la ligne 5. Il faut donc $O(n)$ opérations pour effectuer ces deux lignes. Enfin, la comparaison à la ligne 6 se fait en temps constant tant que la meilleure solution trouvée et son poids sont stockés en mémoire. Donc cet algorithme a une complexité temporelle égale à $O(t_d \cdot d^k \cdot n^2 \cdot (d+k))$.

À chaque itération, l'algorithme doit stocker le graphe des plus courts chemins $G^{\triangleright}$, de taille $O(n^2)$, la solution T_0, de taille $O(d+k)$, et son poids, le patron courant P, et, pour chaque nœud v de P, les tableaux C_v et D_v de taille $n \cdot (ch(v)+1)$. Puisque P contient exactement $d+k$ arcs, la complexité spatiale de l'algorithme est $O(n^2 + n \cdot (d+k))$. □

Théorème 10.1.8. *Si $\mathcal{I}$ est une instance de DST, la complexité temporelle de l'algorithme est $O^*(1 \cdot 3 \cdot 5 \cdots 2k-3)$.*

Démonstration. Si $\mathcal{I}$ est une instance de DST, $d = k-1$. D'après le lemme 10.1.2, il existe alors $1 \cdot 3 \cdot 5 \cdots 2k-3$ patrons pour $\mathcal{I}$, on peut donc réduire la complexité temporelle du théorème 10.1.7 à $O^*(1 \cdot 3 \cdot 5 \cdots 2k-3)$. □

10.2 Algorithme exact probabiliste FPT en k et n_s pour DSTLB et USTLB

Nous allons nous inspirer de l'algorithme précédent et de la technique utilisée dans [AYZ95] pour trouver un algorithme FPT en k et n_s pour DSTLB et USTLB. Nous démontrons le résultat dans le cas orienté, la version non orientée est similaire.

Cette technique est celle du *color coding*. Elle permet, à l'aide d'un algorithme probabiliste, de chercher des structures prédéterminées dans un graphe.

10.2.1 Introduction au color coding

Si on recherche, par exemple, un chemin de longueur 4 dans le graphe, une première idée est de tester si les n^4 tuples de 4 nœuds qui existent ne forment pas un chemin dans le graphe. Cet algorithme n'est pas FPT en la taille du chemin. La technique de color coding accélère cette recherche, avec un algorithme probabiliste : désigner 4 couleurs distinctes et les classer dans un ordre arbitraire, par exemple (Rouge, Vert, Bleu, Jaune). Puis colorier chacun des nœuds du graphe aléatoirement uniformément avec une de ces quatre couleurs. Rechercher dans le graphe un chemin reliant successivement un nœud Rouge, puis Vert, puis Bleu, puis Jaune, peut être fait en temps $O(4 \cdot n^2)$. Cet algorithme est FPT en la longueur du chemin.

La probabilité de réussir dépend de la probabilité de colorier correctement une des solutions réalisables recherchées. Si aucun chemin de longueur 4 n'existe, l'algorithme répondra toujours par la négative. Mais s'il existe effectivement un chemin de longueur 4, la coloration doit colorier le premier nœud de ce chemin en Rouge, le second en Bleu, etc. Cette probabilité ne dépend que de la taille du chemin, et on peut montrer qu'elle vaut ici 4^{-4}. En recommençant l'algorithme un nombre suffisant de fois, on est capable de rendre cette probabilité aussi proche de 1 qu'on le souhaite.

10.2.2 Définition des patrons

Plaçons-nous dans une instance $\mathcal{I} = (G = (V, A), r, X, \omega, p)$ de DSTLB. On recherche des solutions réalisables contenant au plus n_s nœuds non terminaux.

Nous allons encore une fois nous servir de la notion de patrons, adaptée à notre problème. Les définitions changent légèrement par rapport à la section 10.1, car nous devons tenir compte du fait que la solution réalisable ne peut contenir que n_s nœuds non terminaux et que les terminaux ne sont pas nécessairement des feuilles.

Définition 11. Un *patron* pour l'instance $\mathcal{I}$ est une arborescence enracinée en r contenant les k terminaux X, et au plus n_s autres nœuds non étiquetés. Un patron ne peut contenir plus de p nœuds de branchement. Les fils d'un même nœud ne sont pas ordonnés : il est possible de les inverser (avec leurs descendants) sans altérer le patron.

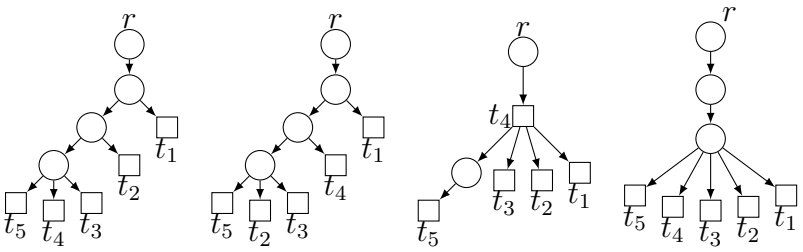


Figure 10.3 – *Quatre exemples de patrons distincts pour une instance vérifiant $n_s = 3$ et $k = 5$. Si, dans le premier patron, on échange t_2 et t_4, on obtient le deuxième patron, mais, si on échange t_3 et t_4, le patron ne change pas car ils sont fils d'un même nœud.*

Les définitions d'un étiquetage κ et d'une solution réalisable $\kappa(P)$ ne changent pas, à ceci près qu'il est possible que $\kappa(P)$ n'existe pas dans le graphe.

Définition 12. Soit P un patron de $\mathcal{I}$ et I_P ses nœuds internes non étiquetés. Un *étiquetage* de p est une fonction κ associant à chaque nœud de I_P un nœud de V et associant la racine et les terminaux à eux-mêmes. Si l'arbre résultant de cet étiquetage existe dans G, on le note $\kappa(P)$.

Lemme 10.2.1. *Il existe au plus $t_{n_s+k+1} \cdot (n_s + k + 1)(n_s + k) \cdots (n_s + 2)$ patrons pour l'instance $\mathcal{I}$ où t_d est le nombre d'arborescences dont les fils de chaque nœud ne sont pas ordonnés contenant d nœuds internes non étiquetés.*

Démonstration. Un patron pour $\mathcal{I}$ est une arborescence nombre d'arborescences dont les fils de chaque nœud ne sont pas ordonnés enracinée en r, contenant n_s nœuds internes non étiquetés et k terminaux étiquetés.

Donc il est possible de construire un ensemble contenant tous les patrons (ainsi que d'autres arbres) en énumérant toutes les arborescences dont les fils de chaque nœud ne sont pas ordonnés et contenant $n_s + k + 1$ nœuds internes non étiquetés, et en étiquetant k nœuds avec les terminaux. Le nombre de tels arbres s'élève à $t_{n_s+k+1} \cdot (n_s+k+1)(n_s+k)\cdots(n_s+2)$. En réalité, il n'existe pas autant de patrons, car nous ne nous intéressons qu'à ceux contenant au plus p nœuds de branchement et pour lesquels les feuilles sont toutes terminales. □

10.2.3 Trouver une solution réalisable contenant exactement n_s nœuds dans une instance de DSTLB

La technique qui suit est une adaptation de celle donnée dans [AYZ95] pour retrouver toute structure de largeur d'arbre bornée prédéterminée dans un graphe.

Soit P un patron pour $\mathcal{I}$. On recherche un étiquetage de P tel que $\kappa(P)$ existe.

La procédure de coloration et l'étiquetage coloré d'un patron. On commence par associer à tout nœud v de P une unique couleur c_v. Pour chaque nœud u de $\{r\} \cup X$, on colorie u dans G avec c_u. On nomme l'ensemble de toutes les autres couleurs c_{I_P}. Pour tout nœud u de $G \backslash (\{r\} \cup X)$, on colorie u aléatoirement uniformément avec une couleur de c_{I_P}. Un exemple est donné en figure 10.4. De la même manière que pour le problème de recherche d'un chemin de longueur 4, on va rechercher, dans le graphe G, notre patron colorié. En d'autres termes, on va chercher un étiquetage de P sous la contrainte que le nœud Rouge de P ne puisse être étiqueté qu'avec un nœud Rouge de G, et qu'il en soit de même pour toutes les autres couleurs.

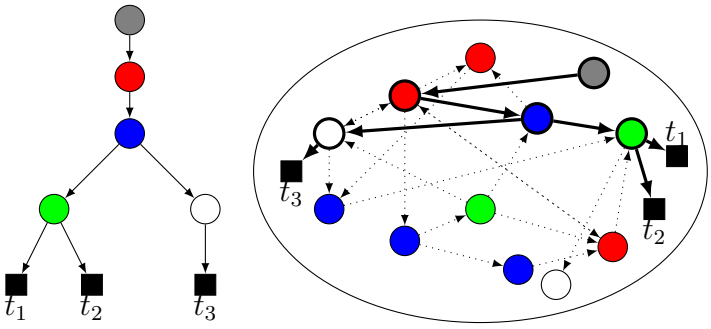


FIGURE 10.4 – *A gauche, l'arbre T, et le choix des couleurs $c_1, c_2, \ldots, c_{n_s}$ de ses nœuds internes. La racine est en gris, les terminaux en noir. A droite, dans le graphe G, un arbre qui correspond au coloriage recherché.*

Définition 13. Un *étiquetage colorié* de P est un étiquetage κ de P où tout nœud v de P est étiqueté avec un nœud de couleur c_v. Un *étiquetage colorié consistant* κ est un étiquetage colorié tel $\kappa(P)$ existe dans $\mathcal{I}$.

Question. Existe-t-il un étiquetage colorié consistant de P ?

Nous utilisons l'algorithme 10 pour renvoyer une réponse à cette question en temps polynomial.

Algorithme 10 Algorithme d'étiquetage d'un patron colorié

```
1: Ordonner les nœuds de P avec un ordre inverse d'un parcours en largeur.
2: Pour v ∈ P Faire
3:     L ← les nœuds G colorés avec c_v
4:     Pour u ∈ L et w, fils de v dans P Faire
5:         Si u n'est relié à aucun nœud non marqué de G coloré avec c_w Alors
6:             Marquer u.
7:     Si L est entièrement marqué Alors Renvoyer Faux
8: Renvoyer Vrai
```

La procédure d'étiquetage. Si l'algorithme 10 renvoie Faux, cette procédure n'est pas utilisée. Sinon, la *procédure d'étiquetage* construit récursivement un étiquetage κ de P. Connaissant un nœud v de P et un nœud u de couleur c_v non marqué, la procédure affecte $\kappa(v) = u$ et, pour tout fils w de v, s'appelle elle-même avec, comme paramètres, w et un nœud non marqué de couleur c_w.

Pour construire l'étiquetage complet, on appelle cette procédure sur la racine de P et r.

Preuve de la validité de l'algorithme.

Lemme 10.2.2. *L'algorithme 10 renvoie Vrai si et seulement s'il existe un étiquetage colorié consistant de P.*

Démonstration. Si l'algorithme renvoie vrai, r n'est pas marqué. De plus, pour tout nœud $v \in P$, tout nœud u de G coloré avec c_v non marqué, et tout fils w de v, u contient au moins un successeur coloré avec c_w non marqué. On peut donc prouver par récurrence que la procédure d'étiquetage construit un étiquetage κ tel que $\kappa(P)$ existe et, pour tout nœud v, $\kappa(v)$ est coloré avec c_v.

Si l'algorithme renvoie Faux, alors il existe un nœud v tel que tous les nœuds de G colorés avec c_v sont marqués. Supposons qu'il existait un étiquetage coloré consistant κ. Soit $u = \kappa(v)$. Puisque $\kappa(P)$ existe, alors, pour tout fils w de v, l'arc $(u, \kappa(w))$ existe dans G et $\kappa(w)$ est coloré avec c_w. Or u est marqué, donc $\kappa(w)$ est également marqué. On peut donc prouver par récurrence que, pour toute feuille l qui est un descendant de v, $\kappa(l)$ est marqué, ce qui est une contradiction avec le fait que l'algorithme ne peut marquer les nœuds colorés avec c_l si l est une feuille. □

Lemme 10.2.3. *La complexité temporelle de l'algorithme 10 est $O((n_s + k + 1) \cdot n^2)$.*

Démonstration. Pour tout nœud v, il existe au plus n nœuds de couleur c_v et n nœuds colorés avec les couleurs de l'ensemble des fils de v. La boucle interne est donc appelée au plus n^2 fois. Un patron contenant au plus $n_s + k + 1$ nœuds, le temps de calcul s'en déduit. □

Trouver une solution réalisable pour $\mathcal{I}$. On suppose qu'il existe un étiquetage coloré consistant κ de P pour une certaine coloration de G. Si, pour tout nœud v de P, la procédure de coloration colorie $\kappa(v)$ avec c_v, alors l'algorithme 10 renvoie Vrai d'après le lemme 10.2.2. La probabilité qu'une telle coloration soit faite est $1/n_s^{n_s}$.

Si, à l'inverse, aucune solution réalisable n'existe, alors d'après le lemme 10.2.2, l'algorithme 10 renvoie Faux quels que soient le patron et le coloriage.

L'algorithme probabiliste qui suit trouve donc une solution réalisable pour $\mathcal{I}$ en temps FPT vis-à-vis de k et n_s avec une probabilité $1/n_s^{n_s}$. On commence par énumérer tous les patrons P de $\mathcal{I}$en énumérant les arbres décrits dans la démonstration du lemme 10.2.1. Si l'arbre contient strictement plus de p nœuds de branchement, on le rejette, sinon on colorie ce patron P et le graphe G avec la procédure de coloration et on applique l'algorithme 10. Si l'algorithme renvoie Vrai, il existe une solution réalisable que la procédure de construction dévoilera. Sinon on continue avec le patron suivant.

D'après le lemme 10.2.1, le nombre de patrons est au plus $t_{n_s+k+1} \cdot (n_s + k + 1)(n_s + k) \cdots (n_s + 2)$ où t_d est le nombre d'arborescences dont les fils de chaque nœud ne sont pas ordonnés et contenant d nœuds internes non étiquetés. Chaque appel de l'algorithme 10 effectue au plus $O((n_s + k + 1) \cdot n^2)$ opérations d'après le lemme 10.2.3. Donc cet algorithme est FPT en k et n_s.

Théorème 10.2.4. *Il existe un algorithme probabiliste et FPT en k et n_s renvoyant une solution réalisable de $\mathcal{I}$ si elle existe.*

Remarque 20. Pour tout $\epsilon > 0$, il est possible de recommencer cette procédure un nombre de fois dépendant uniquement de n_s et de ϵ pour trouver une solution, si elle existe, avec une probabilité de $1 - \epsilon$.

10.2.4 Trouver une solution optimale contenant exactement n_s nœuds dans une instance de DSTLB

Il est possible de trouver une solution optimale en temps FPT en k et n_s. La solution consiste à remplacer l'algorithme 10 par une variante de l'algorithme 9 basé sur la programmation dynamique dans laquelle nous devons tenir compte des couleurs et des nœuds marqués. Les modifications à apporter sont les suivantes :

- on recherche, pour chaque patron P, un étiquetage colorié consistant de poids minimum pour P ;
- à la définition des tableaux C_v et D_v s'ajoute celle des nœuds marqués M_v : la case $M_v[u]$ est vraie si et seulement s'il existe un étiquetage colorié consistant du sous-patron enraciné en v tel que $\kappa(v) = u$;
- on construit C, D et M ainsi :

$$M(v)[u] = \bigwedge_{w \in ch(v)} \bigvee_{x \in V} ((u, x) \in G \wedge M(w)[x]) \tag{10.3}$$

$$D(v)[u, w] = \arg \min_{\substack{x \in V \\ (u,x) \in G \wedge M(w)[x]}} (\omega(u, x) + C(w)[x]) \tag{10.4}$$

$$C(v)[u] = \sum_{w \in ch(v)} \min_{\substack{x \in V \\ (u,x) \in G \wedge M(w)[x]}} (\omega(u, x) + C(w)[x]) \tag{10.5}$$

- la procédure d'étiquetage n'est utilisée que si aucun tableau M_v n'est rempli que par la valeur Faux.

Les démonstrations prouvant la validité de cet algorithme sont semblables à celles de la section 10.1.

Théorème 10.2.5. *Il existe un algorithme probabiliste et FPT en k et n_s renvoyant une solution optimale de $\mathcal{I}$, si elle existe, avec une probabilité $1/n_s^{n_s}$.*

Il n'est pas exclu que ces algorithmes probabilistes soient déterminisables sans détériorer le temps de calcul, puisqu'il est généralement possible de déterminiser les algorithmes basés sur la technique du color coding.

10.3 Algorithme exact XP en k et p pour DSTLB et USTLB

Dans cette section, nous utilisons une deuxième variante de l'algorithme d'énumération pour démontrer, dans trois cas distincts, des propriétés de complexité paramétrée de DSTLB et USTLB vis-à-vis des paramètres k et p.

Soient $\mathcal{I}^D_{Pl}$ une instance de DSTLB dans un graphe planaire, $\mathcal{I}^U$ une instance de USTLB et $\mathcal{I}^U_{Tw}$ une instance de USTLB dont la largeur d'arbre est bornée. Dans ces trois instances, on cherchent à couvrir k terminaux et on ne s'autorisent pas à utiliser plus de p nœuds de branchement dans une solution. Nous allons démontrer trois résultats :

- le problème de trouver une solution **réalisable** de l'instance $\mathcal{I}^D_{Pl}$ est XP en k et p ;
- le problème de trouver une solution **réalisable** de l'instance $\mathcal{I}^U$ est XP en k et p ;
- le problème de trouver une solution **optimale** de l'instance $\mathcal{I}^U_{Tw}$ est XP en k et p.

10.3.1 Définition des patrons

Nous utiliserons dans cette partie un troisième type de patrons, simplifiés par rapport aux deux premiers types présentés dans ce chapitre.

Un *patron* est une solution réalisable modifiée de façon à ne contenir que des nœuds de branchement (on contracte les sommets de degré 2 tant qu'il en reste), les terminaux de cette solution, et la racine si l'instance est orientée. Les propriétés principales des patrons considérés sont les suivantes :

- chaque solution réalisable est associée à un patron, mais un patron peutêtre associé à plusieurs solutions réalisables,
- le nombre de patrons ne dépend que de p et k,
- pour chaque patron, en temps polynomial une solution réalisable ou optimale (selon le cas), qui lui est associée.

Soit $\mathcal{I}$ une instance du problème DSTLB. Les définitions qui suivent sont similaires dans le cas non orienté.

Définition 14. Etant donnée T une solution réalisable de $\mathcal{I}$, le *patron P associé à T* est une arborescence construite ainsi : on supprime toutes les extrémités de T jusqu'à ce que toutes les feuilles soient terminales, puis on contracte les nœuds non terminaux de degré 2 dans T, tant que c'est possible. On note $T \to P$.

Un exemple d'arborescence et de patron associé est donné en figure 10.5.

Définition 15. On note $\Pi(\mathcal{I})$ l'ensemble des arborescences ne contenant que les k terminaux de $\mathcal{I}$, au plus p nœuds de branchement choisis dans V et la racine de $\mathcal{I}$.

Remarque 21. Les arborescences de $\Pi(\mathcal{I})$ ne sont pas nécessairement des sous-graphes de G. En effet, la définition 15 ne pose pas de contrainte sur les arcs de ces arborescences : deux nœuds peuvent être reliés par un arc qui n'existe pas dans G. Par exemple dans la figure 10.5,

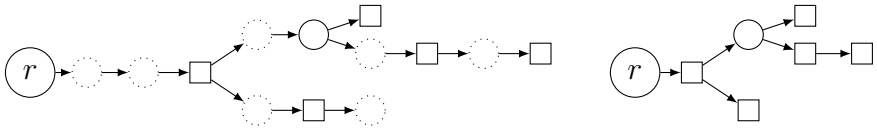


FIGURE 10.5 – *Un exemple d'arborescence T et de patron P tels que $T \to P$. Les nœuds en pointillés sont contractés ou supprimés.*

le patron P est dans $\Pi(\mathcal{I})$ mais on peut supposer que P n'est pas un sous-graphe de G, contrairement à la solution réalisable dont il est issu.

Lemme 10.3.1. *L'ensemble des patrons de $\mathcal{I}$ est inclus dans $\Pi(\mathcal{I})$.*

Il est donc possible d'énumérer tous les éléments de $\Pi(\mathcal{I})$ pour parcourir l'ensemble des patrons de $\mathcal{I}$. Cette énumération se fait en temps XP en k et p.

Par abus de langage, pour toute solution réalisable T de $\mathcal{I}$ et toute arborescence T' de $\Pi(\mathcal{I})$, la notation $T \to T'$ peut être utilisée et représente un booléen qui n'est vrai que si T' est le patron associé à T, décrit par la définition 14.

Définition 16. Pour toute arborescence $P \in \Pi(\mathcal{I})$, $S(P)$ est l'ensemble des solutions réalisables T de $\mathcal{I}$ telles que $T \to P$.

10.3.2 Trouver une solution réalisable pour l'instance $\mathcal{I}^U$

Il nous faut maintenant déterminer si $S(P)$ est vide pour tout $P \in \Pi(\mathcal{I}^U)$. L'union de tous ces ensembles $S(P)$ est vide si et seulement si l'instance n'a pas de solution réalisable.

Lemme 10.3.2. *Soit $P \in \Pi(\mathcal{I}^U)$, ces deux propriétés sont équivalentes :*

1. *T est une solution réalisable de $\mathcal{I}$ vérifiant $T \to P$,*
2. *T est l'union de chemins élémentaires, un par arête de P reliant ses extrémités, tous deux à deux nœuds-disjoints sauf éventuellement aux extrémités.*

Démonstration. Supposons vraie la propriété 2. Alors T contient tous les nœuds de P donc couvre tous les terminaux. P est un arbre donc T est un arbre, sinon il existerait dans T et donc dans P deux nœuds u et v reliés par deux chemins distincts.

Enfin, tous les nœuds de branchement de T sont les nœuds de branchement de P. En effet, si un nœud v est un nœud de branchement dans T, alors il existe au moins trois chemins distincts[3] émanant de v dans T. Soit a un arc de P, on nomme *extension de a* le chemin reliant les extrémités de a dans T. Ces trois chemins sont l'union de trois extensions d'arcs ayant v pour extrémité. Dans le cas contraire, soit on ne respecte pas la contrainte d'élémentarité des extensions, soit on ne respecte pas celle de disjonction. Donc v est un nœud de branchement de P.

Donc T est une solution réalisable. Si on contracte, comme expliqué dans la définition 14, les chemins reliant des nœuds de branchement et/ou des terminaux, on retrouve P. La propriété 1 est donc vérifiée.

3. On rappelle que dans le cas non orienté, un nœud de branchement est défini comme un nœud possédant trois voisins.

Inversement si cette propriété est vraie alors tout chemin contracté en un unique arc reliant des nœuds de branchement et/ou des terminaux est nécessairement disjoint des autres car T est un arbre. □

Lemme 10.3.3. *Soit $P \in \Pi(\mathcal{I}^U)$, il est possible de déterminer si $S(P)$ est vide en temps FPT en k et p.*

Démonstration. D'après le lemme 10.3.2, T est une solution réalisable de $\mathcal{I}^U$ vérifiant $T \to P$ si et seulement si T est l'union de chemins élémentaires reliant les extrémités des arêtes de P, tous deux à deux nœuds-disjoints sauf éventuellement aux extrémités.

Soit m_P le nombre d'arcs de P. Puisque P est un arbre contenant au plus $k+p$ nœuds [4], alors $m_P \leq k+p$. Or, dans un graphe non orienté, trouver des chemins deux à deux nœuds-disjoints entre au plus $k+p$ couples distincts de nœuds, s'ils existent, est un problème FPT en k et p [RS95]. Donc il est possible, en temps FPT vis-à-vis de k et p, de déterminer une solution réalisable T de l'instance $\mathcal{I}^U$ vérifiant $T \to P$ si $S(P)$ est non vide, et de certifier que $S(P)$ est vide sinon. □

Théorème 10.3.4. *Il est possible, en temps XP vis-à-vis de k et p, de trouver une solution réalisable pour l'instance $\mathcal{I}^U$ si elle existe, et de certifier qu'aucune n'existe sinon.*

Démonstration. Pour toute solution réalisable de $\mathcal{I}^U$, il existe un patron P tel que $T \to P$. Puisqu'il est possible d'énumérer tous les arbres de $\Pi(\mathcal{I}^U)$ en temps XP en k et p, et, d'après le lemme 10.3.3, de déterminer ensuite, pour chaque patron P, si $S(P)$ est vide en temps FPT en k et p, alors le théorème est prouvé. □

10.3.3 Trouver une solution réalisable pour l'instance $\mathcal{I}^D_{Pl}$

Sachant qu'il est possible, en temps FPT vis-à-vis de k et p, de trouver des chemins deux à deux nœuds-disjoints entre au plus $k+p$ couples distincts de nœuds dans un graphe orienté planaire s'ils existent (et de certifier qu'ils n'existent pas sinon) [Sch94b], on peut prouver de manière similaire au lemme 10.3.3 le lemme suivant.

Lemme 10.3.5. *Soit $P \in \Pi(\mathcal{I}^D_{Pl})$, il est possible de déterminer si $S(P)$ est vide en temps FPT en k et p.*

On prouve donc le théorème suivant.

Théorème 10.3.6. *Il est possible, en temps XP vis-à-vis de k et p, de trouver une solution réalisable pour l'instance $\mathcal{I}^D_{Pl}$ si elle existe, et de certifier qu'aucune n'existe sinon.*

10.3.4 Trouver une solution optimale pour l'instance $\mathcal{I}^U_{Tw}$

Enfin, nous pouvons montrer que, dans le cas d'une instance non orientée dont la largeur d'arbre est bornée, il est possible de déterminer en temps XP une solution non seulement admissible, mais même optimale.

De manière similaire au lemme 10.3.3, nous allons vérifier le contenu de $S(P)$, mais à présent, nous désirons trouver une solution réalisable de poids minimum dans $S(P)$. Pour

4. On rappelle que dans le cas non orienté, aucune racine n'est définie par les instances : un patron ne contient que des terminaux et des nœuds de branchement.

cela nous allons nous intéresser au problème de trouver, s'ils existent, des chemins deux à deux nœuds-disjoints entre au plus $k + p$ couples distincts de nœuds dans un graphe non orienté de largeur d'arbre bornée tels que la somme des poids des chemins soit minimum. Ce problème est FPT en k et p [Sch94a]. On peut donc prouver le lemme suivant.

Lemme 10.3.7. *Soit $P \in \Pi(\mathcal{I}^U_{Tw})$, il est possible de trouver, si $S(P)$ est non vide, une solution de poids minimum dans cet ensemble en temps FPT en k et p.*

On peut également tester si $S(P)$ est vide en temps FPT vis-à-vis de k et p, et on prouve donc le théorème suivant

Théorème 10.3.8. *Il est possible, en temps XP vis-à-vis de k et p, de trouver une solution optimale pour l'instance $\mathcal{I}^U_{Tw}$ si elle existe, et de certifier qu'aucune solution réalisable n'existe sinon.*

10.4 Synthèse des résultats

Les résultats présentés dans ce chapitre sont résumés dans le tableau 10.6.

Problème	k	p / d	n_s	Conditions	Trouver une solution réalisable	Minimisation
DSTLD	×	×			FPT	FPT
USTLD	×	×			FPT	FPT
DST	×				FPT	FPT
UST	×				FPT	FPT
DSTLB	×		×		FPT Probabiliste	FPT Probabiliste
USTLB	×		×		FPT Probabiliste	FPT Probabiliste
DSTLB	×	×		Graphe planaire	XP	OUVERT
USTLB	×	×			XP	OUVERT
USTLB	×	×		Largeur d'arbre bornée	XP	XP

Table 10.6 – *Résultats démontrés de ce chapitre.*

Nous avons également mentionné le fait que l'algorithme, présenté dans la section 10.1, pour résoudre les problèmes DSTLD, USTLD, DST et UST est un nouvel algorithme qui s'exécute en temps FPT vis-à-vis du paramètre k et en espace polynomial, et qui construit puis renvoie une solution optimale.

Les résultats concernant DSTLD ont été publiés dans le journal IPL [**?**]. Les résultats de complexité paramétrée de DSTLB vis-à-vis des paramètres k et p ont été publiés à la conférence SIROCCO 2013 [WWBB13] avec les résultats du chapitre 8.

Conclusion

Dans cette partie, nous avons introduit et étudié deux variantes du problème de Steiner :

- le problème de l'arborescence de Steiner avec un nombre limité de nœuds de branchement, DSTLB, où on ne recherche que des solutions réalisables qui n'ont pas plus qu'un nombre fixé p de nœuds de branchement ;
- le problème de l'arborescence de Steiner avec un nombre limité de nœuds diffusants, DSTLD, où on ne recherche, dans le graphe des plus courts chemins, que des solutions réalisables qui n'ont pas plus qu'un nombre fixé d de nœuds de branchement.

Au-delà de leur intérêt intrinsèque, l'étude de ces problèmes a aussi pour objectif de permettre d'établir de nouvelles méthodes d'approximation pour le problème de Steiner.

DSTLB est un problème difficile à résoudre. Dans le cas général, prouver l'existence d'une solution réalisable est un problème NP-complet, même quand le nombre de terminaux et de nœuds de branchement maximum sont fixés. De plus, dans des cas plus simples comme les graphes non orientés ou orientés sans circuits, il n'existe pas aujourd'hui de méthode rapide pour résoudre ce problème. Rappelons toutefois que, dans le dernier cas, savoir si ce problème est FPT vis-à-vis du nombre de terminaux est une question ouverte.

DSTLD est un problème plus simple à résoudre. Il existe toujours une solution réalisable, qui est une k-approximation de la solution optimale. En outre, ce problème permet de construire des approximations pour le problème de Steiner : en contraignant artificiellement le nombre de nœuds diffusants à une certaine quantité d, et en déterminant ensuite une solution optimale au problème contraint, on obtient un $\lceil \frac{k-1}{d} \rceil$-approximation pour le problème de Steiner.

C'est donc sur la recherche de solutions optimales ou approchées pour DSTLD qu'il faut se concentrer. Nous avons montré que si d est petit alors il est possible de trouver une solution optimale rapidement. Nous n'avons en revanche pas d'algorithme d'approximation pour DSTLD, autre que la k-approximation des plus courts chemins. De plus, nous connaissons un fort résultat d'inapproximabilité pour ce problème. Cependant, bien que ce résultat ternisse les espoirs que l'on pourrait avoir concernant les capacités d'approximation de cette technique, il diminue quand d augmente, de même que la distance séparant la solution optimale de DSTLD et celle de DST, jusqu'à atteindre une valeur constante quand d atteint sa valeur maximale $k-1$. Il n'est donc pas exclu de trouver de bons rapports d'approximation de DST à l'aide de DSTLD.

À l'inverse, nous savons que DST n'est pas approximable avec un rapport logarithmique en k. C'est donc qu'il existe une zone de transition entre le rapport d'inapproximabilité de DSTLD et celui de DST. Rechercher comment évolue l'inapproximabilité de DSTLD quand d augmente, et en particulier quand il atteint $k-1$, peut donc nous donner des informations sur l'inapproximabilité de DST.

Conclusion générale

Synthèse des résultats

Cette thèse a permis d'établir de nouveaux résultats concernant l'approximabilité du problème de Steiner.

Parmi les résultats qui existent aujourd'hui, seuls deux, datant de plusieurs décennies, dominent. Premièrement l'algorithme de Charikar montre que le problème de l'arborescence de Steiner peut être approché avec un rapport $O(k^{\varepsilon})$ pour tout $\varepsilon > 0$; et secondement, le résultat d'inapproximabilité de Halperin et Krauthgamer montre que ce même problème ne peut être approché avec un rapport $\log^{2-\varepsilon}(k)$, pour tout $\varepsilon > 0$, si $NP \not\subset ZTIME(n^{\text{polylog}(n)})$. Ces deux résultats se rejoignent quand la profondeur H de l'instance est fixée, montrant ainsi que le plus petit rapport d'approximation du problème de Steiner est $H\log(k)$.

Dans le chapitre 5, nous nous sommes intéressés au cas restreint des graphes sans circuits structurés en paliers où H n'est pas constant, et en particulier, où H est supérieur à $2\log(k)$. Nos résultats montrent que, dans ce cas, le rapport d'approximation est $\sqrt{H\log(k)} < H\log(k)$.

Les autres algorithmes d'approximations, précédemment développés pour le problème de Steiner comme par exemple les heuristiques d'ascension du dual, n'ont jamais permis de réduire le rapport d'approximation. Ils n'ont donc que peu d'intérêt théorique. De plus, leur utilité est limitée car l'algorithme des plus courts chemin, une k-approximation, est un algorithme simple à implémenter, très rapide et qui renvoie des solutions de bonne qualité. De ce fait, les applications du problème de Steiner implémentent l'algorithme des plus courts chemins plutôt que tout autre algorithme. Dans le chapitre 4, nous avons développé un algorithme de rapport d'approximation égal à k, nommé Greedy$_{FLAC}$, qui associe les idées de l'algorithme de Charikar et des heuristiques d'ascension du dual. Le résultat est un algorithme efficace, renvoyant des solutions de meilleure qualité que celles produites par l'algorithme des plus courts chemins avec un temps d'exécution comparable.

En outre, le chapitre 4 présente un second algorithme, Greedy$^{\triangleright}_{FLAC}$, qui est dérivé de Greedy$_{FLAC}$. Cet algorithme qui est une $O(\sqrt{k})$-approximation pour DST est cependant dominé en pratique par Greedy$_{FLAC}$. En effet, ces deux algorithmes renvoient des solutions de poids proches, mais Greedy$_{FLAC}$ est bien plus rapide que Greedy$^{\triangleright}_{FLAC}$. Toutefois, Greedy$^{\triangleright}_{FLAC}$ est une première étape visant à réduire le rapport d'approximation de Greedy$_{FLAC}$ pour égaler le rapport $O(k^{\epsilon})$ de l'algorithme de Charikar avec une complexité plus raisonnable ou franchir cette limite et ainsi améliorer le meilleur rapport connu pour DST.

Le chapitre 6 présente une approximation exponentielle, combinant un algorithme d'approximation glouton polynomial et un algorithme exact pour DST. À notre connaissance, aucun autre algorithme d'approximation exponentiel pour le problème de l'arborescence Steiner n'a été développé dans la littérature. Le temps de calcul et le rapport de l'approximation exponentielle sont gérés par un paramètre q qui détermine la part de terminaux couverte par l'approximation polynomiale et la part couverte par l'algorithme exact.

Enfin, dans la partie III, nous avons suivi l'idée qui a permis la création de l'algorithme de Charikar. Elle repose sur le fait qu'en restreignant le nombre de solutions réalisables, il est possible de trouver plus facilement une solution de bonne qualité. Dans le cas de l'algorithme de Charikar, il s'agit de limiter la hauteur des solutions réalisables que l'on s'autorise à renvoyer. Nous avons étudié dans cette partie deux problèmes qui restreignent le problème de Steiner différemment, en limitant le nombre de nœuds de branchement ou le nombre de nœuds diffusants, pour examiner les capacités d'approximation résultant de ces nouvelles contraintes.

Ces contraintes sont issues du développement des réseaux tout-optiques dont le fonctionnement diffère de celui des réseaux opto-électroniques traditionnels. L'étude de la partie III a donc également un intérêt intrinsèque : celui de répondre formellement à des questions concrètes de routage multicast dans un réseau tout-optique. Dans quel cas peut-on résoudre exactement ce problème et comment peut-on l'approcher ?

Dans le chapitre 9, nous avons présenté une approximation qui consiste à restreindre artificiellement le nombre de nœuds diffusants autorisés, résoudre le problème contraint et construire une solution réalisable de DST à partir de la solution du problème contraint. Plus le nombre de nœuds diffusants autorisés est grand, meilleure est la solution.

Enfin, nous avons, à l'issue de cette partie, une connaissance plus précise de la complexité du problème de Steiner paramétrée par le nombre de nœuds de branchement ou diffusants.

Perspectives

Savoir si le problème de Steiner peut être approché avec un rapport polylogarithmique en k est toujours un problème ouvert. Le meilleur rapport d'approximation est toujours $O(k^{\varepsilon})$, celui de l'algorithme de Charikar. La plus grande borne inférieure est toujours $\log^{2-\varepsilon}(k)$. Cependant, les résultats de ce manuscrit montrent que le problème semble approchable avec un rapport polylogarithmique.

Premièrement, nous avons évoqué, avec la remarque 1 en page 19 du chapitre 2, le fait que trouver l'arbre de densité minimum permet de concevoir un algorithme d'approximation de rapport $\log(k)$. Les algorithmes Greedy_{FLAC} et $\text{Greedy}^{\triangleright}_{FLAC}$, par l'intermédiaire de l'algorithme FLAC, décrits dans le chapitre 4, semblent être un premier pas vers la recherche d'un algorithme qui approcherait au mieux cet arbre.

Deuxièmement, le meilleur rapport d'approximation connu quand la profondeur H de l'instance est constante est $H\log(k)$. Dans les graphes structurés en paliers, nos résultats montrent que, si la profondeur augmente au-delà de $2\log(k)$, le problème peut être approché avec un rapport strictement inférieur. Ceci semble indiquer que, dans le cas général, le rapport d'approximation croît avec H, tant qu'il est inférieur à $\log(k)$ puis décroît, ce qui montrerait que le meilleur rapport d'approximation serait $O(\log^2(k))$.

Troisièmement, le résultat d'approximabilité du problème de DST à partir du problème DSTLD, décrit dans le chapitre 9, montre que, quand d augmente et s'approche de k, il peut permettre de trouver une approximation pour le problème de Steiner de rapport polylogarithmique, et cela malgré le résultat d'inapproximabilité de DSTLD.

Les perspectives de cette thèse, dans l'optique de répondre à la problématique, se tournent vers une amélioration de l'algorithme FLAC. Trouver une bonne approximation de l'arbre de densité minimum reste aujourd'hui la piste la plus prometteuse pour obtenir un bon algorithme d'approximation polynomial.

La deuxième piste à suivre est celle de DSTLD. Trouver de bons algorithmes d'approximation pour ce problème permettra d'obtenir de bons algorithmes d'approximation pour le problème de Steiner. Travailler avec une contrainte supplémentaire permet d'envisager le problème d'un autre point de vue et donc de rechercher des solutions qui n'ont pas encore été explorées. Dans le même ordre d'idées, rechercher d'autres contraintes que l'on pourrait imposer à DST pour l'étudier donnerait d'autres manières de résoudre ce problème.

Une dernière perspective consiste à étudier comment pourraient s'appliquer les différents algorithmes développés dans cette thèse à d'autres problèmes, en particulier des variantes du problème de Steiner ou plus généralement des problèmes de couverture.

Bibliographie

[AD00] M Ali and J Deogun. Allocation of Splitting Nodes in All-Optical Wavelength-Routed Networks. *Photonic Network Communications*, 2(3) :247–265, 2000.

[AL97] S Arora and C Lund. Hardness of approximations. In Dorit S. Hochbaum, editor, *Approximation algorithms for NP-hard problems*, pages 399–446. PWS Publishing Company, 1997.

[ALM98] S Arora, C Lund, and R Motwani. Proof verification and the hardness of approximation problems. *Journal of the ACM (JACM)*, 45(3) :501–555, 1998.

[AYZ95] N Alon, R Yuster, and U Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4) :844–856, 1995.

[BBP03] L Bahiense, F Barahona, and O Porto. Solving steiner tree problems in graphs with lagrangian relaxation. *Journal of combinatorial optimization*, 7(3) :259–282, 2003.

[Bea84] JE Beasley. An algorithm for the steiner problem in graphs. *Networks*, 14(1) :147–159, 1984.

[Bea89] JE Beasley. An SST-based algorithm for the steiner problem in graphs. *Networks*, 19(1) :1–16, 1989.

[BHKK07] A Björklund, T Husfeldt, P Kaski, and M Koivisto. Fourier meets Möbius : fast subset convolution. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing - STOC '07*, pages 67–74. ACM, 2007.

[CC08] M Chlebík and J Chlebíková. The Steiner tree problem on graphs : Inapproximability results. *Theoretical Computer Science*, 406(3) :207–214, 2008.

[CCCD98] M Charikar, C Chekuri, T Cheung, and Z Dai. Approximation algorithms for directed Steiner problems. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 192–200. Society for Industrial and Applied Mathematics, 1998.

[CCGG98] 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 thirtieth annual ACM symposium on Theory of computing*, pages 114–123. ACM, 1998.

[CD01] X Cheng and D Du. *Steiner trees in industry*, volume 11 of *Combinatorial optimization*. Springer, 2001.

[CDKM00] RD Carr, S Doddi, G Konjevod, and M Marathe. On the red-blue set cover problem. In *In Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 345—-353. Society for Industrial and Applied Mathematics, 2000.

[Cho78] EA Choukhmane. Une heuristique pour le probleme de l'arbre de Steiner. *Revue française d'automatique, d'informatique et de recherche opérationnelle.*, 12 :207–212, 1978.

[CKW09] M Cygan, L Kowalik, and M Wykurz. Exponential-time approximation of weighted set cover. *Information Processing Letters*, 109(16) :957–961, 2009.

[CR41] R Courrant and H Robbins. *What is Mathematics ?* Oxford University Press, London, 1941.

[dAaUW01] MP de Aragão, E Uchoa, and RF Werneck. Dual heuristics on the exact solution of large Steiner problems. *Electronic Notes in Discrete Mathematics*, 7 :150–153, 2001.

[DH08] D Du and X Hu. *Steiner tree problems in computer communication networks*. World Scientific Publishing Co., Inc., 2008.

[DHK09] E Demaine, MT Hajiaghayi, and P Klein. Node-weighted steiner tree and group steiner tree in planar graphs. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming*, volume 5555 of *LNCS*, pages 328–340. Springer Berlin Heidelberg, 2009.

[DV97] C Duin and S Voß. Efficient path and vertex exchange in Steiner tree algorithms. *Networks*, 29(2) :89–105, 1997.

[DW71] SE Dreyfus and RA Wagner. The steiner problem in graphs. *Networks*, 1(3) :195–207, 1971.

[DYWQ07] B Ding, JX Yu, S Wang, and L Qin. Finding top-k min-cost connected trees in databases. In *23rd International Conference on Data Engineering*, pages 836–845. IEEE, 2007.

[Edm67] J Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B(4) :233–240, 1967.

[Eve07] G Even. Recursive greedy methods. In *Handbook of Approximation Algorithms and Metaheuristics (Computer & Information Science Series)*, chapter 5. Chapman \& Hall/CRC, 2007.

[Fei98] U Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4) :634–652, 1998.

[FGK08] FV Fomin, F Grandoni, and D Kratsch. Faster Steiner tree computation in polynomial-space. *Algorithms-ESA*, 5193(Lecture notes in computer science) :430–441, 2008.

[FHW80] S Fortune, J Hopcroft, and J Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2) :111–121, 1980.

[FKM+07] B Fuchs, W Kern, D Molle, S Richter, P Rossmanith, and X Wang. Dynamic programming for minimum Steiner trees. *Theory of Computing systems*, 41(3) :493–500, 2007.

[Flo62] RW Floyd. Algorithm 97 : shortest path. *Communications of the ACM*, 5(6) :345, 1962.

[FS09] P Flajolet and R Sedgewick. *Analytic combinatorics*. Cambridge University Press, 2009.

[GJ79] MR Garey and DS Johnson. *Computers and intractability : a guide to the theory of NP-completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[GJT76] MR Garey, DS Johnson, and RE Tarjan. The planar Hamiltonian circuit problem is NP-complete. *SIAM Journal on Computing*, 5(4) :704–714, 1976.

[GK99] S Guha and S Khuller. Improved methods for approximating node weighted Steiner trees and connected dominating sets. *Information and computation*, 150(1) :57–74, 1999.

[GKR98] N Garg, G Konjevod, and R. Ravi. A polylogarithmic approximation algorithm for the group Steiner tree problem. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 253–259. Society for Industrial and Applied Mathematics, January 1998.

[GT89] AV Goldberg and RE Tarjan. Finding minimum-cost circulations by canceling negative cycles. *Journal of the ACM (JACM)*, 36(4) :873–886, 1989.

[GT93] EN Gordeev and OG Tarastsov. The steiner problem : a survey. *Discrete Mathematics and Applications*, 3(4) :339–364, 1993.

[HF12] T Hibi and T Fujito. Multi-rooted Greedy Approximation of Directed Steiner Trees with Applications. In *Proceedings of the 38th International Conference on Graph-Theoretic Concepts in Computer Science*, LNCS, pages 215–224. Springer-Verlag, 2012.

[HHT06] Ming-I Hsieh, Eric Hsiao-Kuang Wu, and Meng-Feng Tsai. FasterDSP : A faster approximation algorithm for directed Steiner tree problem. *Journal of information science and Engineering*, 22(6) :1409–1425, 2006.

[HK03] E Halperin and R. Krauthgamer. Polylogarithmic inapproximability. In *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*, pages 585–594. ACM, 2003.

[HKK+07] E Halperin, G Kortsarz, R Krauthgamer, A. Srinivasan, and N. Wang. Integrality ratio for group Steiner trees and directed Steiner trees. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 275–284. Society for Industrial and Applied Mathematics, 2007.

[HRZ01] CS Helvig, G Robins, and A Zelikovsky. An improved approximation scheme for the group Steiner problem. *Networks*, 37(1) :8–20, 2001.

[Kar72] RM Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.

[KM98] T Koch and A Martin. Solving Steiner tree problems in graphs to optimality. *Networks*, 32(3) :207–232, October 1998.

[KMB81] L Kou, G Markowsky, and L Berman. A fast algorithm for Steiner trees. *Acta informatica*, 15(2) :141–145, 1981.

[KMV01] T Koch, A Martin, and S Voß. SteinLib : An updated library on Steiner tree problems in graphs. In XiuZhen Cheng and Ding-Zhu Du, editors, *Steiner Tree in Industry*, volume 11 of *Combinatorial optimization*, pages 285–325. Springer, 2001.

[KS06] B Kimelfeld and Y Sagiv. New algorithms for computing Steiner trees for a fixed number of terminals, 2006.

[LB98] A Lucena and JE Beasley. A branch and cut algorithm for the Steiner problem in graphs. *Networks*, 31(1) :39–59, January 1998.

[Mel07] V Melkonian. New primal-dual algorithms for Steiner tree problems. *Computers and operations research*, 34(7) :2147–2167, July 2007.

[mpl] Multiprotocol Label Switching Architecture, Request for Comments : 3031.

[MZQ98] R Malli, X Zhang, and C Qiao. Benefit of Multicasting in All-Optical Networks. In *Proceeding of the SPIE*, volume 3531, pages 209–220, 1998.

[Ned09] J Nederlof. Fast polynomial-space algorithms using möbius inversion : Improving on steiner tree and related problems. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming : Part I*, LNCS, pages 713–725. Springer-Verlag, 2009.

[Ple99] B Plestenjak. An algorithm for drawing planar graphs. *Software : Practice and Experience*, 29(11) :973–984, 1999.

[RŚ85] JL Rémy. Un procédé itératif de dénombrement d'arbres binaires et son application à leur génération aléatoire. *R.A.I.R.O. Informatique théorique*, 19(2) :179–185, 1985.

[Raz98] R Raz. A parallel repetition theorem. *SIAM Journal on Computing*, 27(3) :763–803, 1998.

[RCT+12] V Reinhard, J Cohen, J Tomasik, D Barth, and MA Weisser. Optimal configuration of an optical network providing predefined multicast transmissions. *Comput. Netw.*, 56(8) :2097–2106, 2012.

[RN10] O Russakovsky and A.Y. Ng. A Steiner tree approach to efficient object detection. In *Conference on Computer Vision and Pattern Recognition*, pages 1070–1077. IEEE, 2010.

[Roo01] S Roos. Scheduling for ReMove and other partially connected architectures. Technical report, Laboratory of Computer Engineering, Faculty of Information Technology and Systems, Delft University of Technology, 2001.

[Rot11] T Rothvoß. Directed Steiner tree and the Lasserre hierarchy. *CORR, arXiv 1111.5473*, 2011.

[RS95] N Robertson and PD Seymour. Graph Minors .XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1) :65–110, 1995.

[RS97] R Raz and S Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, number 0, pages 475–484. ACM, 1997.

[RTBW09] V Reinhard, J Tomasik, D Barth, and MA Weisser. Bandwidth Optimization for Multicast Transmissions in Virtual Circuit Networks. In *IFIP Networking*, volume 5550, pages 859–870. Springer Berlin Heidelberg, 2009.

[RZ00] G Robins and A Zelikovsky. Improved Steiner tree approximation in graphs. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 770–779. Society for Industrial and Applied Mathematics, 2000.

[Sch94a] P Scheffler. *A Practical Linear Time Algorithm for Disjoint Paths in Graphs with Bounded Tree Width*. TU, Fachbereich 3, 1994.

[Sch94b] A Schrijver. Finding k disjoint paths in a directed planar graph. *SIAM Journal on Computing*, 23(4) :780–788, 1994.

[Sri01] A Srinivasan. New approaches to covering and packing problems. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 567–576. Society for Industrial and Applied Mathematics, 2001.

[SV06] M Stanojevic and M Vujoševic. An Exact Algorithm for Steiner Tree Problem on Graphs. *International Journal of Computers, Communications and Control*, 1(1) :41–46, 2006.

[Tar77] RE Tarjan. Finding optimum branchings. *Networks*, 7(1) :25–35, 1977.

[Vaz01] VV Vazirani. *Approximation algorithms*. Springer, 2001.

[Voß06] S Voß. Steiner tree problems in telecommunications. In *Handbook of optimization in telecommunications*, pages 459–492. Springer, January 2006.

[Win87] P Winter. Steiner problem in networks : A survey. *Networks*, 17(2) :129–167, 1987.

[Won84] RT Wong. A dual ascent approach for steiner tree problems on a directed graph. *Mathematical Programming*, 28(3) :271–287, 1984.

[WW] D Watel and MA Weisser. A Practical Greedy Approximation for the Directed Steiner Tree Problem. In *Proceeding of the 8th Annual International Conference on Combinatorial Optimization and Applications, accepté, à paraître*. Springer.

[WWBB13] D Watel, MA Weisser, C Bentz, and D Barth. Steiner Problems with Limited Number of Branching Nodes. In *SIROCCO*, pages 310–321, 2013.

[WWBB14] D Watel, MA Weisser, C Bentz, and D Barth. Directed Steiner Tree with Branching Constraint. In Zhipeng Cai, Alex Zelikovsky, and Anu Bourgeois, editors, *Computing and Combinatorics*, volume 8591 of *Lecture Notes in Computer Science*, pages 263–275. Springer International Publishing, 2014.

[YDA03] S Yan, JS Deogun, and M Ali. Routing in sparse splitting optical networks with multicast traffic. *Computer Networks*, 41(1) :89–113, 2003.

[Zel93] A Zelikovsky. An 11/6-approximation algorithm for the network steiner problem. *Algorithmica*, 9(5) :463–470, May 1993.

[Zel97] A Zelikovsky. A series of approximation algorithms for the acyclic directed Steiner tree problem. *Algorithmica*, 18(1) :99–110, 1997.

[ZK02] L Zosin and S Khuller. On directed Steiner trees. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 59–63. Society for Industrial and Applied Mathematics, 2002.

[ZR03] M Zachariasen and A Rohe. Rectilinear group Steiner trees and applications in VLSI design. *Mathematical Programming*, 94(2-3) :407–433, 2003.

Annexes

Annexe A.

Démonstrations reportées

Cette annexe est dédiée aux preuves non données ou partielles dans le chapitre 4 du fait de leur longueur.

Démonstration du théorème 4.2.6, page 45. Soit T une arborescence enracinée en r. Soit X l'ensemble des feuilles de T, toutes terminales. On attache à la racine de T un arc $b = (r_0, r)$ de poids infini, et r_0 remplace r en tant que racine de l'instance de Steiner.

Soit ω le poids de T, k le nombre de feuilles de T et $d = \frac{\omega}{k}$ sa densité. Soit φ l'itération au début de laquelle tous les arcs de T sont saturés. On désigne par ε le flot $f_\varphi(b)$ dans l'arc b au début de cette itération.

Théorème 4.2.6.

$$t_\varphi = d + \frac{\varepsilon}{k}$$

Démonstration. La preuve de cette égalité se fait par récurrence sur la hauteur de l'arbre T.

Initialisation de la récurrence. Si T est de hauteur 1, soient α_1, α_2, ... α_k les poids des arcs reliant la racine aux terminaux $(x_1, x_2, \ldots, x_k)$, triés par ordre croissant de poids (voir la figure A.1).

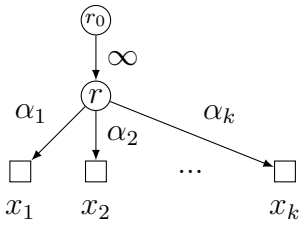


Figure A.1 – *Si la hauteur de T est 1, c'est une étoile accrochée à l'arc b de poids infini.*

Si on applique l'algorithme FLAC à cette arborescence, on remarque qu'à l'itération $i \leq k$, $(u, v)_i = (r, x_i)$ et que $t_i(r, x_i) = \alpha_i - \alpha_{i-1}$ (en posant $\alpha_0 = 0$). On en déduit que $\varphi = k + 1$.

Dans G_{SAT}, chaque nœud x_i ne peut être relié qu'à un seul terminal par un chemin d'arcs saturés : lui-même. Donc $k(r, x_i)$ est toujours égal à 1. D'après le lemme 4.2.2, pour tout $j \leq i + 1$, $f_j(r, x_i) = t_j - t_1 = t_j$. On en déduit que $t_j = \alpha_j$ et $t_\varphi = \alpha_k$.

A chacune de ces itérations, le débit dans l'arc b augmente d'une unité : $k_i(b) = i - 1$. Donc, toujours d'après le lemme 4.2.2, $\varepsilon = \sum_{i=1}^{\varphi-1} (i-1) \cdot (\alpha_i - \alpha_{i-1}) = \sum_{i=1}^{k} (\alpha_k - \alpha_i)$.

On en déduit donc les égalités suivantes :

$$\omega + \varepsilon = \sum_{i=1}^{k} \alpha_i + \sum_{i=1}^{k} (\alpha_k - \alpha_i)$$
$$\omega + \varepsilon = k \cdot \alpha_k$$
$$\omega + \varepsilon = k \cdot t_\varphi$$

Le lemme est donc prouvé si l'arbre est de hauteur 1.

Hérédité de la récurrence. Supposons maintenant l'égalité vérifiée pour tout arbre de hauteur inférieure ou égale à H et supposons la hauteur de T égale à $H + 1$. Soient v_1, v_2, ... v_s les fils de r tels que α_i soit le poids de l'arc $a_i = (r, v_i)$, β_i soit le poids de l'arborescence T_i enracinée en v_i, et k_i le nombre de terminaux de ce sous-arbre (voir la figure A.2).

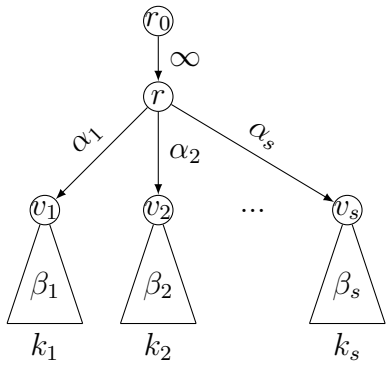


Figure A.2 – *Si la hauteur de T est $H + 1$, r est relié à des sous-arbres de hauteur au plus H.*

Soient φ_i et ε_i respectivement l'itération au début de laquelle tous les arcs de T_i sont saturés et le volume d'eau émis via v_i au début de cette itération. Ce flot va donc ici dans l'arc a_i, ou, si celui ci est saturé, dans b. On peut voir ε_i comme le flot contenu dans un arc (r_0, v_i) de poids infini au début de cette itération : il vérifie d'après le lemme 4.2.2, $\varepsilon_i = \sum_{j=1}^{\varphi_i - 1} k_j(r_0, v_i) \cdot (t_{i+1} - t_i) = \sum_{j=1}^{\varphi_i - 1} k_j(v_i) \cdot (t_{i+1} - t_i)$.

On va premièrement déterminer la valeur de t_φ, en fonction du signe de $\alpha_i - \varepsilon_i$ (intuitivement, ce signe permet de déterminer si l'arc a_i est saturé ou non quand tous les arcs de T_i sont saturés.). On déterminera ensuite la valeur de ε pour démontrer l'égalité souhaitée.

Détermination de la valeur de t_φ. Intéressons-nous à la première itération φ'_i au début de laquelle l'arbre T_i et l'arc a_i sont saturés, et plus précisément au temps $t_{\varphi'_i}$. La valeur de $t_{\varphi'_i}$ dépend du signe de $\alpha_i - \varepsilon_i$: soit a_i est déjà saturé lors de l'itération φ_i, soit il ne l'est pas.

Si $\alpha_i \leq \varepsilon_i$, montrons que $t_{\varphi_i} = t_{\varphi'_i}$. Si a_i est saturé au début de l'itération φ_i, cette égalité est vraie par définition de φ'_i. Si a_i n'est pas saturé, d'après le lemme 4.2.2, $f_{\varphi_i}(a_i) = \sum_{j=1}^{\varphi_i - 1} k_j(a_i) \cdot (t_{i+1} - t_i) = \sum_{j=1}^{\varphi_i - 1} k_j(v_i) \cdot (t_{i+1} - t_i) = \varepsilon_i \geq \alpha_i$. D'après le lemme 4.2.1, $\alpha_i \geq f_{\varphi_i}(a_i)$. Donc $\alpha_i = \varepsilon_i = f_{\varphi_i}(a_i)$. Par définition, le temps $t_{\varphi_i}(a_i)$ restant avant saturation de a_i est

donc 0. Ainsi tant que l'arc a_i n'est pas saturé (plus précisément, choisi à la ligne 6 et ajouté à G_{SAT}), l'incrément de la variable t à la ligne 9 est 0. On en déduit donc que $t_{\varphi_i} = t_{\varphi'_i}$.

Sinon $\alpha_i > \varepsilon_i$. On montre de la même manière que $f_{\varphi_i}(a_i) = \varepsilon_i < \alpha_i$. D'après le lemme 4.2.1, l'arc a_i n'est donc pas saturé. L'itération φ'_i - 1 est donc l'itération où cet arc est saturé. D'après le lemme 4.2.2 :

$$f_{\varphi'_i}(a_i) = f_{\varphi_i}(a_i) + \sum_{j=\varphi_i}^{\varphi'_i-1}(k_j(a_i) \cdot (t_{i+1} - t_i))$$

Puisque tous les arcs de T_i sont saturés dès l'itération φ_i, alors $k_j(a_i) = k_j(v_i) = k_i$ pour tout $j \in [\![\varphi_i; \varphi'_i - 1]\!]$.

$$\begin{aligned} f_{\varphi'_i}(a_i) &= f_{\varphi_i}(a_i) + k_i \cdot \sum_{j=\varphi_i}^{\varphi'_i-1}(t_{i+1} - t_i) \\ f_{\varphi'_i}(a_i) &= f_{\varphi_i}(a_i) + k_i \cdot (t_{\varphi'_i} - t_{\varphi_i}) \\ t_{\varphi'_i} &= t_{\varphi_i} + \frac{f_{\varphi'_i}(a_i) - f_{\varphi_i}(a_i)}{k_i} \end{aligned}$$

Or, au début de l'itération φ'_i, a_i est saturé donc $f_{\varphi'_i}(a_i) = \alpha_i$. De plus, $f_{\varphi_i}(a_i) = \varepsilon_i$. On en déduit donc que :

$$t_{\varphi'_i} = t_{\varphi_i} + \frac{\alpha_i - \varepsilon_i}{k_i}$$

φ est l'itération où tous les arbres T_i et arcs a_i sont saturés, soit $\max_{i \leq s}(\varphi'_i)$. De même, puisque la variable t ne fait qu'augmenter au cours des itérations, $t_\varphi = \max_{i \leq s}(t_{\varphi'_i})$.

Détermination de la valeur de ε. Le volume ε du flot qui rentre dans l'arc b est la somme sur i des volumes émis par les feuilles des sous-arbres T_i entre l'itération où l'arc a_i est saturé et l'itération φ.

On va montrer que ce flot est découpé en deux parties : une partie $\sum_{i=1}^{s}(t_\varphi - t_{\varphi'_i}) \cdot k_i$ représentant tous les volumes passés au travers des arcs a_i une fois T_i complètement saturé, et $\sum_{i:\alpha_i \leq \varepsilon_i}(\varepsilon_i - \alpha_i)$ qui représente l'ensemble des volumes passés au travers des arcs a_i avant que T_i ne soit saturé. D'après le lemme 4.2.2 :

$$\varepsilon = \sum_{j=1}^{\varphi-1} k_j(b)(t_{j+1} - t_j)$$

Soit $\varphi_{a_i} - 1$ l'itération pendant laquelle l'arc a_i est saturé. On suppose sans perte de généralité que les nœuds v_i sont ordonnés par ordre de saturation des arcs a_i : $\varphi_{a_1} \leq \varphi_{a_2} \leq \cdots \leq \varphi_{a_s}$. Au début de l'itération φ_{a_i}, tous les arcs $a_1, a_2, \ldots, a_i$ sont saturés, donc quel que soit $j \in [\![\varphi_{a_i}; \varphi_{a_{i+1}} - 1]\!]$, $k_j(b) = k_j(v_1) + k_j(v_2) + \cdots + k_j(v_i)$.

$$\varepsilon = \sum_{j=1}^{\varphi-1} \left(\sum_{i:j \in [\![\varphi_{a_i}; \varphi_{a_{i+1}-1}]\!]} k_j(v_1) + k_j(v_2) + \cdots + k_j(v_i) \right) \cdot (t_{j+1} - t_j)$$

$$\varepsilon = \sum_{j=1}^{\varphi-1} \sum_{i:\varphi_{a_i} \leq j} k_j(v_i) \cdot (t_{j+1} - t_j)$$

Or, par définition, a_i est saturé au début de l'itération φ'_i, donc on vérifie $\varphi'_i \geq \varphi_{a_i}$.

$$\varepsilon = \sum_{j=1}^{\varphi-1} \left(\sum_{i:\varphi'_i \leq j} k_j(v_i) \cdot (t_{j+1} - t_j) + \sum_{i:\varphi_{a_i} \leq j < \varphi'_i} k_j(v_i) \cdot (t_{j+1} - t_j) \right)$$

Une fois l'itération $\varphi'_i \geq \varphi_i$ passée, tous les arcs de T_i sont saturés donc $k_j(v_i) = k_i$ pour tout $j \geq \varphi'_i$.

$$\varepsilon = \sum_{j=1}^{\varphi-1} \left(\sum_{i:\varphi'_i \leq j} k_i \cdot (t_{j+1} - t_j) + \sum_{i:\varphi_{a_i} \leq j < \varphi'_i} k_j(v_i) \cdot (t_{j+1} - t_j) \right)$$

Inversons les sommes. Soit $\mathbb{1}_f$, où f est une formule booléenne, la fonction indicatrice qui vaut 1 si f est vrai et 0 sinon.

$$\varepsilon = \sum_{j=1}^{\varphi-1} \left(\sum_{i=1}^{s} \mathbb{1}_{\varphi'_i \leq j} \cdot k_i \cdot (t_{j+1} - t_j) + \sum_{i=1}^{s} \mathbb{1}_{\varphi_{a_i} \leq j < \varphi'_i} \cdot k_j(v_i) \cdot (t_{j+1} - t_j) \right)$$

$$\varepsilon = \sum_{i=1}^{s} \sum_{j=1}^{\varphi-1} \mathbb{1}_{\varphi'_i \leq j} \cdot k_i \cdot (t_{j+1} - t_j) + \sum_{i=1}^{s} \sum_{j=1}^{\varphi-1} \mathbb{1}_{\varphi_{a_i} \leq j < \varphi'_i} \cdot k_j(v_i) \cdot (t_{j+1} - t_j)$$

$$\varepsilon = \sum_{i=1}^{s} \sum_{j=\varphi'_i}^{\varphi-1} k_i \cdot (t_{j+1} - t_j) \sum_{i=1}^{s} \sum_{j=\varphi_{a_i}}^{\varphi'_i - 1} k_j(v_i) \cdot (t_{j+1} - t_j)$$

$$\varepsilon = \sum_{i=1}^{s} k_i \cdot (t_\varphi - t_{\varphi'_i}) + \sum_{i=1}^{s} \sum_{j=\varphi_{a_i}}^{\varphi'_i - 1} k_j(v_i) \cdot (t_{j+1} - t_j)$$

On rappelle que le membre de droite est nul si $\alpha_i > \varepsilon_i$, car alors $\varphi'_i = \varphi_{a_i}$.

$$\varepsilon = \sum_{i=1}^{s} (t_\varphi - t_{\varphi'_i}) \cdot k_i + \sum_{i:\alpha_i \leq \varepsilon_i} \sum_{j=\varphi_{a_i}}^{\varphi'_i - 1} k_j(v_i) \cdot (t_{j+1} - t_j)$$

De plus, si $\alpha_i \leq \varepsilon_i$, nous avons montré que $t_{\varphi'_i} = t_\varphi$: pour tout $j \in [\![\varphi; \varphi'_i - 1]\!]$, $t_{j+1} - t_j = 0$.

$$\varepsilon = \sum_{i=1}^{s} (t_\varphi - t_{\varphi'_i}) \cdot k_i + \sum_{i:\alpha_i \leq \varepsilon_i} \sum_{j=\varphi_{a_i}}^{\varphi_i - 1} k_j(v_i) \cdot (t_{j+1} - t_j)$$

$$\varepsilon = \sum_{i=1}^{s} (t_\varphi - t_{\varphi'_i}) \cdot k_i + \sum_{i:\alpha_i \leq \varepsilon_i} \left(\sum_{j=1}^{\varphi_i - 1} k_j(v_i) \cdot (t_{j+1} - t_j) - \sum_{j=1}^{\varphi_{a_i} - 1} k_j(v_i) \cdot (t_{j+1} - t_j) \right)$$

D'après le lemme 4.2.2, on vérifie : $\varepsilon_i = \sum_{j=1}^{\varphi_i - 1} k_j(v_i) \cdot (t_{i+1} - t_i)$ et $\alpha_i = \sum_{j=1}^{\varphi_{a_i} - 1} k_j(v_i) \cdot (t_{j+1} - t_j)$.

$$\varepsilon = \sum_{i=1}^{s} (t_\varphi - t_{\varphi'_i}) \cdot k_i + \sum_{i:\alpha_i \leq \varepsilon_i} (\varepsilon_i - \alpha_i)$$

Il ne nous reste qu'à démontrer l'égalité souhaitée.

$$\varepsilon = \sum_{i=1}^{s} (t_\varphi - t_{\varphi'_i}) \cdot k_i + \sum_{i:\alpha_i \leq \varepsilon_i} (\varepsilon_i - \alpha_i)$$

$$\varepsilon = t_\varphi \cdot k - \sum_{i=1}^{s} t_{\varphi'_i} \cdot k_i + \sum_{i:\alpha_i \leq \varepsilon_i} (\varepsilon_i - \alpha_i)$$

On rappelle la valeur de t'_{φ_i} : si $\alpha_i \leq \varepsilon_i$, alors $t'_{\varphi_i} = t_\varphi$ et, si $\alpha_i > \varepsilon_i$, alors $t'_{\varphi_i} = t_\varphi + \frac{\alpha_i - \varepsilon_i}{k_i}$.

$$\varepsilon = t_\varphi \cdot k - \sum_{i=1}^{s} t_{\varphi_i} \cdot k_i - \sum_{i:\alpha_i > \varepsilon_i} \frac{\alpha_i - \varepsilon_i}{k_i} \cdot k_i + \sum_{i:\alpha_i \leq \varepsilon_i} (\varepsilon_i - \alpha_i)$$

$$t_\varphi \cdot k = \sum_{i=1}^{s} t_{\varphi_i} \cdot k_i + \sum_{i=1}^{s} (\alpha_i - \varepsilon_i) + \varepsilon$$

D'après l'hypothèse de récurrence, quel que soit $i \leq s$, $k_i \cdot t_{\varphi_i} = \beta_i + \varepsilon_i$, où β_i est le poids de l'arbre T_i.

$$t_\varphi \cdot k = \sum_{i=1}^{s} (\beta_i + \varepsilon_i) + \sum_{i=1}^{s} (\alpha_i - \varepsilon_i) + \varepsilon$$

$$t_\varphi \cdot k = \sum_{i=1}^{s} (\beta_i + \alpha_i) + \varepsilon$$

$$t_\varphi \cdot k = \omega + \varepsilon$$

Ceci conclut la récurrence, et le lemme est prouvé. □

Démonstration du théorème 4.2.9, page 46. Soit T une arborescence enracinée en r. Soit X l'ensemble des feuilles de T, toutes terminales. Soit T_0 le sous-arbre de T renvoyé par l'algorithme FLAC. Soient ω_0, k_0 et d_0 les poids, nombre de feuilles, et densité de T_0. Soit $\varphi_0 - 1$ l'itération pendant laquelle T_0 est renvoyé.

Théorème 4.2.9. *T_0 est un sous-arbre de T de densité minimum.*

Démonstration. Supposons qu'il existe un sous-arbre T_1 de T de densité $d_1 < d_0$.

Intéressons-nous à l'application de FLAC dans T_1 seul. Pour différencier simplement les applications de FLAC dans T_1 et dans T, on note t^1, $f^1(a)$, $k^1(a)$ et $t^1(a)$ les variables $t, f(a), k(a)$ et $t(a)$ si on applique FLAC à T_1 seul.

Soit T_2 le sous-arbre saturé de T_1 renvoyé par FLAC. D'après le lemme 4.2.8, la densité d_2 de T_2 est plus petite que d_1, et, d'après le lemme 4.2.7, la variable t^1 est égale à d_2 au moment où T_2 est saturé.

Revenons maintenant à la saturation des arcs de T. On va montrer que T_2 est saturé plus vite dans T que dans T_1 seul. Ce résultat suffit à démontrer le lemme, car la racine est alors nécessairement atteinte avant ou lors de la saturation de T_2, donc strictement avant l'itération φ_0 ce qui est en contradiction avec la définition de T_0.

Pour caractériser le fait que la saturation est plus rapide dans T que dans T_1 seul, nous allons démontrer que dès que la variable t est plus grande dans T que dans T_1, chaque arc de T_2 saturé dans T_1 l'est aussi dans T et inversement tout arc non saturé dans T n'est pas saturé dans T_1.

Soient $a_1, a_2, \dots a_s$ les arcs de T_2 classés par ordre de saturation lorsque l'on applique FLAC dans T_1 seul. Démontrons par récurrence sur i que chaque arc a_i vérifie l'assertion suivante : pour tous l et l^1 tels que $t^1_{l^1} < t_l$ alors

- soit a_i est saturé dans T strictement avant l'itération $l - 1$,
- soit a_i n'est pas saturé dans T_1 pendant ou avant l'itération $l^1 - 1$.

Initialisation de la récurrence. À la première itération, aucun arc ne contient de flot donc $f_1(a_1) = f^1_1(a_1) = 0$. Le premier arc à saturer dans T_2 est nécessairement un arc relié à un terminal. Puisque tout terminal est une feuille, aucun autre terminal ne peut remplir a_1 de flot avant sa saturation : $k(a_1)$ et $k^1(a_1)$ sont constamment égaux à 1. Donc on vérifie que $t_1(a_1) = t^1_1(a_1) = \omega(a_1)$.

Soient l et l^1 tels que $t^1_{l^1} < t_l$. Si a_1 est saturé dans T strictement avant l'itération $l - 1$, alors la propriété est satisfaite. Si ce n'est pas le cas, d'après le lemme 4.2.4, $t_l \leq t_l + t_l(a) \leq t_1 + t_1(a_1) = \omega(a_1)$. Donc $t^1_{l^1} < \omega(a_1)$. Ainsi a_1 ne peut être saturé pendant ou avant l'itération $l^1 - 1$.

Hérédité de la récurrence. Soit $i > 1$ tel que $a_i = (u, v)$ et tel que tout arc a_j, avec $j < i$, vérifie l'hypothèse de récurrence. Soient l et l^1 tels que $t^1_{l^1} < t_l$. Si a_i est saturé dans T strictement avant l'itération $l - 1$, alors la propriété est satisfaite. Supposons maintenant que ce n'est pas le cas et que, par l'absurde, a_i est saturé dans T_1 lors d'une itération $\varphi^1_i - 1 \leq l^1 - 1$.

D'où provient l'absurdité ? Puisque le flot doit aller plus vite dans T que dans T_1 seul, et puisque l'arc est saturé à l'instant $t^1_{l^1}$ dans T_1, il doit être saturé à tout instant ultérieur dans T. Or il ne l'est pas, c'est donc qu'il contient nécessairement plus de flot qu'il ne peut en contenir : $f_l(a_i) > \omega(a_i)$ ce qui est exclu par le lemme 4.2.1. Si cette inégalité est démontrée, alors la récurrence l'est et le lemme l'est aussi.

D'après le lemme 4.2.2, $\omega(a_i) = \sum_{j^1=1}^{\varphi_i^1-1} k_{j^1}^1(a_i)(t_{j^1+1}^1 - t_{j^1}^1)$, et $f_l(a_i) = \sum_{j=1}^{l-1} k_j(a_i)(t_{j+1} - t_j)$. Pour démontrer notre inégalité stricte, nous allons montrer que le débit $k(a_i)$ augmente *plus vite* que le débit $k^1(a_i)$. Pour être plus précis, nous allons démontrer que si $t_j > t_{j^1}^1$ alors $k_{j-1}(a_i) \geq k_{j^1}^1(a_i)$.

Puisque les arcs de T_2 sont classés par ordre de saturation dans T_1, chaque arc a_j, pour $j < i$ est saturé pendant une itération $\varphi_j^1 - 1$ telle que $\varphi_1^1 < \varphi_2^1 < \cdots < \varphi_i^1$. À chacune de ces itérations, la valeur de $k^1(a_i)$ peut augmenter. Lors des autres itérations, ce sont des arcs de $T_1 \backslash T_2$ qui sont saturés, et n'interviennent donc pas dans le calcul de $k^1(a_i)$, sinon ils appartiendraient à T_2. Donc $k^1(a_i)$ ne dépend que de la présence ou non des arcs $a_1, a_2, \ldots, a_{i-1}$ dans G_{SAT}^1 et vaut successivement $0 = k_1^1(a_i) \leq k_{\varphi_1^1}^1(a_i) \leq \cdots \leq k_{\varphi_{i-1}^1}^1(a_i)$.

Soient j et j^1 tels que $t_j > t_{j^1}^1$. Si a_1 appartient à G_{SAT}^1 au début de l'itération j^1, alors, a_1 a été saturé pendant ou avant l'itération $j^1 - 1$. Donc, par hypothèse de récurrence, a_1 est saturé dans T strictement avant l'itération $j - 1$. Ainsi, au début de l'itération $j - 1$, le graphe G_{SAT} contient l'arc a_1. De même avec les arcs $a_2, a_3, \ldots, a_{i-1}$. Donc le nombre de terminaux auxquels est relié l'arc a_i est plus élevé dans G_{SAT}, au début de l'itération $j - 1$, que dans G_{SAT}^1, au début de l'itération j^1 : $k_{j-1}(a_i) \geq k_{j^1}^1(a_i)$.

Posons $\varphi_0 = 1$. Pour tout $j \in [\![1; i]\!]$, posons φ_j la plus petite itération vérifiant $t_{\varphi_j} > t_{\varphi_j^1}^1$. Par définition, on vérifie donc $t_{(\varphi_j)-1} \leq t_{\varphi_j^1}^1$. On sait :

- premièrement que l'itération φ_j existe car $t_l > t_{l^1}^1 \geq t_{\varphi_j^1}^1$: au pire $\varphi_j = l$;
- deuxièmement que l'itération $\varphi_j - 1$ existe, sans quoi $\varphi_j = 1$; or $t_1 = t_1^1 = 0 \leq t_{\varphi_j^1}^1 < t_{\varphi_j}$.

Puisque $t_{\varphi_j^1}^1$ est une fonction croissante de j, alors $1 = \varphi_0 \leq \varphi_1 \leq \varphi_2 \leq \cdots \leq \varphi_i$.

Nous allons maintenant démontrer que $f_l(a_i) > \omega(a_1)$. La figure A.3 explique intuitivement pourquoi cette inégalité est vérifiée.

D'après le lemme 4.2.2 :

$$
\begin{aligned}
f_l(a_i) &= \sum_{j=1}^{l-1} k_j(a_i) \cdot (t_{j+1} - t_j) \\
f_l(a_i) &\geq \sum_{j=\varphi_1-1}^{\varphi_i-1} k_j(a_i) \cdot (t_{j+1} - t_j) \\
f_l(a_i) &\geq \sum_{p=1}^{i-1} \left(\sum_{j=\varphi_p-1}^{\varphi_{p+1}-2} k_j(a_i) \cdot (t_{j+1} - t_j) \right) + k_{\varphi_i-1}(a_i) \cdot (t_{\varphi_i} - t_{\varphi_i-1})
\end{aligned}
$$

La fonction $k_j(a_i)$ est une fonction croissante de j.

$$
f_l(a_i) \geq \sum_{p=1}^{i-1} \left(k_{\varphi_p-1}(a_i) \cdot \sum_{j=\varphi_p-1}^{\varphi_{p+1}-2} (t_{j+1} - t_j) \right) + k_{\varphi_{i-1}-1}(a_i) \cdot (t_{\varphi_i} - t_{\varphi_i-1})
$$

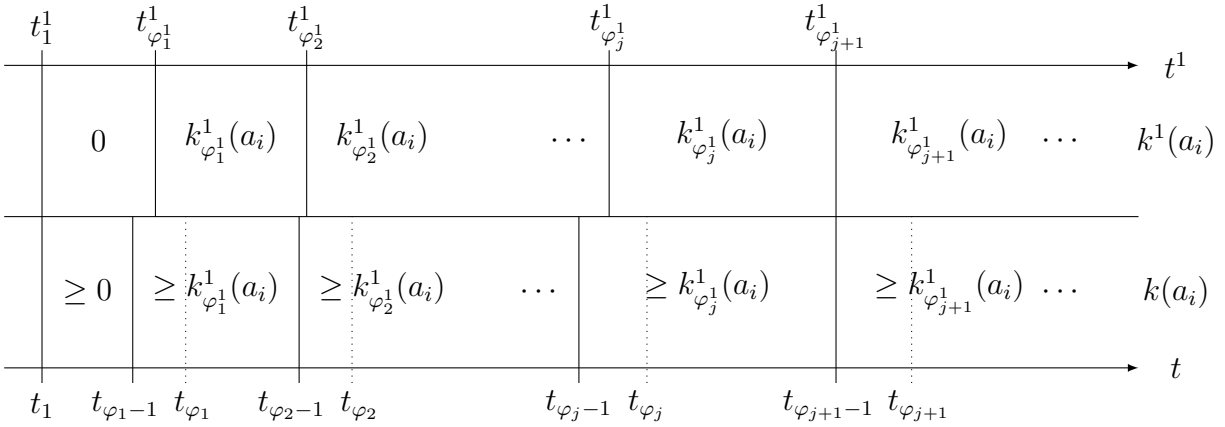


FIGURE A.3 – *Cette illustration explique intuitivement pourquoi $f_l(a_i) > \omega(a_1)$. Les deux lignes de temps représentent les valeurs des variables t et t^1 au cours des itérations. On a représenté les itérations φ_j^1, φ_j et $\varphi_j - 1$ vérifiant, par définition de φ_j, $t_{(\varphi_j)-1} \leq t_{\varphi_j^1}^1 < t_{\varphi_j}$. Entre les deux lignes sont marquées les valeurs du débit de a_i dans T_1 et une borne inférieure du débit de a_i dans T. Ce que l'on observe sur cette figure, c'est que lorsque le débit dans T_1 augmente, le débit dans T a déjà augmenté au moins autant. Par conséquent, le flot dans a_i est donc au moins aussi élevé dans T que dans T_1. Ceci montre que $f_l(a_i) \geq \omega(a_1)$. L'inégalité stricte est donnée par le fait que $t_{\varphi_i} > t_{\varphi_i^1}^1$.*

Nous avons montré que pour tous j et j^1 tels que $t_j > t_{j^1}^1$, $k_{j-1}(a_i) \geq k_{j^1}^1(a_i)$. Or, pour tout $p \in [\![1; i]\!]$, $t_{\varphi_p} > t_{\varphi_p^1}^1$.

$$f_l(a_i) \geq \sum_{p=1}^{i-1} \left(k_{\varphi_p^1}^1(a_i) \cdot \sum_{j=\varphi_p-1}^{\varphi_{p+1}-2} (t_{j+1} - t_j) \right) + k_{\varphi_{i-1}^1}^1(a_i) \cdot (t_{\varphi_i} - t_{\varphi_i-1})$$

$$f_l(a_i) \geq \sum_{p=1}^{i-1} \left(k_{\varphi_p^1}^1(a_i) \cdot (t_{\varphi_{p+1}-1} - t_{\varphi_p-1}) \right) + k_{\varphi_{i-1}^1}^1(a_i) \cdot (t_{\varphi_i} - t_{\varphi_i-1})$$

$$f_l(a_i) \geq \sum_{p=1}^{i-1} \left(k_{\varphi_p^1}^1(a_i) \cdot (t_{\varphi_{p+1}-1} - t_{\varphi_p^1}^1 + t_{\varphi_p^1}^1 - t_{\varphi_p-1}) \right) + k_{\varphi_{i-1}^1}^1(a_i) \cdot (t_{\varphi_i} - t_{\varphi_i-1})$$

$$f_l(a_i) \geq \sum_{p=1}^{i-1} \left(k_{\varphi_p^1}^1(a_i) \cdot (t_{\varphi_{p+1}-1} - t_{\varphi_p^1}^1) + k_{\varphi_p^1}^1(a_i) \cdot (t_{\varphi_p^1}^1 - t_{\varphi_p-1}) \right) + k_{\varphi_{i-1}^1}^1(a_i) \cdot (t_{\varphi_i} - t_{\varphi_i-1})$$

La fonction $k_{j^1}^1(a_i)$ est une fonction croissante de j^1.

$$f_l(a_i) \geq \sum_{p=1}^{i-1} \left(k_{\varphi_p^1}^1(a_i) \cdot (t_{\varphi_{p+1}-1} - t_{\varphi_p^1}^1) + k_{\varphi_{p-1}^1}^1(a_i) \cdot (t_{\varphi_p^1}^1 - t_{\varphi_p-1}) \right) + k_{\varphi_{i-1}^1}^1(a_i) \cdot (t_{\varphi_i} - t_{\varphi_i-1})$$

Sachant que $k^1_{\varphi^1_{p-1}}(a_i) = 0$ si $p = 1$, en décalant d'un cran le second élément de chaque terme de la somme, on peut réécrire la somme ainsi :

$$\begin{aligned} f_l(a_i) &\geq \sum_{p=1}^{i-2} \left(k^1_{\varphi^1_p}(a_i) \cdot (t_{\varphi_{p+1}-1} - t^1_{\varphi^1_p}) + k^1_{\varphi^1_p}(a_i) \cdot (t^1_{\varphi^1_{p+1}} - t_{\varphi_{p+1}-1}) \right) \\ &\quad + k^1_{\varphi^1_{i-1}}(a_i) \cdot (t_{\varphi_i - 1} - t^1_{\varphi^1_{i-1}}) + k^1_{\varphi^1_{i-1}}(a_i) \cdot (t_{\varphi_i} - t_{\varphi_i - 1}) \\ f_l(a_i) &\geq \sum_{p=1}^{i-2} \left(k^1_{\varphi^1_p}(a_i) \cdot (t^1_{\varphi^1_{p+1}} - t^1_{\varphi^1_p}) \right) + k^1_{\varphi^1_{i-1}}(a_i) \cdot (t_{\varphi_i} - t^1_{\varphi^1_{i-1}}) \end{aligned}$$

Par définition, $t_{\varphi_i} > t^1_{\varphi^1_i}$.

$$\begin{aligned} f_l(a_i) &> \sum_{p=1}^{i-2} \left(k^1_{\varphi^1_p}(a_i) \cdot (t^1_{\varphi^1_{p+1}} - t^1_{\varphi^1_p}) \right) + k^1_{\varphi^1_{i-1}}(a_i) \cdot (t^1_{\varphi^1_i} - t^1_{\varphi^1_{i-1}}) \\ f_l(a_i) &> \sum_{p=1}^{i-1} \left(k^1_{\varphi^1_p}(a_i) \cdot (t^1_{\varphi^1_{p+1}} - t^1_{\varphi^1_p}) \right) \end{aligned}$$

Enfin, puisque, $k^1(a_i)$ vaut successivement $0 = k^1_1(a_i) \leq k^1_{\varphi^1_1}(a_i) \leq \cdots \leq k^1_{\varphi^1_{i-1}}(a_i)$, alors le terme de droite peut se réécrire ainsi :

$$f_l(a_i) > \sum_{j=1}^{\varphi^1_{i-1}} \left(k^1_j(a_i) \cdot (t^1_{j+1} - t^1_j) \right)$$

D'après le lemme 4.2.2, le terme de droite est la définition de $\omega(a_i)$.

$$f_l(a_i) > \omega(a_i)$$

Ceci conclue et prouve la récurrence. a_s est un arc sortant de r, vérifiant la priopriété de récurrence, et la variable $t^1_{\varphi^1_s}$ vaut $d(T_2)$. C'est strictement plus petit que t_{φ_0}. Puisque a_s est saturé pendant l'itération φ^1_s, par la propriété de récurrence, a_s est saturé dans T strictement avant l'itération $\varphi_0 - 1$. Ce qui est exclu puisque la première itération où un arc sortant de r est saturé est φ_0 par définition. □

Exactitude de l'implémentation FLAC 2. Dans le chapitre 4, nous avons décrit deux implémentations : l'algorithme 4, FLAC, en page 41 et l'algorithme 5, FLAC 2, en page 56. Nous souhaitons montrer que ces deux algorithmes renvoient les mêmes solutions réalisables pour une instance donnée. Ainsi, nous garantissons que les propriétés d'approximation de FLAC 2 sont les mêmes que celles de FLAC. Du fait de sa rapidité, il est préférable, en pratique, d'utiliser FLAC 2 plutôt que FLAC.

Nous allons démontrer que les deux implémentations choisissent à chaque itération le même arc. Elles effectuent donc les mêmes opérations de saturation et de marquage, et renvoient nécessairement les mêmes solutions.

Ce raisonnement fonctionne à condition que les deux algorithmes restent déterministes. Or une part d'aléatoire subsiste lorsque, dans l'expérience, deux arcs sont saturés au même instant, car un unique arc est choisi à chaque itération. Nous allons définir arbitrairement un ordre : $A = \{a_1, a_2, \ldots, a_m\}$. Dans l'algorithme 4, si deux arcs a_i et a_j vérifient $t(a_i) = t(a_j)$, et si l'algorithme doit choisir entre ces deux arcs à la ligne 6, il choisira celui de plus petit indice. De même si, dans la seconde implémentation, l'algorithme doit faire un choix pour ordonner deux nœuds dans $\mathcal{F}$, car ils ont même clé, alors il mettra en premier celui dont l'arc entrant non saturé et de poids minimum a le plus petit indice. Enfin, il procèdera de la même façon s'il doit faire un choix pour ordonner les arcs de $\Gamma^-(v)$, où v est un nœud du graphe.

Lemme A.1. *Au début de chaque itération, l'arc (u, v) qui est choisi par l'algorithme 4 et l'arc a_0 qui est choisi par l'algorithme 5 sont les mêmes.*

Démonstration. On fera référence à l'algorithme 4 par l'implémentation FLAC 1 ou plus simplement FLAC 1. On fera référence à l'algorithme 5 par l'implémentation FLAC 2 ou plus simplement FLAC 2. Chaque variable de chaque implémentation sera associée si nécessaire à la référence de son implémentation, mais aussi si nécessaire à l'itération à laquelle on fait référence (par exemple la variable t_i^1 ou t_i^2).

Soit a_i^1 l'arc choisi par FLAC 1 pendant l'itération i, et a_i^2 l'arc choisi par FLAC 2 pendant l'itération i. Démontrons par récurrence sur i que :

(i) a_i^1 et a_i^2 sont les mêmes arcs, et cet arc est marqué dans FLAC 1 si et seulement s'il est marqué dans FLAC 2,

(ii) au début de l'itération i, $t_i^1 = t_i^2$: le temps s'écoule de la même façon dans les deux implémentations,

(iii) au début de l'itération i, pour tout nœud v, $k_i^1(v) = k_i^2(v)$: le débit de chaque arc est le même dans les deux implémentations,

(iv) au début de l'itération i, pour chaque nœud v, $t_i^1 + t_i^1(a_v) = t_i^2(v)$: l'instant de saturation prévu par la durée $t_i^1(a_v)$ dans FLAC 1 et l'instant $t_i^2(v)$ dans FLAC 2 est le même, où a_v est au début de l'itération i l'arc entrant en v non saturé de plus petit poids.

Initialisation de la récurrence. Au début de la première itération de l'implémentation FLAC 1, chaque arc a un flot nul. Le débit de chaque arc entrant dans un terminal vaut 1 et le débit de chaque autre arc vaut 0. Dans l'implémentation FLAC 2, d'après l'initialisation ligne 4, le débit de chaque arc est le même dans FLAC 1 et FLAC 2, la propriété (iii) est donc vérifiée à la première itération. De plus, l'instant de saturation prévu de chaque arc a dans FLAC 1 est $t_1^1 + t_1^1(a) = t_1^1(a)$. Cet instant vaut $\omega(a)$ pour tout arc entrant dans un terminal et $+\infty$ pour tout autre arc. D'après l'initialisation dans FLAC 2, pour tout nœud v, si v est terminal alors l'arc a_v sera saturé à l'instant $t_1^2(v) = \omega(a_v)$, et, sinon, alors cet instant est $t_1^2(v) = +\infty$. Donc l'instant de saturation de a_v est le même dans FLAC 1 et FLAC 2. Ainsi, la propriété (iv) est vérifiée à la première itération.

L'arc a_1^1 a la plus petite valeur $t_1^1(a)$ qui existe parmi tous les arcs a. Donc a_1^1 minimise $\omega(a)$ parmi tous les arcs a qui entrent dans un terminal. En cas d'égalité, a_1^1 est l'arc qui a le plus petit indice. Dans l'implémentation FLAC 2, on sélectionne le nœud qui a la plus petite valeur $t_1^2(v)$ et son arc a_v entrant non saturé et de poids minimum. Alors ce nœud est un terminal et $t_1^2(v)$ est la plus petite valeur $\omega(a)$ parmi tout arc a entrant dans un terminal. En cas d'égalité, l'arc a_1^2 est celui qui a le plus petit indice. Donc il s'agit également de a_1^1. Ainsi, la propriété (i) est vérifiée à la première itération.

Enfin, la valeur de la variable t dans les deux implémentations est initialisée à 0. Par conséquent, la propriété (ii), et donc la propriété de récurrence sont démontrées pour la première itération.

Hérédité de la récurrence. Supposons que les deux implémentations aient effectué $i-1$ itérations vérifiant l'hypothèse de récurrence. Montrons que l'itération i la vérifie aussi.

On peut immédiatement remarquer qu'il existe un lien entre la propriété (ii) de l'itération i et la propriété (iv) de l'itération $i-1$: si, lors d'une itération $i-1$, l'instant de saturation prévu des arcs est le même dans les deux implémentations, c'est le cas de l'arc a_{i-1} qui est saturé à cette itération. Le temps s'écoule donc pareillement et la propriété (ii) est bien vérifiée à l'itération i.

Plus formellement, au début de l'itération $i-1$, $t^1_{i-1} = t^2_{i-1}$, et pour tout nœud v, $k^1_{i-1}(v) = k^2_{i-1}(v)$ et $t^1_{i-1} + t^1_{i-1}(a_v) = t^2_{i-1}(v)$. Enfin, $a^1_{i-1} = a^2_{i-1}$. Soit a_{i-1} cet arc et soient u_{i-1} et v_{i-1} tels que $a_{i-1} = (u_{i-1}, v_{i-1})$. D'après la ligne 17 de l'implémentation FLAC 2, a_{i-1} est un arc a_v pour un nœud v, on en déduit que $t^1_{i-1} + t^1_{i-1}(a_{i-1}) = t^2_{i-1}(v_{i-1})$. Or $t^1_i = t^1_{i-1} + t^1_{i-1}(a_{i-1})$ et $t^2_i = t^2_{i-1}(v_{i-1})$. Donc la valeur de la variable t est la même dans FLAC 1 et FLAC 2. La propriété (ii) est vérifiée.

On sait que l'arc a_{i-1} est marqué dans FLAC 1 si et seulement s'il est marqué dans FLAC 2. Supposons, dans un premier temps, que cet arc soit marqué. Alors, le débit à travers les nœuds ne change pas dans les deux implémentations : la propriété (iii) reste vérifiée à l'itération i. Donc, dans FLAC 1, la durée prévue de saturation de chaque arc est réduite de $t^1_{i-1}(a_{i-1})$ à la fin de l'itération $i-1$, soit exactement la durée ajoutée à la variable t^1. Ainsi, l'instant de saturation prévu n'est modifié pour aucun arc. Quel que soit $v \neq v_{i-1}$, le temps de saturation $t^2(v)$ ne change pas non plus. Pour tous ces nœuds, on vérifie donc toujours $t^1_i + t^1_i(a_v) = t^2_i(v)$.

On va maintenant traiter le cas particulier du nœud v_{i-1}. Soit a'_{i-1} le second arc de $\Gamma^-(v_{i-1})$ au début de l'itération $i-1$. S'il n'existe pas, alors ce nœud n'est plus concerné par la dernière propriété de récurrence. Supposons qu'il existe, alors $t^2_i(v_{i-1}) = t^2_i + \frac{\omega(a'_{i-1}) - \omega(a_{i-1})}{k^2_i(v_{i-1})}$. Par conséquence, on a :

$$t^2_i(v_{i-1}) - (t^1_i + t^1_i(a'_{i-1})) = t^2_i + \frac{\omega(a'_{i-1}) - \omega(a_{i-1})}{k^2_i(v_{i-1})} - t^1_i - \frac{\omega(a'_{i-1}) - f^1_i(a'_{i-1})}{k^1_i(v_{i-1})}$$

Or, les propriétés (ii) et (iii) sont vérifiées : $t^2_i = t^1_i$ et $k^1_i(v_{i-1})$ et $k^2_i(v_{i-1})$.

$$\begin{aligned} t^2_i(v_{i-1}) - (t^1_i + t^1_i(a'_{i-1})) &= \frac{\omega(a'_{i-1}) - \omega(a_{i-1})}{k^2_i(v_{i-1})} - \frac{\omega(a'_{i-1}) - f^1_i(a'_{i-1})}{k^2_i(v_{i-1})} \\ &= \frac{f^1_i(a'_{i-1}) - \omega(a_{i-1})}{k^2_i(v_{i-1})} \end{aligned}$$

Or, d'après le lemme 4.2.1, le flot de a_{i-1} est égal à son poids. Et puisque le débit dans les arcs entrant dans un même nœud est toujours le même, d'après le lemme 4.2.2, le poids de a_{i-1} est aussi le flot contenu dans a'_{i-1}. Donc l'instant de saturation prévu de a'_{i-1} est le même dans les deux implémentations, et la propriété (iv) est vérifiée.

Ainsi, pour tout nœud v, au début de l'itération i, l'instant de saturation prévu de a_v est le même dans les deux implémentations. Par conséquent, on peut en déduire, comme on l'a fait dans l'initialisation de la récurrence, que l'arc choisi dans chaque implémentation

pendant l'itération i est le même. Ceci implique que la propriété (i), et donc la propriété de récurrence, sont vérifiées si l'arc est marqué à l'itération $i-1$.

Supposons maintenant que l'arc a_{i-1} ne soit pas marqué. Dans FLAC 1, le débit de chaque nœud w augmente si, en saturant a_{i-1}, il existe un nouveau chemin reliant w à un terminal. Ce chemin emprunte donc a_{i-1} : les nœuds w concernés sont ceux qui appartiennent à $T_{u_{i-1}}$ et le nombre de terminaux ajoutés est $k^1_{i-1}(v_{i-1}) = k^2_{i-1}(v_{i-1})$. Pour les autres nœuds le débit reste inchangé. Donc le débit augmente de la même manière dans FLAC 1 et FLAC 2 : la propriété (iii) est vérifiée.

Dans FLAC 1 et FLAC 2, l'instant de saturation de tout arc a_w entrant dans un nœud w qui n'appartient pas à $T_{u_{i-1}}$ (y compris v_{i-1}) évolue de la même manière si l'arc est marqué ou non. On sait donc que, pour ces nœuds, l'instant de saturation de a_w est identique au début de l'itération i. On va donc s'intéresser au cas où $w \in T_{u_{i-1}}$. Dans FLAC 1, le débit et le flot dans l'arc a_w augmentent : la durée avant saturation diminue et l'instant de saturation est avancé. Dans FLAC 2, l'instant de saturation est également avancé.

Premièrement, si $k^1_{i-1}(w) = k^2_{i-1}(w) = 0$, alors aucun flot n'est entré en a_w et la durée de saturation prévue au début de l'itération $i-1$ est $+\infty$. Dans FLAC 1, l'instant de saturation prévu au début de l'itération i est donc $t^1_i + \frac{\omega(a_w)}{k^1_i(v_{i_1})}$. Puisque, d'après la propriété (ii), $t^1_i = t^2_i$ et que, d'après la propriété (iii), le débit est le même pour tout nœud, d'après la ligne 33 dans FLAC 2, l'instant de saturation prévu de a_w est le même dans FLAC 1 et FLAC 2 et la propriété (iv) est vérifiée dans ce cas.

Deuxièmement, si $k^1_{i-1}(w) = k^2_{i-1}(w) \neq 0$, alors, d'après la ligne 35 dans FLAC 2, on vérifie $t^2_i(w) = t^2_i + \frac{(t^2_{i-1}(w) - t^2_i) \cdot k^2_{i-1}(w)}{k^2_i(w)}$.

$$t^2_i(w) - (t^1_i + t^1_i(a_w)) = \left(t^2_i + \frac{(t^2_{i-1}(w) - t^2_i) \cdot k^2_{i-1}(w)}{k^2_i(w)}\right) - \left(t^1_i + \frac{\omega(a_w) - f^1_i(a_w)}{k^1_i(w)}\right)$$

Les propriétés (ii) et (iii) sont vérifiées : $t^2_i = t^1_i$ et $k^1_i(w)$ et $k^2_i(w)$.

$$t^2_i(w) - (t^1_i + t^1_i(a_w)) = \left(t^1_i + \frac{(t^2_{i-1}(w) - t^1_i) \cdot k^1_{i-1}(w)}{k^1_i(w)}\right) - \left((t^1_i + \frac{\omega(a_w) - f^1_i(a_w)}{k^1_i(w)}\right)$$

$$t^2_i(w) - (t^1_i + t^1_i(a_w)) = \frac{(t^2_{i-1}(w) - t^1_i) \cdot k^1_{i-1}(w) - (\omega(a_w) - f^1_i(a_w))}{k^1_i(w)}$$

D'après la ligne 8 de FLAC 1, $f^1_i(a_w) = f^1_{i-1}(a_w) + k^1_{i-1}(w) \cdot t^1_{i-1}(a_{i-1})$.

$$t^2_i(w) - (t^1_i + t^1_i(a_w)) = \frac{(t^2_{i-1}(w) - t^1_i) \cdot k^1_{i-1}(w) - (\omega(a_w) - f^1_{i-1}(a_w) - k^1_{i-1}(w) \cdot t^1_{i-1}(a_{i-1}))}{k^1_i(w)}$$

$$t^2_i(w) - (t^1_i + t^1_i(a_w)) = \frac{(t^2_{i-1}(w) - t^1_i + t^1_{i-1}(a_{i-1})) \cdot k^1_{i-1}(w) - (\omega(a_w) - f^1_{i-1}(a_w))}{k^1_i(w)}$$

La propriété (iv) est vérifiée à l'itération $i-1$: $t^2_{i-1}(w) = t^1_{i-1}(a_w) + t^1_{i-1}$.

$$t^2_i(w) - (t^1_i + t^1_i(a_w)) = \frac{(t^1_{i-1}(a_w) + t^1_{i-1} - t^1_i + t^1_{i-1}(a_{i-1})) \cdot k^1_{i-1}(w) - (\omega(a_w) - f^1_{i-1}(a_w))}{k^1_i(w)}$$

Par définition de $t^1_{i-1}(a_w) \cdot k^1_{i-1}(w) = \omega(a_w) - f^1_{i-1}(a_w)$.

$$t^2_i(w) - (t^1_i + t^1_i(a_w)) = \frac{\omega(a_w) - f^1_{i-1}(a_w) + (t^1_{i-1} - t^1_i + t^1_{i-1}(a_{i-1})) \cdot k^1_{i-1}(w) - (\omega(a_w) - f^1_{i-1}(a_w))}{k^1_i(w)}$$

$$t^2_i(w) - (t^1_i + t^1_i(a_w)) = \frac{(t^1_{i-1} - t^1_i + t^1_{i-1}(a_{i-1})) \cdot k^1_{i-1}(w)}{k^1_i(w)}$$

D'après la ligne 9 de FLAC 1, $t^1_i = t^1_{i-1} + t^1_{i-1}(a_{i-1})$.

$$t^2_i(w) - (t^1_i + t^1_i(a_w)) = 0$$

Ainsi l'instant de saturation de tout arc a_w est identique au début de l'itération i, et la propriété (iv) est vérifiée dans ce cas aussi. On en déduit, comme on l'a fait dans l'initialisation de la récurrence, que l'arc a_i est le même dans les deux implémentations. Par conséquent, la propriété (i), et donc la propriété de récurrence, sont vérifiées si l'arc a_{i-1} non marqué à l'itération $i - 1$.

Ceci conclut la récurrence, et prouve le lemme. □

Ce lemme démontre que, à chaque itération, c'est le même arc qui est choisi par les deux implémentations, et c'est donc bien le même arbre qui est renvoyé lors de la dernière itération des deux implémentations.

Annexe B.

Reproductibilité de l'expérience

Cette annexe se focalise sur la reproductibilité des expériences du chapitre 4. L'ensemble des calculs ont été effectués par un programme en JAVA dont le code est disponible publiquement en ligne à l'adresse `https://github.com/mouton5000/DSTAlgoEvaluation`.

Le code contient en particulier :

- une représentation des instances du problème de Steiner ;
- un programme de traduction des instances de Steiner, depuis le format STP de la bibliotèque Steinlib ;
- les différents algorithmes testés.

B.1 Reproduction des expériences comparant les rapports d'approximation

Les instances des jeux de tests pour ces expériences ont été générées à partir des instances non orientées de la bibliothèque SteinLib, et sont toutes disponibles publiquement à l'adresse `http://steinlib.zib.de`. À chaque instance est associée, si elle est connue, le poids d'une solution optimale de l'instance. Certaines instances d'autres groupes étant corrompues ou incomplète, nous n'avons travaillé qu'avec les groupes suivants : WRP3, WRP4, ALUE, ALUT, DIW, DXMA, GAP, MSM, TAQ, LIN, SP, X, MC, I080 à I640, ES10FST à ES10000FST, B,C,D,E, P6E et P6Z.

Les tests concernaient trois classes d'instances : A, B et C. La première classe contient les instances transformées en instances biorientées et peut être construite sans nécessiter plus d'informations que celles données sur le site de la bibliothèque de SteinLib. Le deux autres classes ont besoin, pour construire une instance orientée à partir d'une instance non orientée, d'une solution optimale de l'instance non orientée. Une partie des instances ont été résolues de manière exacte et une solution optimale a pu être extraite de chacune de ces instances. Dans le dossier **SteinLibOptimalSolutions**, situé à la racine du projet, se trouvent un ensemble de fichiers, classés par groupe d'instances de la bibliothèque de SteinLib. À chaque instance est associé le poids prévu par la bibliothèque, ainsi qu'une solution optimale, si elle a été calculée. Il est possible de vérifier que cette solution est optimale en constatant que son poids est égal à celui prévu, en déterminant la racine de l'arborescence et en listant l'ensemble des terminaux couverts par cette solution.

À partir des solutions optimales et des instances, les instances des classes B et C peuvent être générées. Une partie de ces générations sont aléatoires (ajout et orientation d'arcs, choix d'une racine,...). Lorsque la période du générateur pseudo-aléatoire de JAVA pouvait être trop faible (par exemple lors du choix d'une permutation aléatoire), nous avons utilisé un autre générateur pseudo-aléatoire dont le code est également disponible dans le projet, dans le fichier **graphTheory/utils/HighRandomQuality.java**.

Enfin, le projet contient, pour chaque instance, les résultats numériques des tests, affichés sous forme synthétiques dans ce manuscrit.

B.2 Reproduction des expériences comparant les temps de calculs

Le projet contient une classe qui génère aléatoirement des instances du problème de Steiner avec l'algorithme décrit au début de l'expérience dans le chapitre 4 dans le fichier **graphTheory/generators/RandomSteinerDirectedGraphGenerator2.java**.

Le projet contient également, pour chaque instance que nous avons générée, les résultats numériques des tests, affichés sous forme synthétiques dans ce manuscrit.

Annexe C.

Détail des expériences

Cette annexe se focalise sur le détail des expériences relatives au rapport d'approximation du chapitre 4. Les premiers résultats fournis dans ce chapitre utilisent ensembles toutes les instances de la bibliothèque Steinlib, sans tenir compte des différences relatives à chaque instance. Nous les avons regroupées ici, en fonction des descriptions faites de chaque instance, pour observer plus en détail les résultats des expériences sur chacun des groupes.

Nous avons repris les classes d'instances du chapitre 4 : chaque instance non orientée de la bibliothèque a été transformée en trois instances orientées. La classe A est celle des instances biorientées. La classe B est celle des instances sans circuits. Et la classe C est celle des instances fortement connexes. On rappelle que seules 353 instances non orientées sur les 999 instances utilisées ont pu être transformées en instance de classes B et C.

Pour chaque groupe, nous traçons les mêmes courbes que celles des figures 4.9, 4.10 et 4.11, respectivement en pages 63, 65 et 66, si ce groupe contient suffisamment d'instances pour que cette représentation soit lisible (c'est-à-dire au moins 60 dans notre cas). S'il n'en contient pas assez, alors nous donnons directement sous forme de tableau les résultats renvoyés par les algorithmes.

C.1 Présentation des différents groupes d'instances

Nous commençons par présenter dans cette section les différents groupes de la bibliothèque Steinlib et comment ils sont regroupés dans la section suivante.

Nous étudions en premier lieu les groupes WRP3 et WRP4. Ces instances sont décrites comme étant des grilles contenant des trous, introduites dans [ZR03]. Chaque groupe contient une soixantaine d'instances dont le nombre de nœuds varie entre plusieurs centaines et quelques milliers. Leur particularité est la très grande valeur des solutions optimales. Ainsi, la plupart des algorithmes ont résolu presque toutes les instances avec une marge d'erreur inférieure à 1%.

Deuxièmement, nous étudions les groupes dits *VLSI*. Ces instances font partie des groupes ALUE, ALUT, DIW, DMXA, GAP, MSM, TAQ et LIN, introduits dans [KM98]. Elle sont, comme pour les groupes WRP3 et WRP4, des grilles contenant des trous, créées pour être appliquées dans le domaine Very Large Scale Integration (VLSI).

Troisièmement, nous regroupons toutes les instances des groupes ES10FST, ES20FST, ES30FST, ES40FST, ES50FST, ES60FST, ES70FST, ES80FST, ES90FST, ES100FST, ES250FST, ES500FST. Toutes ces instances sont des graphes rectilinéaires. À chaque groupe correspondent 15 instances dont le nombre de terminaux est précisé dans le nom du groupe. Nous n'étudions pas le groupe ES1000FST pour lequel CH_2 n'a pas renvoyé de résultat, ni l'unique instance de ES10000FST pour laquelle seuls Greedy$_{FLAC}$ et SHP 1 ont renvoyés un résultat.

On regroupe dans une quatrième sous-section les instances des groupes I080, I160, I320 et I640. Ces groupes contiennent chacun une centaine d'instances peu denses. Le nom du groupe indique le nombre de nœuds de l'instance. Le nombre de terminaux varie d'une dizaine à un quart du nombre de nœuds. Une description plus précise de ces instances peut être trouvée dans [DV97, KM98].

L'avant dernier regroupement est celui des groupes B,C,D et E. Ces quatre groupes contiennent chacun une vingtaine d'instances, introduites dans [Bea84, Bea89] dont le graphe est peu dense et dont les poids sont générés aléatoirement entre 1 et 10. Le nombre de nœuds de ces instances dépend du groupe choisi : moins de 100 nœuds pour le groupe B, 500 nœuds pour le groupe C, 1000 pour le groupe D et 2500 pour le groupe E.

Enfin nous donnons les derniers résultats concernant les groupes MC, SP, P6Z, P6E et X, introduits dans [KM98]. Le dernier groupe contient trois instances complètes et les quatre autres groupes contiennent des instances peu denses. Leurs résultats ne sont pas affichés sous forme de courbe car, même réunis, ils ne contiennent pas assez d'instances.

C.2 Résultats détaillés

C.2.1 Groupes WRP3 et WRP4

La figure C.1 trace les résultats pour la classe A. Attention, elle indique les résultats non par en pourcents, mais en dix millièmes de pourcent, du fait de la très grande proximité entre les solutions renvoyées par les algorithmes et les solutions optimales. Les résultats pour les classes B et C sont donnés dans les tableaux C.2 et C.3.

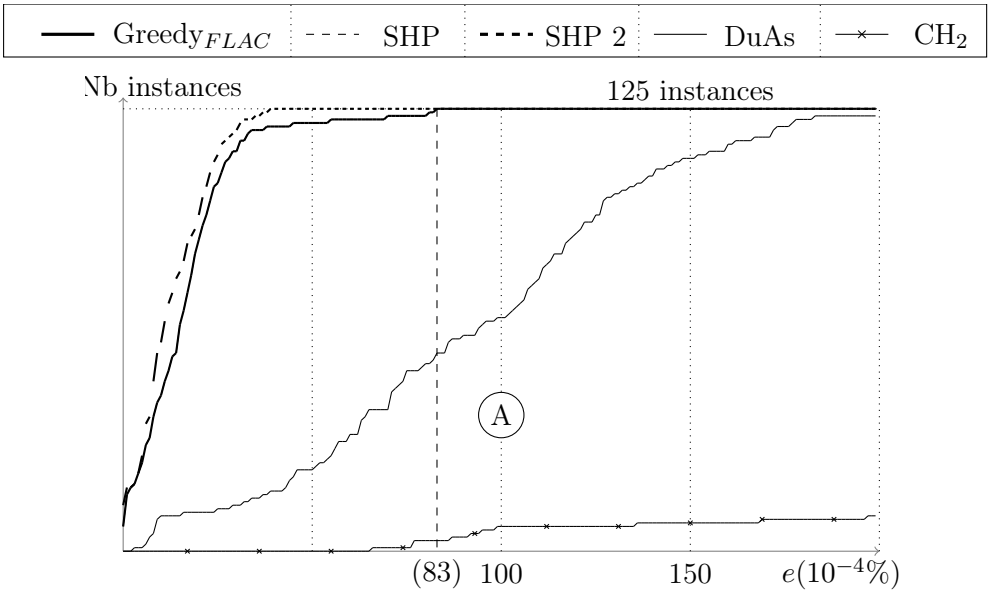


Figure C.1 – *Fonction de répartition des marges d'erreurs, en dix millièmes de pourcent, de chaque algorithme sur toutes les instances des groupes WRP3 et WRP4 de la classe A. Il est précisé en abscisse entre parenthèse la marge d'erreur maximale de Greedy$_{FLAC}$: l'écart entre une solution renvoyée par Greedy$_{FLAC}$ et une solution optimale ne dépasse pas 0.0083%. Attention, sur cette figure, les courbes des algorithmes Greedy$_{FLAC}$ et SHP 1 sont confondues car les résultats numériques sont très proches.*

Name	ω^*	Greedy$_{FLAC}$	SHP 1	SHP 2	DuAs	CH$_2$
wrp3-11ac	1100361	1,00	1,09	1,00	1,09	1,00
wrp3-12ac	1200237	1,00	1,08	1,08	1,00	1,00
wrp3-23ac	2300245	1,00	1,04	1,04	1,00	1,00
wrp4-11ac	1100179	1,00	1,00	1,00	1,00	1,00
wrp4-13ac	1300798	1,00	1,00	1,00	1,00	1,00

Table C.2 – *Rapport entre le poids de la solution renvoyée par chaque algorithme et le poids d'une solution optimale pour chaque instance des groupes WRP3 et WRP4 de la classe B. Attention, ces rapports sont arrondis à 2 chiffres après la virgule. Lorsque le rapport est égal à* 1*, il est noté* 1*. Si la valeur arrondie est égale à* 1*, alors elle est notée* 1.00*.*

Name	ω^*	Greedy$_{FLAC}$	SHP 1	SHP 2	DuAs	CH_2
wrp3-11st	1100361	1,00	1,00	1,00	1,00	1,00
wrp3-12st	1200237	1	1	1	1,00	1,00
wrp3-23st	2300245	1	1,00	1	1,00	1,00
wrp4-11st	1100179	1,00	1,00	1,00	1,00	1,00
wrp4-13st	1300798	1,00	1,00	1,00	1	1,00

Table C.3 – *Rapport entre le poids de la solution renvoyée par chaque algorithme et le poids d'une solution optimale pour chaque instance des groupes WRP3 et WRP4 de la classe C. Attention, ces rapports sont arrondis à 2 chiffres après la virgule. Lorsque le rapport est égal à* 1*, il est noté* 1*. Si la valeur arrondie est égale à* 1*, alors elle est notée* 1.00*.*

C.2.2 Groupes VLSI

La figure C.4 trace les résultats pour la classe A. Les résultats pour les classes B et C sont donnés dans les tableaux C.5 et C.6.

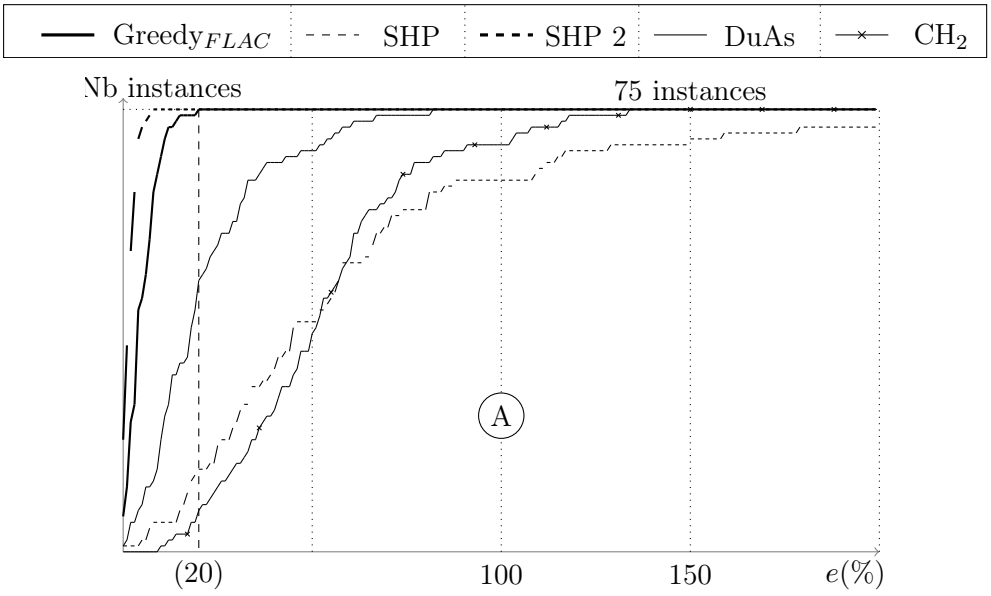


Figure C.4 – *Fonction de répartition des marges d'erreurs de chaque algorithme sur toutes les instances des groupes VLSI de la classe A. Il est précisé en abscisse entre parenthèse la marge d'erreur maximale de Greedy$_{FLAC}$: l'écart entre une solution renvoyée par Greedy$_{FLAC}$ et une solution optimale ne dépasse pas 20%.*

Name	ω^*	Greedy$_{FLAC}$	SHP 1	SHP 2	DuAs	CH$_2$
diw0250ac	350	1	1,33	1	1	1,71
diw0540ac	374	1	1,26	1	1	1,72
gap1500ac	254	1	1,34	1,03	1	1,68
gap2975ac	245	1,12	1,23	1,05	1,13	1,60
msm1844ac	188	1,06	1,15	1,14	1,19	1,62
msm4114ac	393	1,03	2,12	1,02	1	1,97
msm4224ac	311	1,03	1,36	1,04	1,03	1,87
msm4414ac	408	1,01	1,53	1,04	1	2,05
taq0920ac	210	1	1,73	1,15	1,10	2,57

Table C.5 – *Rapport entre le poids de la solution renvoyée par chaque algorithme et le poids d'une solution optimale pour chaque instance des groupes VLSI de la classe B. Attention, ces rapports sont arrondis à 2 chiffres après la virgule. Lorsque le rapport est égal à* 1*, il est noté* 1*. Si la valeur arrondie est égale à* 1*, alors elle est notée* 1.00*.*

Name	ω^*	Greedy$_{FLAC}$	SHP 1	SHP 2	DuAs	CH_2
diw0250st	350	1,01	1,08	1,01	1	1,18
diw0540st	374	1	1,11	1	1	1,24
gap1500st	254	1	1,03	1	1,04	1,68
gap2975st	245	1	1,11	1	1,02	1,47
msm1844st	188	1	1,04	1	1	1,45
msm4114st	393	1	1,40	1,03	1,03	1,76
msm4224st	311	1,02	1	1	1	1,36
msm4414st	408	1	1,21	1	1	1,72
taq0920st	210	1	1,86	1,04	1,07	1,96

Table C.6 – *Rapport entre le poids de la solution renvoyée par chaque algorithme et le poids d'une solution optimale pour chaque instance des groupes VLSI de la classe C. Attention, ces rapports sont arrondis à 2 chiffres après la virgule. Lorsque le rapport est égal à* 1*, il est noté* 1*. Si la valeur arrondie est égale à* 1*, alors elle est notée* 1.00*.*

C.2.3 Groupes ESXFST

La figure C.7 trace les résultats pour la classe A. Les résultats pour les classes B et C sont donnés dans les tableaux C.8 et C.9.

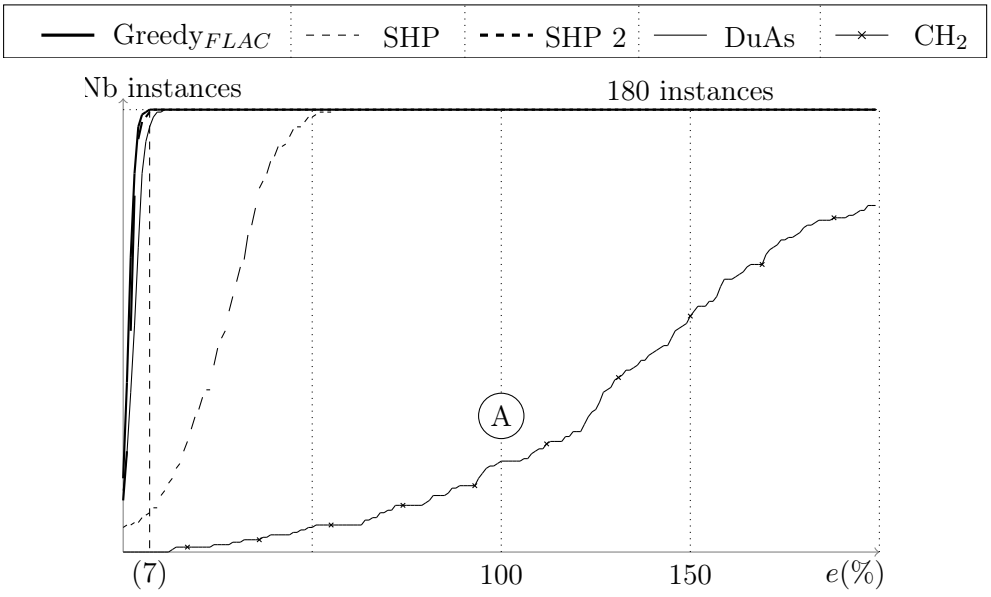


Figure C.7 – *Fonction de répartition des marges d'erreurs de chaque algorithme sur toutes les instances des groupes ESXFST de la classe A. Il est précisé en abscisse entre parenthèse la marge d'erreur maximale de Greedy$_{FLAC}$: l'écart entre une solution renvoyée par Greedy$_{FLAC}$ et une solution optimale ne dépasse pas 7%.*

Name	ω^*	Greedy$_{FLAC}$	SHP 1	SHP 2	DuAs	CH_2
es10fst10ac	23936095	1	1,14	1,01	1,01	1,76
es10fst11ac	22239535	1	1	1	1	1,65
es10fst12ac	19626318	1	1	1	1	2,02
es10fst13ac	19483914	1,05	1,18	1	1	1,76
es10fst14ac	21856128	1,11	1,20	1,16	1,03	1,58
es10fst15ac	18641924	1,03	1,10	1,09	1	1,71
es30fst02ac	40900061	1	1,20	1,03	1,01	2,84
es30fst05ac	41739748	1,01	1,13	1,13	1,02	2,31
es30fst07ac	43761391	1	1,11	1,24	1,00	3,53
es30fst08ac	41691217	1,03	1,32	1,05	1	3,52
es30fst09ac	37133658	1	1,02	1	1	2,79
es30fst10ac	42686610	1,00	1,06	1,07	1	4,13
es30fst12ac	38416720	1	1,10	1	1	3,07
es30fst13ac	37406646	1,10	1,23	1,06	1,02	2,78
es30fst14ac	42897025	1,00	1,06	1,02	1,00	2,54
es40fst02ac	46811310	1,04	1,25	1,10	1,03	3,45
es40fst04ac	45289864	1	1,02	1	1	4,77
es40fst06ac	49753385	1,02	1,32	1,13	1,05	4,47
es40fst07ac	45639009	1,00	1,12	1,02	1,03	4,63
es40fst12ac	43843378	1,01	1,11	1,03	1,00	3,19
es40fst13ac	51884545	1,03	1,19	1,07	1,02	2,90
es40fst14ac	49166952	1,03	1,27	1,07	1,07	2,79
es40fst15ac	50828067	1,01	1,27	1,03	1,01	3,29
es50fst04ac	51535766	1,02	1,22	1,07	1,02	4,07
es50fst05ac	55186015	1,03	1,20	1,12	1,00	4,26
es50fst08ac	53754708	1,01	1,12	1,03	1,00	4,57
es50fst11ac	52532923	1,03	1,32	1,03	1	3,60
es60fst08ac	58138178	1,03	1,23	1,02	1	4,73
es80fst12ac	65070089	1,01	1,15	1,04	1	5,39

Table C.8 – *Rapport entre le poids de la solution renvoyée par chaque algorithme et le poids d'une solution optimale pour chaque instance des groupes ESXFST de la classe B. Attention, ces rapports sont arrondis à 2 chiffres après la virgule. Lorsque le rapport est égal à* 1*, il est noté* 1*. Si la valeur arrondie est égale à* 1*, alors elle est notée* 1.00*.*

Name	ω^*	Greedy$_{FLAC}$	SHP 1	SHP 2	DuAs	CH_2
es10fst10st	23936095	1,05	1,05	1,05	1	1,41
es10fst11st	22239535	1	1	1	1	1,50
es10fst12st	19626318	1	1	1	1	1,56
es10fst13st	19483914	1	1,06	1	1	1,38
es10fst14st	21856128	1,03	1,17	1,03	1,08	1,49
es10fst15st	18641924	1,03	1,10	1,09	1	1,39
es30fst02st	40900061	1,01	1,12	1,02	1	1,98
es30fst05st	41739748	1,01	1,18	1,11	1	2,76
es30fst07st	43761391	1	1,05	1,05	1	2,63
es30fst08st	41691217	1,02	1,32	1,04	1	2,54
es30fst09st	37133658	1	1,04	1	1	2,44
es30fst10st	42686610	1	1	1	1	2,98
es30fst12st	38416720	1	1,21	1	1	2,30
es30fst13st	37406646	1,01	1,12	1,01	1,01	2,70
es30fst14st	42897025	1	1,02	1,01	1	2,14
es40fst02st	46811310	1,02	1,22	1,02	1,03	3,14
es40fst04st	45289864	1	1	1	1	2,28
es40fst06st	49753385	1,01	1,19	1,07	1,02	2,33
es40fst07st	45639009	1	1,04	1,02	1,03	2,14
es40fst12st	43843378	1,00	1,27	1,00	1	2,65
es40fst13st	51884545	1,03	1,13	1,05	1	2,73
es40fst14st	49166952	1	1,40	1,03	1	2,54
es40fst15st	50828067	1,01	1,79	1,02	1,01	3,12
es50fst04st	51535766	1,00	1,27	1,03	1	2,92
es50fst05st	55186015	1,01	1,41	1,03	1	3,04
es50fst08st	53754708	1,01	1,39	1,03	1	3,09
es50fst11st	52532923	1	1,67	1	1	3,12
es60fst08st	58138178	1,01	1,21	1,01	1	2,37
es80fst12st	65070089	1,01	1,13	1,01	1	2,95

Table C.9 – *Rapport entre le poids de la solution renvoyée par chaque algorithme et le poids d'une solution optimale pour chaque instance des groupes ESXFST de la classe C. Attention, ces rapports sont arrondis à 2 chiffres après la virgule. Lorsque le rapport est égal à* 1*, il est noté* 1*. Si la valeur arrondie est égale à* 1*, alors elle est notée* 1.00*.*

C.2.4 Groupes I080, I160, I320 et I640

Les résultats pour les classes A, B et C sont présentés respectivement en figures C.10, C.11 et C.12.

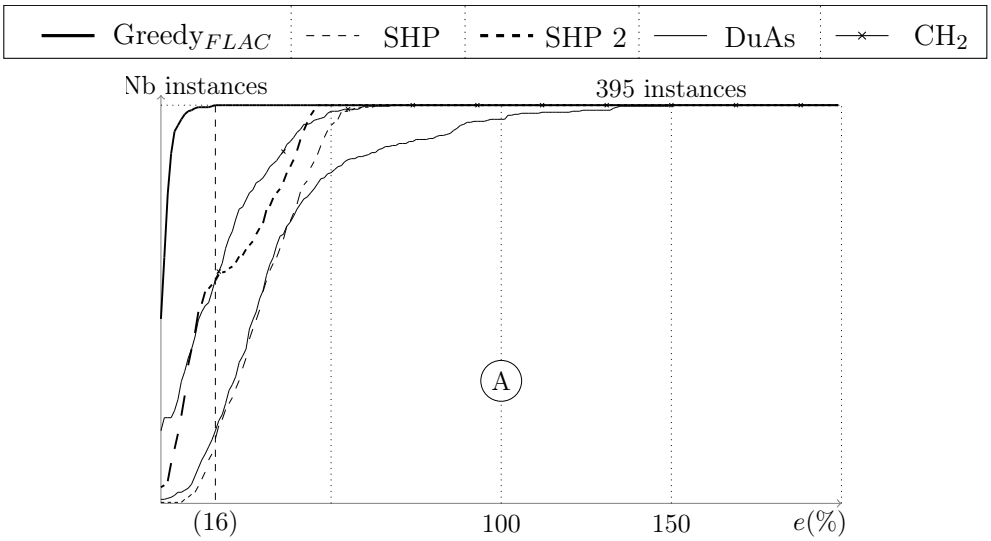


Figure C.10 – *Fonction de répartition des marges d'erreurs de chaque algorithme sur toutes les instances des groupes I080, I160, I320 et I640 de la classe A. Il est précisé en abscisse entre parenthèse la marge d'erreur maximale de Greedy$_{FLAC}$: l'écart entre une solution renvoyée par Greedy$_{FLAC}$ et une solution optimale ne dépasse pas 16%.*

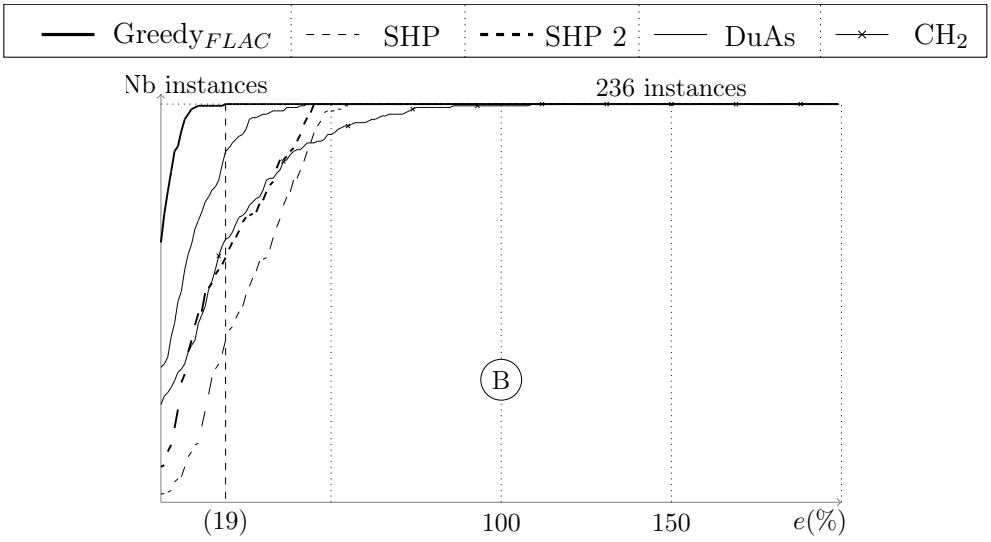


Figure C.11 – *Fonction de répartition des marges d'erreurs de chaque algorithme sur toutes les instances des groupes I080, I160, I320 et I640 de la classe B. Il est précisé en abscisse entre parenthèse la marge d'erreur maximale de Greedy$_{FLAC}$: l'écart entre une solution renvoyée par Greedy$_{FLAC}$ et une solution optimale ne dépasse pas 19%.*

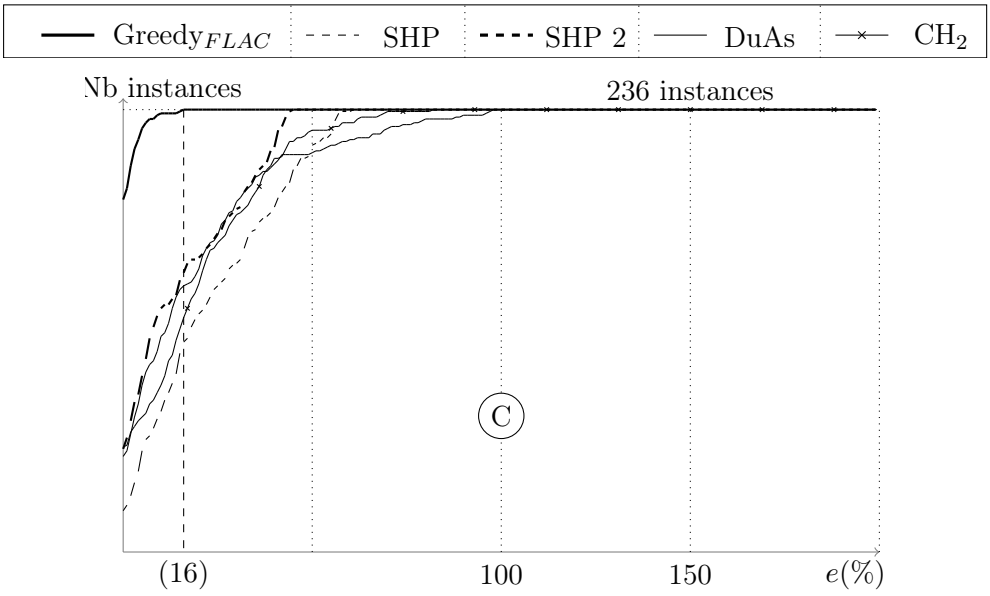


FIGURE C.12 – *Fonction de répartition des marges d'erreurs de chaque algorithme sur toutes les instances des groupes I080, I160, I320 et I640 de la classe C. Il est précisé en abscisse entre parenthèse la marge d'erreur maximale de Greedy$_{FLAC}$: l'écart entre une solution renvoyée par Greedy$_{FLAC}$ et une solution optimale ne dépasse pas 16%.*

C.2.5 Groupes B, C, D et E

La figure C.13 trace les résultats pour la classe A. Les résultats pour les classes B et C sont donnés dans les tableaux C.14 et C.15.

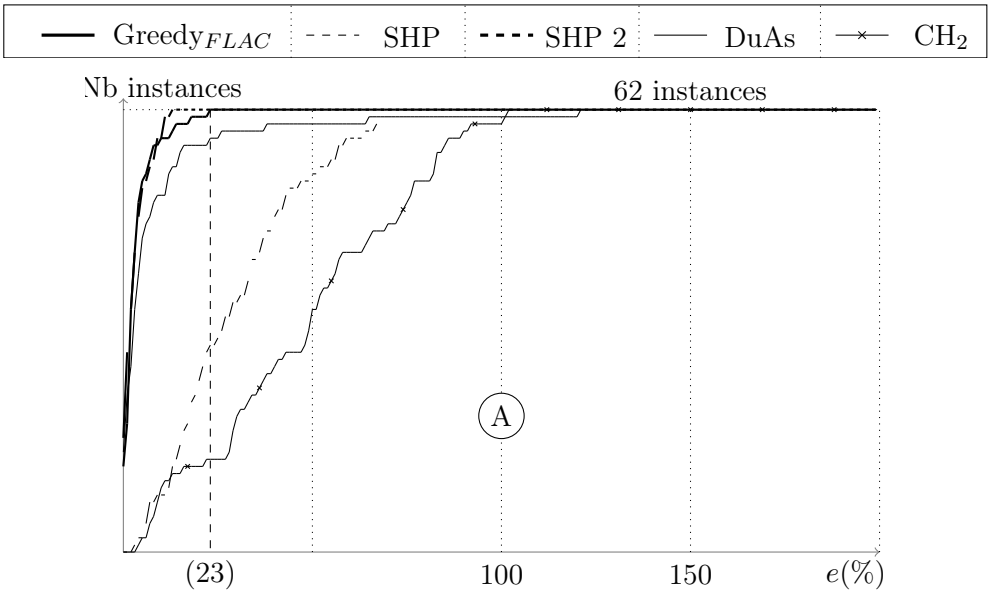


Figure C.13 – *Fonction de répartition des marges d'erreurs de chaque algorithme sur toutes les instances des groupes B, C, D et E de la classe A. Il est précisé en abscisse entre parenthèse la marge d'erreur maximale de Greedy$_{FLAC}$: l'écart entre une solution renvoyée par Greedy$_{FLAC}$ et une solution optimale ne dépasse pas 23%.*

Name	ω^*	Greedy$_{FLAC}$	SHP 1	SHP 2	DuAs	CH_2
b01ac	82	1	1	1	1	1,22
b02ac	83	1,08	1,49	1,08	1	1,82
b03ac	138	1	1,02	1	1	1,75
b04ac	59	1,07	1,27	1,07	1	1,27
b05ac	61	1,03	1,18	1	1	1,28
b06ac	122	1,01	1,10	1,04	1	1,92
b07ac	111	1	1,06	1	1	1,50
b08ac	104	1	1,36	1	1	2,43
b09ac	220	1	1,19	1,01	1	2,19
b10ac	86	1	1,30	1	1,12	1,53
b11ac	88	1	1,28	1	1	1,91
b12ac	174	1	1,29	1	1	1,69
b13ac	165	1,04	1,10	1,07	1,07	1,55
b14ac	235	1,04	1,11	1,06	1	1,71
b15ac	318	1	1,16	1,02	1	2,78
b16ac	127	1,16	1,41	1,02	1	1,47
b17ac	131	1,01	1,05	1,01	1	1,63
b18ac	218	1,00	1,53	1,03	1	2,08
c01ac	85	1	1,04	1,01	1	1,14
c02ac	144	1	1,26	1	1	1,49
c06ac	55	1	1,31	1	1	1,15
c07ac	102	1	1,17	1	1,13	1,43
c11ac	32	1	1,44	1,41	1,16	1,09
c12ac	46	1	1,17	1	1,04	1,28
c16ac	11	1	1,27	1	1,09	1,18
d01ac	106	1,10	1,10	1,10	1	1,18
d02ac	220	1	1,15	1,02	1	1,22
d06ac	67	1,18	1,42	1,27	1,04	1,16
d07ac	103	1	1,23	1	1	1,28
d11ac	29	1	1,59	1	1	1,24
d12ac	42	1,10	1,29	1,10	1,21	1,36

Table C.14 – *Rapport entre le poids de la solution renvoyée par chaque algorithme et le poids d'une solution optimale pour chaque instance des groupes B, C, D et E de la classe B. Attention, ces rapports sont arrondis à 2 chiffres après la virgule. Lorsque le rapport est égal à* 1*, il est noté* 1*. Si la valeur arrondie est égale à* 1*, alors elle est notée* 1.00*.*

Name	ω^*	Greedy$_{FLAC}$	SHP 1	SHP 2	DuAs	CH_2
b01st	82	1	1	1	1	1,22
b02st	83	1	1,20	1	1	1,49
b03st	138	1	1,12	1,01	1	2,04
b04st	59	1	1,39	1	1	1,44
b05st	61	1,02	1,08	1,02	1	1,20
b06st	122	1,03	1,36	1,11	1	1,83
b07st	111	1	1,05	1	1	1,59
b08st	104	1	1,27	1	1	1,45
b09st	220	1,01	1,09	1,01	1	1,67
b10st	86	1	1,07	1,07	1	1,34
b11st	88	1	1,17	1	1	1,61
b12st	174	1	1,28	1,03	1	1,80
b13st	165	1,04	1,07	1,07	1,07	1,22
b14st	235	1	1,02	1	1,02	1,60
b15st	318	1	1,18	1,03	1	1,75
b16st	127	1,03	1,09	1	1	1,25
b17st	131	1,01	1,15	1,01	1	1,63
b18st	218	1,00	1,26	1,01	1	1,68
c01st	85	1	1	1	1	1,21
c02st	144	1	1	1	1	1,56
c06st	55	1	1	1	1	1,22
c07st	102	1	1	1	1	1,35
c11st	32	1	1	1	1	1,03
c12st	46	1	1	1,04	1	1,15
c16st	11	1	1,18	1	1	1,27
d01st	106	1	1	1	1	1,10
d02st	220	1	1,06	1	1	1,15
d06st	67	1	1,09	1	1	1,09
d07st	103	1	1,26	1	1,22	1,51
d11st	29	1	1	1	1	1,28
d12st	42	1	2,10	1,10	1	1,64

Table C.15 – *Rapport entre le poids de la solution renvoyée par chaque algorithme et le poids d'une solution optimale pour chaque instance des groupes B, C, D et E de la classe C. Attention, ces rapports sont arrondis à 2 chiffres après la virgule. Lorsque le rapport est égal à* 1*, il est noté* 1*. Si la valeur arrondie est égale à* 1*, alors elle est notée* 1.00*.*

C.2.6 Autres groupes

Les résultats pour les classes A, B et C sont donnés dans les tableaux C.16, C.17 et C.18.

Name	ω^*	Greedy$_{FLAC}$	SHP 1	SHP 2	DuAs	CH_2
mc11bd	11689	1,03	1,18	1,01	1,01	2,12
mc13bd	92	1,02	1,28	1,27	1,15	1,16
mc2bd	71	1,06	1,32	1,21	1,18	1,21
mc3bd	47	1	1,06	1,45	1,26	1,02
mc7bd	3417	1,07	1,50	1,01	1,04	2,77
mc8bd	1566	1,03	1,21	1,03	1,02	1,93
berlin52bd	1044	1,08	2,93	1,02	1,04	1,63
brasil58bd	13655	1,01	2,66	1,00	1,02	1,62
antiwheel5bd	7	1	1	1	1	1,14
design432bd	9	1	1,11	1	1	1
oddcycle3bd	4	1	1	1	1	1
oddwheel3bd	5	1	1	1	1	1
se03bd	12	1	1	1	1	1
w13c29bd	508	1,03	1,29	1,11	1,26	1,80
w23c23bd	694	1,03	1,30	1,10	1,24	1,66
w3c571bd	2854	1	1,20	1,06	1,23	-0,00
p6e619bd	7485	1	1,10	1	1,03	1,01
p6e620bd	8746	1,04	1,30	1,01	1,13	1,21
p6e621bd	8688	1	1	1	1,06	1
p6e622bd	15972	1,04	1,59	1,04	1,28	1,33
p6e623bd	19496	1	1,62	1,03	1,09	1,49
p6e624bd	20246	1,04	2,32	1,01	1,08	2,03
p6e625bd	23078	1,03	1,75	1,02	1,08	1,75
p6e626bd	22346	1,03	1,28	1	1,13	1,35
p6e627bd	40647	1,03	2,01	1,00	1,07	2,20
p6e628bd	40008	1,03	2,19	1	1,04	2,17
p6e629bd	43287	1,02	1,88	1,01	1,04	2,37
p6e630bd	26125	1,09	1,38	1,02	1,24	1,46
p6e631bd	39067	1,02	1,66	1,08	1,19	1,71
p6e632bd	56217	1,05	1,90	1,01	1,24	2,22
p6e633bd	86268	1,03	1,88	1,00	1,02	2,63
p6z602bd	8083	1,04	1,66	1	1,11	1,27
p6z603bd	5022	1,34	1,08	1	1	1,09
p6z604bd	11397	1	1,45	1	1	1,31
p6z605bd	10355	1	1,17	1	1,08	1,70
p6z606bd	13048	1,05	1,35	1	1,01	1,37
p6z607bd	15358	1	1,45	1	1,02	1,60
p6z608bd	14439	1,08	1,28	1	1,01	1,45
p6z609bd	18263	1,08	1,39	1,01	1	1,57
p6z610bd	30161	1,04	1,40	1,02	1,02	1,75
p6z611bd	26903	1,04	1,47	1,01	1,02	1,92
p6z612bd	30258	1,03	1,35	1,00	1,04	1,91
p6z613bd	18429	1,06	1,26	1,01	1,17	1,40
p6z614bd	27276	1,06	1,56	1,03	1,07	1,65
p6z615bd	42474	1,05	1,33	1,03	1,11	1,89
p6z616bd	62263	1,06	1,34	1,01	1,04	2,30

Table C.16 – *Rapport entre le poids de la solution renvoyée par chaque algorithme et le poids d'une solution optimale pour chaque instance des groupes MC, X, SP, P6E et P6Z de la classe A. Attention, ces rapports sont arrondis à 2 chiffres après la virgule. Lorsque le rapport est égal à* 1*, il est noté* 1*. Si la valeur arrondie est égale à* 1*, alors elle est notée* 1.00*.*

Name	ω^*	Greedy$_{FLAC}$	SHP 1	SHP 2	DuAs	CH_2
mc3ac	47	1	1,11	1	1	1,06
berlin52ac	1044	1,09	2,82	1,04	1,01	1,54
brasil58ac	13655	1,04	3,56	1	1,01	1,77
antiwheel5ac	7	1	1	1	1,14	1,14
design432ac	9	1	1	1	1	1
oddcycle3ac	4	1	1	1	1	1
oddwheel3ac	5	1	1	1	1	1
se03ac	12	1	1	1	1	1
p6e619ac	7485	1	1,56	1	1	1,11
p6e620ac	8746	1,12	1,25	1,12	1	1,06
p6e621ac	8688	1,09	1,12	1,08	1,13	1,12
p6e622ac	15972	1,02	1,63	1	1,11	1,44
p6e623ac	19496	1	1,41	1,04	1	1,53
p6e624ac	20246	1	1,90	1,11	1	2,58
p6e625ac	23078	1,08	1,75	1	1	2,26
p6e626ac	22346	1,01	1,66	1,06	1,01	3,04
p6e627ac	40647	1,01	1,99	1,06	1,01	3,80
p6e628ac	40008	1,03	2,17	1,08	1	4,97
p6e630ac	26125	1	1,27	1,05	1,09	1,60
p6z602ac	8083	1	1,14	1	1,14	1,28
p6z603ac	5022	1	1,09	1	1	1,31
p6z604ac	11397	1	1,34	1	1,07	2,06
p6z605ac	10355	1	1	1	1	1,71
p6z606ac	13048	1	1,22	1	1,12	1,43
p6z607ac	15358	1	1,46	1	1	2,13
p6z608ac	14439	1,02	1,94	1,00	1,01	2,14
p6z609ac	18263	1	1,48	1,03	1	2,12
p6z610ac	30161	1,03	1,24	1,12	1,03	2,81
p6z611ac	26903	1,01	1,68	1	1	3,05
p6z612ac	30258	1,04	1,33	1,04	1	2,86
p6z613ac	18429	1,01	1,25	1,01	1,03	1,44
p6z614ac	27276	1,15	1,27	1,06	1,01	1,92
p6z615ac	42474	1,09	1,54	1,14	1,01	2,62
p6z616ac	62263	1,02	1,46	1,01	1	3,85

Table C.17 – *Rapport entre le poids de la solution renvoyée par chaque algorithme et le poids d'une solution optimale pour chaque instance des groupes MC, X, SP, P6E et P6Z de la classe B. Attention, ces rapports sont arrondis à 2 chiffres après la virgule. Lorsque le rapport est égal à* 1*, il est noté* 1*. Si la valeur arrondie est égale à* 1*, alors elle est notée* 1.00*.*

Name	ω^*	Greedy$_{FLAC}$	SHP 1	SHP 2	DuAs	CH_2
mc3st	47	1	1,19	1,15	1,21	1,06
berlin52st	1044	1,01	1,58	1,02	1,00	1,33
brasil58st	13655	1,06	2,36	1,05	1,04	1,75
antiwheel5st	7	1	1,14	1	1	1,14
design432st	9	1,11	1,11	1,11	1	1,11
oddcycle3st	4	1	1	1	1	1
oddwheel3st	5	1	1	1	1	1
se03st	12	1	1	1	1	1
p6e619st	7485	1	1,11	1	1,03	1,24
p6e620st	8746	1,12	1,12	1,12	1	1,06
p6e621st	8688	1	1,08	1,08	1,02	1,04
p6e622st	15972	1	1,12	1,08	1,07	1,61
p6e623st	19496	1	1,18	1	1,03	1,62
p6e624st	20246	1,01	1,45	1	1	1,91
p6e625st	23078	1	1,62	1	1,07	2,29
p6e626st	22346	1,04	1,28	1	1,01	2,15
p6e627st	40647	1	1,58	1,03	1	2,44
p6e628st	40008	1,02	1,24	1,01	1	2,43
p6e630st	26125	1,17	1	1	1,02	1,37
p6z602st	8083	1	1	1	1	1,51
p6z603st	5022	1	1,09	1	1	1,12
p6z604st	11397	1	1,35	1	1	1,87
p6z605st	10355	1	1,28	1	1	1,67
p6z606st	13048	1	1,09	1	1	1,47
p6z607st	15358	1	1,02	1	1	1,62
p6z608st	14439	1	1,04	1	1	1,84
p6z609st	18263	1	1,81	1,05	1	2,21
p6z610st	30161	1	1,27	1,01	1	2,39
p6z611st	26903	1	1,42	1	1	2,43
p6z612st	30258	1,02	1,19	1,03	1	2,28
p6z613st	18429	1,01	1,10	1,01	1	1,37
p6z614st	27276	1,08	1,19	1,02	1,02	1,70
p6z615st	42474	1	1,82	1,03	1	2,54
p6z616st	62263	1,00	1,26	1,02	1,01	2,74

Table C.18 – *Rapport entre le poids de la solution renvoyée par chaque algorithme et le poids d'une solution optimale pour chaque instance des groupes MC, X, SP, P6E et P6Z de la classe C. Attention, ces rapports sont arrondis à 2 chiffres après la virgule. Lorsque le rapport est égal à* 1*, il est noté* 1*. Si la valeur arrondie est égale à* 1*, alors elle est notée* 1.00*.*

Annexe D.

Notions de théorie de la complexité paramétrée

Cette annexe présente quelques notions fondamentales de la théorie de la complexité paramétrée nécessaires à la compréhension de ce manuscrit.

La complexité en temps ou en espace d'un algorithme est généralement donnée en fonction d'un paramètre décrivant la taille de l'instance donnée en entrée, mais il est parfois possible de définir cette complexité plus finement, à l'aide d'un ou plusieurs autres paramètres de l'instance, permettant ainsi de déterminer l'impact de ces paramètres sur la complexité de l'algorithme. Par exemple, dans le cas du problème de Steiner, un paramètre souvent utilisé est le nombre de terminaux. Généralement, le paramètre est plus petit que la taille de l'instance et on étudie la complexité paramétrée d'un problème pour déterminer s'il existe un algorithme rapide pour le résoudre quand le paramètre est petit. Cette théorie a été énoncée et développée dans [**?**].

Les définitions de cette annexe concernent par défaut la complexité en temps, mais peuvent être adaptées à la complexité en espace.

D.1 Les classes XP et FPT

Soient $\mathcal{A}$ un algorithme dont les entrées sont de taille n et de paramètre k, et $\mathcal{P}$ un problème de décision ou d'optimisation dont les instances sont de taille n et de paramètre k.

Définition 17. $\mathcal{A}$ est un algorithme XP vis-à-vis du paramètre k s'il existe une fonction $f : \mathbb{N} \to \mathbb{R}$ telle que la complexité en temps soit $O(n^{f(k)})$.

Définition 18. $\mathcal{P}$ appartient à la classe XP vis-à-vis du paramètre k s'il existe un algorithme de résolution exacte de $\mathcal{P}$ qui soit XP vis-à-vis du paramètre k.

Remarque 22. La classe XP est l'ensemble des problèmes paramétrés qui appartiennent à la classe P si on se restreint aux instances dont le paramètre est fixé. Plus généralement, la classe X$\mathcal{C}$ est l'ensemble des problèmes paramétrés qui appartiennent à la classe $\mathcal{C}$ si on se restreint aux instances dont le paramètre est fixé.

Définition 19. $\mathcal{A}$ est un algorithme FPT vis-à-vis du paramètre k s'il existe une constance réelle c, une fonction $f : \mathbb{N} \to \mathbb{R}$ tels que la complexité en temps soit $O(n^c \cdot f(k))$.

Définition 20. $\mathcal{P}$ appartient à la classe FPT vis-à-vis du paramètre k s'il existe un algorithme de résolution exacte de $\mathcal{P}$ qui soit FPT vis-à-vis du paramètre k.

On définit de manière similaire des problèmes qui peuvent être approchés en temps XP ou en temps FPT.

D.2 Réduction FPT et W-hiérarchie

La réduction FPT a été introduite pour classer les problèmes paramétrés.

Définition 21. Une réduction FPT d'un problème de décision $\mathcal{P}_1$ paramétré par k_1 vers un problème de décision $\mathcal{P}_2$ paramétré par k_2 est une transformation R de chaque instance $\mathcal{I}$ de $\mathcal{P}_1$ en instance $R(\mathcal{I})$ de $\mathcal{P}_2$ telle que :

- pour toute instance $\mathcal{I}$ de $\mathcal{P}_1$, $\mathcal{I}$ est une instance positive de $\mathcal{P}_1$ si et seulement si $R(\mathcal{I})$ est une instance positive de $\mathcal{P}_2$;
- la transformation R se fait en temps FPT vis-à-vis du paramètre k_1 ;
- il existe une fonction calculable $g : \mathbb{N} \to \mathbb{N}$ telle que pour tous paramètres k_1 d'une instance $\mathcal{I}$ de $\mathcal{P}_1$ et k_2 de l'instance $R(\mathcal{I})$, on vérifie $k_2 \leq g(k_1)$.

Théorème D.2.1. *S'il existe une réduction FPT d'un problème paramétré $\mathcal{P}_1$ vers un second problème paramétré $\mathcal{P}_2$ et si $\mathcal{P}_1$ est FPT, alors $\mathcal{P}_2$ est FPT.*

La W-hiérarchie définit un ensemble de classes de problèmes paramétrés ordonnées à l'aide de la réduction FPT. Pour tout entier t, la W-hiérarchie contient la classe W[t]. Elle est définie à partir d'un problème de satisfiabilité, nommé *satisfiabilité de circuit booléen de profondeur t de poids minimum*[1] ou WCS(t), paramétré par le nombre de variables vraies dans une solution optimale. Nous n'en donnons pas la définition dans cette annexe, mais le lecteur intéressé peut se reporter à [**?**] pour plus de détails.

Définition 22. La classe W[t] est définie comme l'ensemble des problèmes paramétrés pour lequel il existe une réduction FPT vers WCS(t), paramétré par le nombre de variables vraies dans une solution optimale.

Théorème D.2.2. *$FPT \subseteq W[1] \subseteq W[2] \subseteq \ldots \subseteq W[t] \subseteq XP$*

On conjecture que ces inclusions sont strictes. Intuitivement, un problème paramétré W[1] est plus "difficile" au sens de la complexité paramétrée, soit plus lent à résoudre, qu'un problème paramétré FPT.

Définition 23. Un problème paramétré $\mathcal{P}$ est W[t]-difficile si pour tout problème paramétré $\mathcal{P}'$ de la classe W[t], il existe une réduction de $\mathcal{P}'$ vers $\mathcal{P}$. Un problème paramétré $\mathcal{P}$ est W[t]-complet s'il appartient à la classe W[t] et s'il est W[t]-difficile.

De la même manière que pour les problèmes de la hiérarchie polynomiale, il est possible de déterminer si un problème paramétré est W[t]-complet s'il appartient à W[t] et s'il existe une réduction depuis un autre problème paramétré W[t]-complet. Par définition, on connait des problèmes W[t]-complets :

Théorème D.2.3. *Pour tout entier t, le problème WCS(t) est W[t]-complet.*

1. *Weighted weft t depth h circuit satisfiability*, en anglais