UNIVERSITÉ DU QUÉBEC À MONTRÉAL

SOUS-ARBRES INDUITS DANS LES GRAPHES SÉRIES-PARALLÈLES

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

MOUSSA ABDENBI

FÉVRIER 2019

REMERCIEMENTS

Je tiens à exprimer toute ma reconnaissance à mon directeur de mémoire Alexandre Blondin Massé et à ma codirectrice Halima Elbiaze. Je les remercie de m'avoir encadré, orienté, aidé et conseillé.

J'adresse mes remerciements à tous les professeurs, intervenants et toutes les personnes qui par leurs paroles, leurs écrits, leurs conseils et leurs critiques ont guidé mes réflexions durant mes études.

Je tiens aussi à remercier les membres de ma famille et mes amis pour leur soutien et leur présence. Une pensée particulière à Maria qui m'a toujours soutenu et écouté, merci !

À tous ces intervenants, je présente mon respect et ma gratitude.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

RÉSUMÉ

Dans ce mémoire nous nous intéressons au problème qui, étant donné un graphe simple G, interroge l'existence de sous-arbres induits de taille i ayant le plus grand nombre de feuilles. Un tel sous-arbre induit s'il existe est dit *pleinement feuillu*. Ce problème d'optimisation que nous nommons SAIPF pour sous-arbres induits pleinement feuillus, est associé au problème de décision SAIF, pour sous-arbres induits feuillus, que nous pouvons énoncer ainsi : étant donné un graphe simple G et deux entiers positifs i et ℓ est-ce qu'il existe un sous-arbre induit dans G avec i sommets et ℓ feuilles ? Nous démontrons que ce problème est NP-complet dans le cas général, ce qui fait du problème SAIPF un problème NP-difficile. Ces problèmes n'étant probablement pas résolubles avec des algorithmes en temps polynomial, comme alternative, nous nous intéressons plutôt à des familles de graphes spécifiques où de tels algorithmes ont plus de chance d'exister.

Nous considérons donc une sous-famille de graphes planaires, la famille particulière des graphes séries-parallèles, où le problème est "facile". Cette famille est définie récursivement par une succession de compositions en série ou en parallèle. Cette définition qui nous a permis, entre autres, de suivre la construction d'un graphe série-parallèle, nous a aussi permis d'énoncer un résultat essentiel de préservation d'optimalité dans ce processus de construction. Résultat qui a motivé et a inspiré la formulation d'équations récursives qui couvrent tous les cas possibles de construction de sous-arbres induits pleinement feuillus. Il s'avère que ces équations cachent des algorithmes de complexité temporelle polynomiale. Nous avons donc traduit ces équations en algorithmes qui prennent en paramètre d'entrée la représentation arborescente d'un graphe série-parallèle, appelée *arbre de construction*, et qui fournissent en sortie le nombre maximal de feuilles qu'un sous-arbre induit, s'il existe, peut réaliser pour un nombre de sommets donné.

Mots-clés. Graphe série-parallèle, sous-arbre induit, feuille, pleinement feuillu, arbre de construction, graphe planaire

INTRODUCTION

Les graphes existent dans plusieurs domaines pratiques : ils sont un moyen naturel pour représenter des structures qui sont vues comme un ensemble d'objets élémentaires connectés entre eux. Comme exemples nous pouvons citer les structures moléculaires, les structures chimiques élémentaires (Krissinel et Henrick, 2004), les réseaux sociaux, l'intérêt des utilisateurs dans le commerce électronique etc. (Cai *et al.*, 2017). Ces structures peuvent être étudiées au travers des graphes qui en offrent une excellente abstraction.

La *théorie des graphes* est la branche des mathématiques qui étudie les graphes : leur définitions a été formalisée, plusieurs propriétés, notions et de nombreuses familles de graphes particuliers ont vu le jour et ont été largement étudiées. Nous pouvons, entre autre, citer les familles de graphes planaires, les graphes bipartis, les graphes complets, la propriété de connexité, la notion de cycle, etc. (Diestel, 2010). Il en va de même pour les problèmes théoriques et pratiques inhérents à ces familles de graphes, avec plus ou moins de succès dans leur résolution. En effet, souvent les problèmes sont théoriquement résolubles, mais dans la pratique, les implémentations des solutions demandent des temps de calcul déraisonnables, ce qui en fait des problèmes difficiles à résoudre ou NP-difficiles (Garey et Johnson, 1990; Woeginger, 2003).

En particulier, l'étude des *sous-graphes* d'un graphe s'avère riche en information sur plusieurs aspects du graphe lui-même. Il a par exemple été prouvé qu'un graphe est planaire si et seulement si il ne contient pas de sous-graphe qui est une subdivision du graphe complet à 5 sommets ou le graphe complet biparti à $3 + 3$

sommets (Kennedy *et al.*, 1985). L'étude de sous-graphes est aussi particulièrement intéressante dans le cas des graphes volumineux, représentant des données hétérogènes et non structurées, type de données générées par de gros systèmes informatiques par exemple (Sun *et al.*, 2012). Chercher des sous-graphes fortement connectés ou des sous-graphes ayant une caractéristique particulière est l'une des techniques de forage ou d'exploration de ces données à des fins de reconnaissance de formes ou de motifs (AlSudairy *et al.*, 2011).

Dans ce mémoire nous nous intéressons, entre autres, aux sous-graphes *induits*, plus précisément, aux *sous-arbres induits*. La notion d'*induit* impose la contrainte d'inclure toutes les arêtes qui ont leurs deux extrémités dans le sous-graphe (Diestel, 2010). Les problématiques liées à l'étude des sous-graphes induits sont nombreuses. En biologie, les sous-graphes induits connexes de graphes représentant un réseau d'interactions moléculaires sont d'un intérêt particulier, car ces sous-graphes sont associés à des fonctionnalités de biomolécules, donc les extraire permet d'isoler ces biomolécules (Maxwell *et al.*, 2014). À l'instar de la problématique des sous-graphes induits communs entre deux graphes, par exemple, qui a des applications notamment en chimie (Cuissart et Hébrard, 2005), le problème de détecter les sous-arbres induits dans un graphe permet par exemple la récupération d'information ou la recherche d'information (Zaki, 2002). Dès lors, connaitre le nombre de ces sous-arbres induits, pouvoir les énumérer, étudier leurs propriétés, etc. sont des questions dignes d'intérêt. Évidemment, en plus de proposer des algorithmes pour résoudre ces problèmes, il est vital que ces algorithmes aient un besoin raisonnable en ressources. En effet, souvent ces problèmes sont NP-complets, nous pouvons citer comme exemple, le problème de trouver un sous-arbre induit ayant un nombre de sommets supérieur à i dans un certain graphe G (Erdős *et al.*, 1986). Dans (Wasa *et al.*, 2014) les auteurs proposent un algorithme paramétrique efficace pour générer les sous-arbres induits d'un graphe avec un temps

d'exécution $\mathcal{O}(d)$, par sous-arbre induit généré, et un espace de $\mathcal{O}(n+m)$, où d est le degré maximal du graphe, n et m représentant respectivement son nombre de sommets et son nombre d'arêtes.

Les sous-arbres induits qui nous intéressent dans ce mémoire sont ceux ayant un nombre de feuilles maximal pour un nombre de sommets donné. Dans (Blondin Massé *et al.*, 2018) ces sous-arbres induits sont appelés *sous-arbres induits pleinement feuillus*. Les auteurs ont considéré le nombre maximal de feuilles des polyominos-arborescents constitués d'ensembles d'unités carrées connectées par arêtes et des polycubes constitués d'ensembles de cubes connectés par faces. L'investigation des sous-arbres induits pleinement feuillus a conduit à la découverte d'une nouvelle structure arborescente de polyformes qui réalise le nombre maximal de feuilles. Le problème SAIF de décider si, dans un graphe simple, il existe un sous-arbre induit de i sommets et ayant ℓ feuilles, est NP-complet (Blondin Massé *et al.*, 2017). Par conséquent, l'intérêt à écrire des algorithmes qui donnent des solutions exactes diminue et la recherche s'oriente soit vers des algorithmes d'approximation avec des taux de succès satisfaisants, soit vers des familles de graphes où le problème devient plus facile à résoudre, où nous pouvons par exemple formuler des algorithmes avec une complexité temporelle polynomiale. Ainsi, dans (Blondin Massé *et al.*, 2017) les auteurs ont développé un algorithme de séparation et évaluation (anglais : *Branch and Bound*) basé sur une structure de données appelée *configuration de sous-arbre induit*. Il consiste en un sous-arbre induit enrichi d'informations qui permettent de générer d'autres sous-arbres induits soit par extension, soit par exclusion ou soit par retour sur trace (anglais : *backtracking*). Par la suite, ils se sont intéressés au cas particulier des arbres, où il s'avère que le problème SAIF admet une solution d'une complexité temporelle polynomiale (Blondin Massé *et al.*, 2017).

Dans la même philosophie, nous nous intéressons dans ce mémoire à la famille

particulière des graphes séries-parallèles, où nous démontrons qu'il est possible de résoudre le problème SAIF en temps polynomial. La famille des graphes *séries-parallèles* est une sous-famille des graphes planaires qui ne contient pas de subdivision du graphe complet K_4 (Takamizawa *et al.*, 1982). Cette famille de graphes est intéressante de par ses domaines d'applications, comme par exemple dans les circuits électroniques. Dans ces derniers, les graphes séries-parallèles servent à calculer les résistances de circuits globaux ainsi que des circuits locaux représentés par des sous-graphes. Ces circuits électriques simples, appelés *connexions séries-parallèles*, où la résistance totale de résistances en série s'additionnent alors que la conductance électrique des résistances s'additionne dans des connexions parallèles (Duffin, 1965). Aussi, de nombreux problèmes combinatoires se résolvent en temps polynomial ou parfois linéaire sur cette famille (K. Takamizaw et Saito, 1982). En dépit du fait que ses graphes ont une structure simple, elle reste assez riche pour que de nombreux problèmes théoriques et combinatoires restent non-résolus sur les graphes séries-parallèles et il subsiste certains problèmes qui sont NP-complets même sur cette famille (Nishizeki *et al.*, 2001).

Le reste de ce mémoire est organisé comme suit.

Dans le chapitre 1, nous rappelons les notions de base sur la théorie des graphes, nous introduisons, avec plus de détails, les sous-arbres induits pleinement feuillus et la problématique de leur existence, et la famille des graphes séries-parallèles.

Dans le chapitre 2 nous étudions les sous-arbres induits dans les graphes séries-parallèles, nous énonçons des définitions de paramètres et de fonctions spécifiques à notre résultat et nous démontrons un résultat d'optimalité garantissant la justesse de notre approche. Puis nous énonçons un ensemble d'équations récursives qui résolvent le problème SAIF sur les graphes séries-parallèles.

Dans le chapitre 3 nous discutons des arbres de construction d'un graphe série-

parallèle qui constituent le paramètre d'entrée de nos algorithmes qui implémentent les équations récursives du chapitre 2, en démontrant pour chaque algorithme que sa complexité temporelle est polynomiale, ce qui fournit, à la fin, une solution en temps polynomial dans les graphes séries-parallèles d'un problème NP-complet pour les graphes généraux.

Enfin, nous terminons avec une conclusion qui présente des perspectives et des développements potentiels des résultats énoncés dans ce mémoire.

CHAPITRE I

PRÉLIMINAIRES

Dans ce chapitre nous rappelons quelques définitions de base sur la théorie des graphes et nous introduisons la problématique qui motive ce mémoire. Dans la section 1.1 nous faisons un rappel de quelques définitions de théorie des graphes, notamment sur les sous-graphes, les sous-arbres induits et les ensembles indépendants. Par la suite, dans la section 1.2 nous introduisons la famille des graphes séries-parallèles, nous motivons l'intérêt de travailler avec quelques-unes de leurs caractéristiques. Enfin, dans la section 1.3 nous introduisons la catégorie de sous-arbres induits qui nous intéresse dans ce mémoire, leur définition avec quelques exemples et dans la sous-section 1.3.1 nous démontrons que le problème de décision de l'existence, sous certaines conditions, de ces sous-arbres induits est NP-complet dans le cas général.

1.1 Définitions et concepts de base

Nous commençons par rappeler des définitions de base de la théorie des graphes. Pour de plus amples détails se référer à (Diestel, 2010).

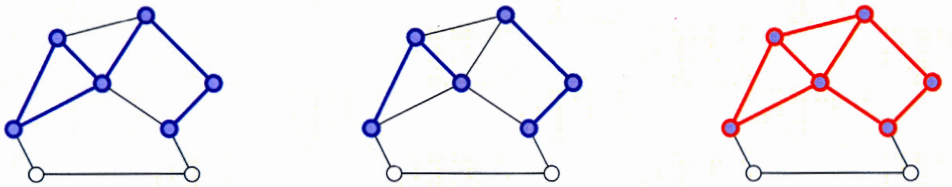
Un *graphe* est un couple $G = (V, E)$ d'ensembles tel que les éléments de E, appelés *arêtes*, sont des paires non ordonnées d'éléments de V, appelés *sommets* (ou *noeuds*). Autrement dit, une arête est un ensemble $\{u, v\}$ où $u, v \in V$ et

$u \neq v$, donc $E \subseteq \{\{u, v\} \mid u, v \in V \text{ et } u \neq v\}$. Il est possible d'avoir plusieurs copies d'une arête $\{u, v\}$, dans ce cas nous parlons de *multigraphes*. Un graphe $G = (E, V)$ est *simple*, si $E \subseteq \mathcal{P}_2(V)$, où $\mathcal{P}_2(V)$ est l'ensemble de tous les sous-ensembles distincts de V de taille 2 ($\mathcal{P}_2(V) = \{\{v_1, v_2\} \mid v_1, v_2 \in V \text{ et } v_1 \neq v_2\}$).

Deux sommets u et v sont dits *adjacents* ou *voisins*, s'il existe une arête $\{u, v\}$ dans E. Pour un sommet u nous notons $N_G(u)$ (ou $N(u)$ tout court s'il n'y a pas d'ambiguïté) l'ensemble des sommets adjacents de u dans G, c'est-à-dire, $N_G(u) = \{v \in V \mid \{u, v\} \in E\}$. Nous définissons le *degré* d'un sommet $u \in V$ par $deg(u) = |N_G(u)|$ où $|.|$ dénote la cardinalité d'un ensemble. En particulier, un sommet u est dit *pendant* si $deg(u) = 1$, si $deg(u) > 1$ le sommet u est dit *interne*.

Nous disons que $C = (u_1, u_2, \ldots, u_k)$ est une *chaîne* de $G = (V, E)$ si $u_i \in V$ pour $i = 1, 2, \ldots, k$ et $\{u_i, u_{i+1}\} \in E$ pour $i = 1, 2, \ldots, k-1$. Une chaîne est dite *élémentaire* si elle ne passe pas deux fois par un même sommet, et elle est dite *simple* si elle ne passe pas deux fois par une même arête. Un *cycle* est une chaîne simple ayant le même sommet dans les deux extrémités, c'est-à-dire, $u_1 = u_k$. Un graphe est dit *connexe* si pour tous $u, v \in V$ tels que $u \neq v$, il existe une chaîne C entre u et v. Une *composante connexe* est un sous-graphe connexe de taille maximale. En particulier, un *arbre* est un graphe qui est connexe et *acyclique*, c'est-à-dire, qu'il ne contient aucun cycle simple. Dans ce dernier cas, un sommet pendant est appelé *feuille*. Nous notons le nombre de feuilles d'un arbre T par $|T|_{\mathscr{D}}$.

Pour $U \subseteq V$, nous définissons le *sous-graphe induit* par U dans G, noté $G[U]$, par le graphe $G[U] = (U, E \cap \mathcal{P}_2(U))$. Il est important de différencier les sous-graphes induits des *sous-graphes*. En effet, un sous-graphe $H = (V_H, E_H)$ de $G = (V, E)$ est un graphe contenu dans G tel que $V_H \subseteq V$ et $E_H \subseteq \{\{u_1, u_2\} \in E \mid u_1, u_2 \in V_H\}$, noter que nous n'imposons pas que toutes les arêtes possibles de E entre les



(a) (b) (c)

Figure 1.1: Sous-graphes et sous-graphes induits. (a) et (b) sont des sous-graphes, avec (b) non connexe. Notez que, pour les mêmes sommets, nous avons le choix d'inclure ou non des arêtes dans les sous-graphes. (c) Le sous-graphe induit ayant les mêmes sommets que (a) et (b). Dans ce cas, la propriété "induit" nous oblige à inclure toutes les arêtes existantes entres les sommets du sous-graphe.

sommets de V_H soient dans E_H. La figure 1.1 illustre cette nuance importante entre les sous-graphes et les sous-graphes induits.

En particulier, un *sous-arbre induit* est un sous-graphe induit, connexe et acyclique. Donc, dans ce cas, il faut non seulement s'assurer que toutes les arêtes sont incluses dans le sous-graphe, mais aussi qu'il est connexe et acyclique. La figure 1.2 donne des exemples de sous-arbres et de sous-arbres induits.

Une *sous-forêt induite* de $k \geq 1$ composantes est un sous-graphe induit acyclique ayant exactement k composantes connexes, autrement dit, k sous-arbres induits. Dans la suite et pour alléger l'écriture, nous parlons de *forêt induite* pour indiquer une sous-forêt induite.

Enfin, nous appelons un ensemble de sommets S d'un graphe $G = (V, E)$ un *stable* ou un *ensemble indépendant*, si le sous-graphe induit $G[S] = (S, E \cap \mathcal{P}_2(S))$ ne contient aucune arête de G, c'est-à-dire, $E \cap \mathcal{P}_2(S) = \emptyset$. La taille du stable S est son nombre de sommets $|S|$. La figure 1.3 donne quelques exemples de stables.

Nous associons à ces ensembles particuliers le problème de décision suivant.

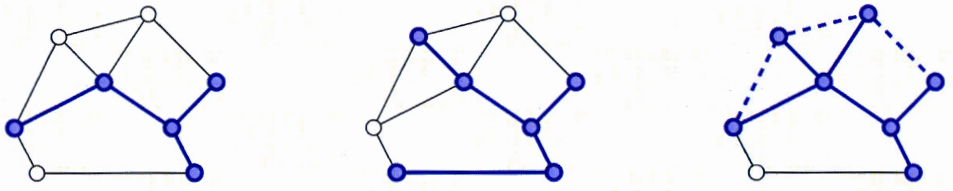


Figure 1.2: Sous-arbres et sous-arbres induits. (a) et (b) sont des sous-arbres induits. (c) Les sommets et les arêtes en bleu forment un sous-arbre, mais il n'est pas induit, car les arêtes en pointillés manquent et si nous les ajoutons nous aurons des cycles.

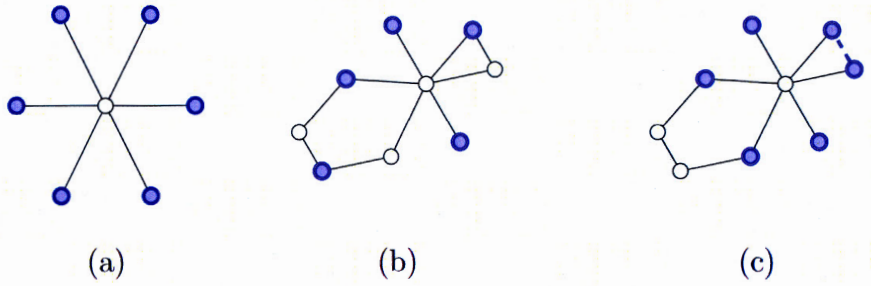


Figure 1.3: Ensembles indépendants (STABLE). Les sommets en bleu dans (a) et (b) forment des stables. Les sommets en bleu dans (c) ne forment pas un stable parce que deux d'entre eux sont reliés par une arête (représentée en pointillés).

Problème 1 (STABLE (Karp, 1972)). Étant donné un graphe G et un entier positif i, est-ce qu'un stable de taille supérieure ou égale à i existe dans G?

Ce problème est NP-complet d'après (Karp, 1972). Nous démontrons plus bas, grâce à une réduction polynomiale du problème STABLE, que le problème que nous traitons dans ce mémoire est lui aussi NP-complet en général.

1.2 Graphes séries-parallèles

Une grande variété de problèmes définis sur les graphes sont NP-complets, ainsi il est probable qu'il n'existe aucun algorithme polynomial pour les résoudre dans des graphes quelconques. Cependant, il existe certaines familles de graphes où des algorithmes polynomiaux existent pour ces problèmes. Par exemple, des algorithmes polynomiaux existent pour résoudre le problème du STABLE sur les arbres (Chen *et al.*, 1988) et sur certains graphes planaires (Alekseev *et al.*, 2008). Pour rappel, un graphe est dit *planaire* si nous pouvons le représenter sur un plan sans qu'aucune arête n'en croise une autre. Autrement dit, les arêtes se croisent uniquement en des sommets (Diestel, 2010).

Les graphes planaires ont plusieurs sous-familles intéressantes comme par exemple les arbres. L'une des sous-familles qui nous intéresse est la suivante.

Définition 1 (Graphe série-parallèle (Bodlaender et de Fluiter, 1996a)). Un *graphe série-parallèle* (ou simplement un *graphe SP*) est un triplet (G, s, t), avec $G = (V, E)$ un graphe et $s, t \in V$, tel que l'une des conditions suivantes est satisfaite :

(i) (Graphe SP de base) G a seulement deux sommets $V = \{s, t\}$ et une unique arête $E = \{\{s, t\}\}$,

(ii) (Composition en série) Il existe deux graphes SP (G_1, s_1, t_1) où $G_1 = (V_1, E_1)$

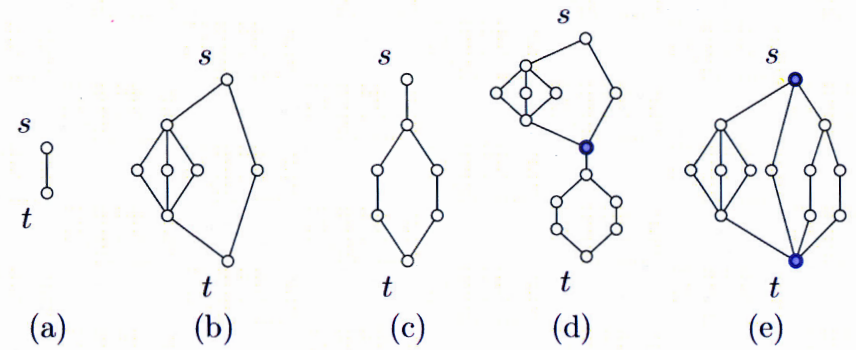


Figure 1.4: Graphes séries-parallèles. (a) Un graphe SP de base (b) Un graphe SP (c) Un autre graphe SP (d) Leur composition en série (e) Leur composition en parallèle. Dans (c) et (d), les sommets bleues sont obtenus par "fusion" du puits de (b) et de la source de (c) pour le graphe (d), et de la source de (b) avec la source de (c) et le puits de (b) avec le puits de (c) pour le graphe (e).

et (G_2, s_2, t_2) où $G_2 = (V_2, E_2)$, tels que $V = V_1 \cup V_2$, $V_1 \cap V_2 = \{t_1\} = \{s_2\}$, $E = E_1 \cup E_2$, $E_1 \cap E_2 = \emptyset$, $s = s_1$ et $t = t_2$. Dans ce cas on écrit $G = G_1 \bowtie G_2$,

(iii) (Composition en parallèle) Il existe deux graphes SP (G_1, s, t) où $G_1 = (V_1, E_1)$ et (G_2, s, t) où $G_2 = (V_2, E_2)$, tels que $V = V_1 \cup V_2$, $V_1 \cap V_2 = \{s, t\}$, $E = E_1 \cup E_2$ et $E_1 \cap E_2 = \emptyset$. Dans ce cas on écrit $G = G_1 \parallel G_2$.

Pour mieux distinguer les graphes SP, nous adoptons dans la suite la notation en quadruplet $G = (V, E, s, t)$ pour désigner ces graphes. Nous appelons les sommets particuliers s et t, respectivement la *source* et le *puits* de G. le graphe G n'étant pas orienté, on aurait pu inverser les appellations source et puits. Dans le contexte d'une composition avec un autre graphe SP, s et t sont aussi appelés *sommets de composition.* Dans la figure 1.4 nous donnons des exemples de graphes SP, notamment d'une composition en série et d'une composition en parallèle.

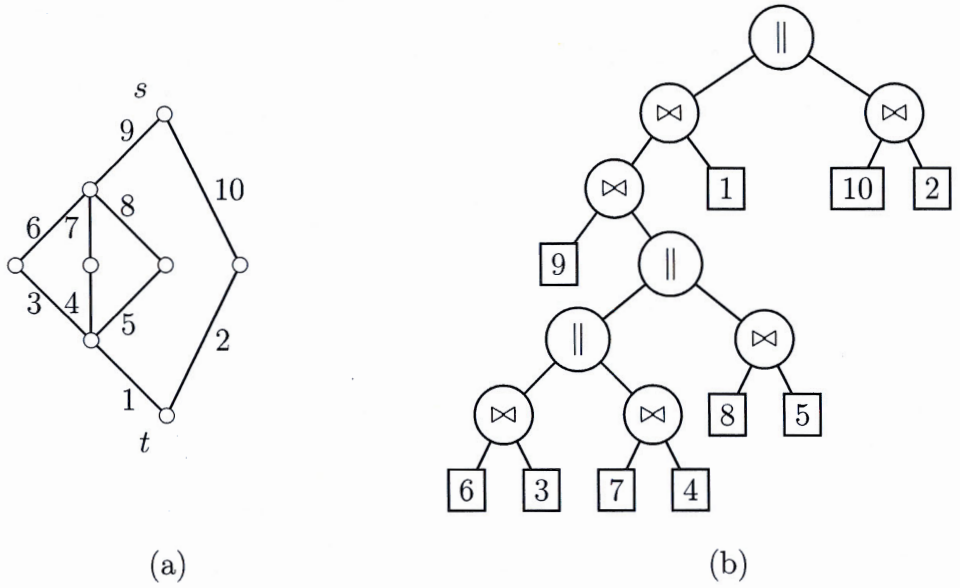
Tel que mentionné dans l'introduction, de nombreux problèmes NP-complets dans les graphes en général sont de complexité polynomiale sur les graphes SP. Donc,

en plus d'être une excellente famille de graphes à considérer pour des problèmes NP-complets, l'étude des graphes séries-parallèles en tant qu'objets mathématiques est tout aussi intéressante. Nous pouvons, par exemple, citer le problème de combinatoire des réseaux séries-parallèles, qui consiste à trouver le nombre de graphes séries-parallèles que nous pouvons former avec un certain nombre d'arêtes (Brylawski, 1971). La même problématique a été étudiée pour un nombre n de sommets (Takamizawa *et al.*, 1982). Les auteurs ont calculé une approximation asymptotique du nombre de graphes SP valant $g \cdot n^{-5/2} \rho^{-n} n!$ où g est une constante calculable et $\rho \approx 0.110213$. Ils se basent sur des variables aléatoires qui décrivent le nombre d'arêtes moyen dans un graphe SP quelconque. Dans (de Mier et Noy, 2009) les auteurs donnent des bornes pour le nombre de cycles, $c(n)$, qu'un graphe SP de taille n peut contenir. Ils étudient d'abord le nombre de chemins qu'un graphe SP peut avoir, ils ont démontré que ce nombre vérifie l'équation récursive $x_{k+1} = 1 + x_k^2$ avec $x_0 = 1$, la solution

$$\beta = \exp\left(\sum_{i \geq 1} \frac{1}{2^i} \log\left(1 + \frac{1}{x_i^2}\right)\right) = 1.502836801\ldots$$

de l'équation a permis de donner les bornes $c_1 \beta^n \leq c(n) \leq c_2 \beta^n$, où c_1 et c_2 sont des constantes positives. Aussi, l'étude des graphes séries-parallèles maximaux, où l'intérêt est porté au nombre maximal d'arêtes qu'un graphe SP peut avoir. Dans (Korenblit et Levit, 2011) les auteurs ont, par exemple, prouvé que pour un graphe SP avec n sommets, le nombre d'arêtes m est borné entre $n - 1 \leq m \leq 2n - 3$ et ils ont décrit une méthode récursive pour construire un graphe SP maximal.

Des travaux ont été menés sur des algorithmes capables de dessiner des graphes SP (Hong *et al.*, 2000; Giacomo *et al.*, 2005). En particulier, dans (Giacomo *et al.*, 2005), les auteurs donnent un algorithme linéaire qui fournit un dessin d'un graphe SP G en 3 dimensions sur k lignes parallèles, sans qu'il y ait intersection d'arêtes. La valeur minimale de k pour laquelle un dessin est possible s'appelle le *nombre*



(a) (b)

Figure 1.5: Arbre de construction d'un graphe SP. (a) Un graphe SP avec des arêtes numérotées. (b) L'arbre de sa construction. Les noeuds circulaires représentent une opération de composition, $\bowtie$ pour série et $\|$ pour parallèle, l'ordre est important, c'est-à-dire le graphe SP de la branche gauche d'un noeud circulaire est composé au graphe SP de sa branche droite dans cet ordre. Les noeuds en carrés sont des étiquettes qui représentent les arêtes du graphe SP en (a) et peuvent aussi être vus comme des graphes SP de base.

de pistes de G ($5 \leq k \leq 15$ pour les graphes SP).

Enfin, il est d'intérêt de mentionner les algorithmes de reconnaissance ou de construction des graphes SP, dont nous parlons en détail dans le chapitre 3. Ces algorithmes, à temps linéaire (Schoenmakers, 1995; Bodlaender et de Fluiter, 1996a), fournissent un arbre qui permet de retracer les compositions en série ou en parallèle au travers desquelles un graphe SP a été construit. Comme c'est une composante de la solution proposée dans ce mémoire, nous en donnons un exemple, voir figure 1.5.

1.3 Sous-arbres induits pleinement feuillus

Comme dit plus haut, nous nous intéressons dans ce mémoire aux sous-arbres induits pleinement feuillus. Nous allons donc commencer par définir ces arbres spéciaux.

Définition 2 (Fonction feuille, (Blondin Massé *et al.*, 2017)). Étant donné un graphe $G = (V, E)$ fini ou infini, soit $\mathcal{T}_G(i)$ la famille de tous les sous-arbres induits de G avec exactement i sommets. La *fonction feuille* de G, notée L_G, est la fonction qui a pour domaine $\{0, 1, 2, \ldots, |G|\}$ et est définie par

$$L_G(i) = \max\{|T|_{\text{🍃}} : T \in \mathcal{T}_G(i)\}$$

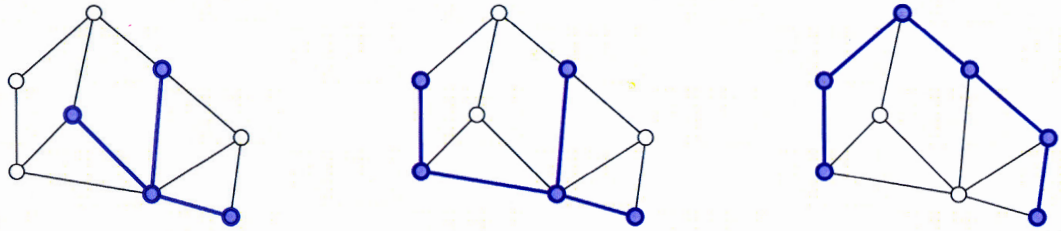
représentant le nombre maximal de feuilles d'un arbre, avec la convention $\max \emptyset = -\infty$. Un sous-arbre induit T de G avec i sommets est dit *pleinement feuillu* si $|T|_{\text{🍃}} = L_G(i)$.

De cette définition découle le problème de décision des *sous-arbres induits feuillus* (SAIF) suivant.

Problème 2. (SAIF) (Blondin Massé *et al.*, 2017)
Étant donné un graphe simple G et deux entiers positifs i et ℓ, est-ce qu'il existe un sous-arbre induit dans G avec i sommets et ℓ feuilles ?

Nous associons naturellement au problème 2 le problème d'optimisation des *sous-arbres induits pleinement feuillus*, dénoté par SAIPF, et défini comme suit.

Problème 3. (SAIPF) (Blondin Massé *et al.*, 2017)
Étant donné un graphe simple G de n sommets, quel est le nombre maximal de feuilles, $L_G(i)$, que nous pouvons réaliser par un sous-arbre induit de G de taille i, avec $i \in \{0, 1, 2, \ldots, n\}$?



(a) (b) (c)

Figure 1.6: Exemple du problème du sous-arbre induit pleinement feuillu. (a) Avec $i = 4$ sommets et $\ell = 3$ feuilles. (b) Avec $i = 5$ sommets et $\ell = 3$ feuilles. (c) Avec $i = 6$ sommets et $\ell = 2$

Avant d'étudier la complexité du problème SAIF nous allons calculer la fonction feuille pour quelques graphes, pour mieux illustrer cette notion centrale pour ce mémoire.

Soit le graphe G de la figure 1.6. Il est facile de trouver des sous-arbres induits pleinement feuillus de tailles $i = 2$ et $i = 3$ avec $\ell = 2$ pour les deux (qui est le nombre maximal de feuilles que nous pouvons former avec ces nombres de sommets). Les parties en bleu des figures 1.6 (a), (b) et (c) donnent des exemples de sous-arbres induits pleinement feuillus pour $i = 4$, $i = 5$ et $i = 6$ respectivement. Il est facile de voir que pour $i = 7$ et $i = 8$ nous ne pouvons pas avoir de sous-arbre induit, car des cycles seront forcément formés à partir de 7 sommets.

Le tableau 1.1 donne la fonction feuille pour le graphe G de la figure 1.6. Nous distinguons les cas où $\ell = 0$ pour $i = 0$ et $i = 1$ qui correspondent, respectivement, au cas de sous-arbre induit vide et au cas d'un sous-arbre réduit à un sommet, du cas où $\ell = -\infty$ qui indique que le sous-arbre induit n'existe pas pour le nombre de sommets i correspondant.

Pour mieux appréhender la fonction feuille, nous donnons ses valeurs pour quelques familles de graphes connues.

Tableau 1.1: La fonction feuille pour le graphe de la figure 1.6.

i	0	1	2	3	4	5	6	7	8
$L_G(i)$	0	0	2	2	3	3	3	$-\infty$	$-\infty$

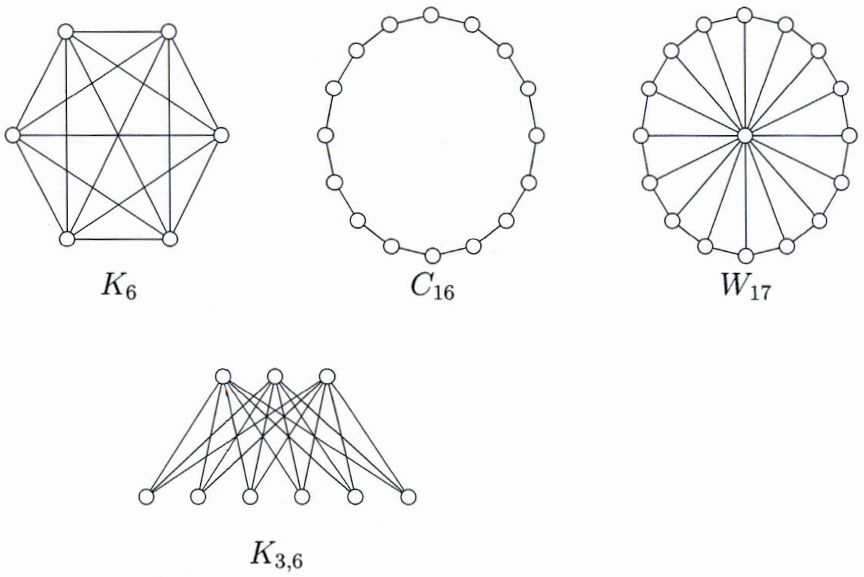


Figure 1.7: Exemples de familles de graphes connues.

Graphes complets K_n. Un graphe est *complet* si tous ses sommets sont adjacents. La figure 1.7 contient une représentation du graphe complet avec 6 sommets

Il est facile de voir que la fonction feuille vaut

$$L_{K_n}(i) = \begin{cases} 0 & \text{si } i = 0 \text{ ou } i = 1 \\ 2 & \text{si } i = 2 \\ -\infty & \text{sinon} \end{cases}$$

En effet, dès que nous dépassons 2 sommets nous avons forcément un cycle. Tous les sommets sont liés entre eux, il n'y a donc pas moyen d'avoir des sous-arbres induits de taille supérieure ou égale à 3.

Graphes cycles C_n. Un graphe *cycle* de n sommets est un graphe constitué d'un unique cycle élémentaire de longueur n. La figure 1.7 contient un exemple d'un tel graphe, pour $n = 16$.

La fonction feuille vaut,

$$L_{C_n}(i) = \begin{cases} 0 & \text{si } i = 0 \text{ ou } i = 1 \\ 2 & \text{si } 2 \leq i < n \\ -\infty & \text{sinon} \end{cases}$$

Graphes roues W_n. Un graphe *roue* W_n est un graphe cycle C_{n-1} auquel on ajoute un nouveau sommet et $n-1$ nouvelles arêtes, une entre chaque sommet de C_{n-1} et le nouveau sommet. La figure 1.7 donne un exemple d'une graphe roue avec $n = 17$.

Pour ce type de graphes, la fonction feuille vaut,

$$L_{W_n}(i) = \begin{cases} 0 & \text{si } i = 0 \text{ ou } i = 1 \\ 2 & \text{si } i = 2 \\ i-1 & \text{si } 3 \leq i \leq \left\lfloor \frac{n}{2} \right\rfloor + 1 \\ 2 & \text{si } \left\lfloor \frac{n}{2} \right\rfloor + 2 \leq i \leq n-1 \\ -\infty & \text{sinon} \end{cases}$$

Graphe biparti complet $K_{p,q}$. Un graphe est *biparti* s'il existe une partition de son ensemble de sommets en deux sous-ensembles U_1 et U_2 telle que chaque arête ait une extrémité dans U_1 et l'autre dans U_2. Un tel graphe est dit *complet* s'il a un maximum d'arêtes. La figure 1.7 donne un exemple d'un tel graphe.

Et la fonction feuille vaut,

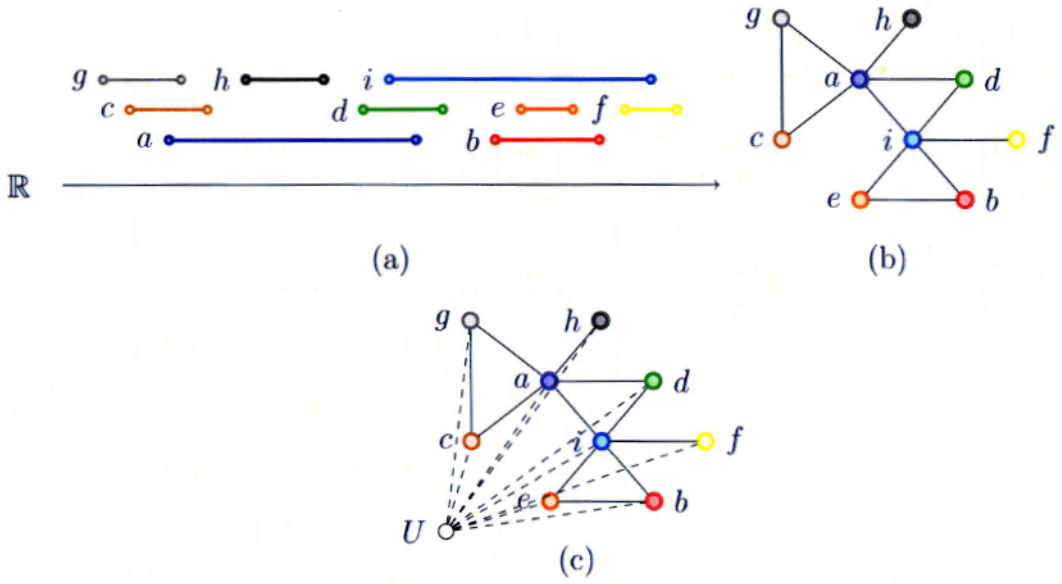


Figure 1.8: Exemple d'un graphe intervalle. (a) Un ensemble d'intervalles. (b) Sa représentation en graphe. Si deux intervalles s'intersectent, alors nous avons une arête qui relie les sommets qui représentent ces deux intervalles. (c) Le graphe obtenu en ajoutant un intervalle fictif qui s'intersecte avec tous les autres intervalles.

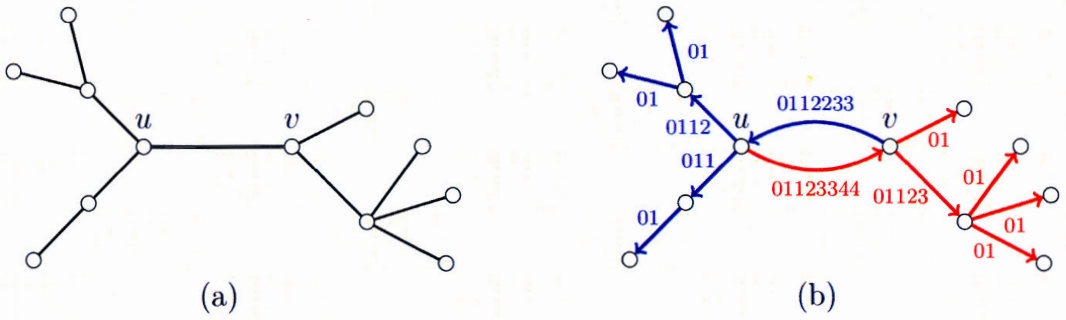
$$L_{K_{p,q}}(i) = \begin{cases} 0 & \text{si } i = 0 \text{ ou } i = 1 \\ 2 & \text{si } i = 2 \\ i-1 & \text{si } 3 \leq i \leq \max(p,q)+1 \\ -\infty & \text{sinon} \end{cases}$$

Graphes intervalles. Une autre famille digne de mention, où un sous-arbre induit pleinement feuillu a une interprétation pratique intéressante, est la famille des graphes intervalles. Un *graphe intervalle* est le graphe d'intersection d'un ensemble d'intervalles de la droite réelle. Chaque sommet du graphe d'intervalles représente un intervalle de l'ensemble, et une arête relie deux sommets lorsque l'intersection des deux intervalles n'est pas vide (McKee et McMorris, 1999). La figure 1.8 donne un exemple d'un graphe intervalle.

L'un des problème que cette famille peut modéliser est le problème d'allocation de ressources (Golumbic, 1985). Supposons que nous disposons d'une ressource qui ne traite qu'une tâche à la fois. Soit un ensemble de tâches avec un besoin en temps de la ressource, que nous représentons par un intervalle. Le problème est de savoir quelle est la meilleure allocation de ressources pouvant être réalisée sans conflits. Si nous ajoutons un intervalle fictif U qui s'intersecte avec tous les autres intervalles, nous aurons un graphe similaire au graphe de la figure 1.8 (c). Il est clair que le nombre de feuilles qu'un sous-arbre induit pleinement feuillu de taille maximale peut réaliser est le plus grand nombre de tâches qui peuvent être traitées sans conflit. En effet, les feuilles du sous-arbre induit pleinement feuillu représentent des intervalles qui ne s'intersectent pas et donc des tâches qui peuvent être traitées sans qu'il y ait de chevauchement. U peut être une feuille uniquement dans le cas d'un sous-arbre induit de taille 2, auquel cas il ne faut pas le prendre en compte.

Graphes arbres. Une autre famille de graphes intéressante est la famille des arbres. Dans (Blondin Massé *et al.*, 2017) les auteurs ont proposé un algorithme polynomial basé sur la programmation dynamique pour calculer la fonction feuille pour un arbre de taille n, avec une complexité temporelle de $\mathcal{O}(\delta n^3)$ et une complexité spatiale $\mathcal{O}(n^2)$, où δ est le degré maximal d'un sommet. Ils considèrent l'arbre non-orienté de départ, figure 1.9 (a), comme deux arborescences enracinés en u et v, comme illustré dans la figure 1.9 (b).

Ensuite, ils calculent récursivement la fonction feuille pour l'arbre enraciné en u et l'arbre enraciné en v. Chaque arête sauvegarde sa propre fonction feuille, c'est-à-dire la fonction feuille de l'arbre orienté qui a pour racine l'un des sommets de ladite arête. Dans la figure 1.9 (b) les valeurs 01 représentent le nombre de feuilles possibles pour un arbre orienté avec 1 sommet, 0 pour un sous-arbre induit de taille 0 et 1 pour un sous-arbre induit de taille 2. Attention, la notion de feuille



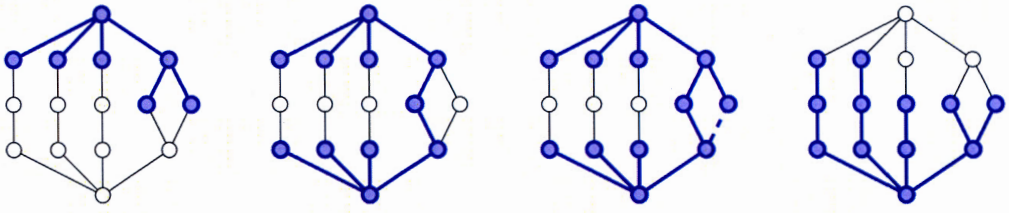
(a) (b)

Figure 1.9: Exemple du problème de sous-arbres induits pleinement feuillus sur un arbre. (a) Un arbre T (b) Étapes de caclul pour d'obtenir la fonction feuille L_T.

Tableau 1.2: La fonction feuille pour le graphe SP de la figure 1.9.

i	0	1	2	3	4	5	6	7	8	9	10	11	12
$L_G(i)$	0	0	2	2	3	4	4	5	5	6	6	7	7

est basée sur le degré sortant d'un sommet, donc la racine ne pouvant pas être une feuille. En revenant sur la récursion, ils effectuent des sommations ou des maximums pour choisir les sous-arbres induits pleinement feuillus. Pour qu'au final ils obtiennent $L_T = 0122344556677$, que nous pouvons représenter par le tableau 1.2.



(a) (b) (c) (d)

Figure 1.10: Exemple du problème du sous-arbre induit pleinement feuillu. (a) Avec $i = 7$ sommets et $\ell = 5$ feuilles. (b) Avec $i = 11$ sommets et $\ell = 6$ feuilles. (c) Un sous-arbre qui n'est pas induit. (d) Un sous-arbre induit pleinement feuillu de taille $i = 12$.

Tableau 1.3: La fonction feuille pour le graphe SP de la figure 1.10.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$L_G(i)$	0	0	2	2	3	4	4	5	5	5	5	6	5	5	$-\infty$	$-\infty$

Enfin, pour un dernier exemple, soit G le graphe SP représenté par la figure 1.10. Il est facile de prouver que les sous-arbres illustrés dans les figures 1.10 (a) et (b) sont des sous-arbres induits pleinement feuillus pour $i = 7$ et $i = 11$ respectivement. Plus précisément, nous pouvons démontrer que $(7, \ell)$ est une instance négative (elle n'a pas de solution) du problème SAIPF pour $\ell > 5$, de même $(11, \ell)$ est une instance négative du problème SAIPF pour $\ell > 6$. Par contre, la figure 1.10 (c) représente bien un sous-arbre, mais il n'est pas induit car l'arête en pointillés devrait appartenir au sous-graphe induit par les sommets en bleu, ce qui créerait un cycle. Évidemment, cet exemple n'est pas suffisant pour dire que l'instance du problème SAIPF pour $i = 12$ est négative, en effet, le sous-arbre induit représenté dans la figure 1.10 (d) est pleinement feuillus pour $i = 12$ par exemple. Dans le tableau 1.3 nous calculons le nombre maximal de feuilles pour chaque nombre de sommets possible.

À présent, nous étendons la notion de *composition* de graphes SP aux sous-arbres induits et nous la définissons ainsi,

Définition 3. Soient $A = (V_A, E_A)$ un sous-arbre induit d'un graphe SP $G_1 = (V_1, E_1, s_1, t_1)$ et $B = (V_B, E_B)$ un sous-arbre induit d'un autre graphe SP $G_2 = (V_2, E_2, s_2, t_2)$, où V_A, V_B, E_A et E_B sont les ensembles des sommets et des arêtes constituant A et B respectivement. Soit $G = (V, E, s, t)$ la composition de G_1 et G_2, alors une composition entre les sous-arbres induits A et B est possible dans les cas suivants.

$G = G_1 \bowtie G_2$ si $t_1 \in A$ et $s_2 \in B$, auquel cas nous aurons $V_A \cap V_B = \{t_1\} = \{s_2\}$

$G = G_1 \parallel G_2$ nous distinguons les deux cas suivants,

— si $s_1 \in A$ et $s_2 \in B$, auquel cas nous aurons $V_A \cap V_B = \{s_1\} = \{s_2\} = \{s\}$

— si $t_1 \in A$ et $t_2 \in B$, auquel cas nous aurons $V_A \cap V_B = \{t_1\} = \{t_2\} = \{t\}$

Dans tous ces cas nous notons $C = A \diamond B$ la composition de A et B où $C = (V_C, E_C)$ est un sous-graphe, avec $E_C = E_A \cup E_B$ et $E_A \cap E_B = \emptyset$, et $V_C = V_A \cup V_B$.

Dans les autres cas, la composition n'est pas bien définie.

Remarque 1. C n'est pas forcément un sous-arbre ni un sous-graphe induit. En effet, C peut contenir un cycle et il faut que $E_C = E \cap \mathcal{P}_2(V_C)$ pour que C soit induit. Dans la suite, ces deux propriétés sont discutées au cas par cas et des conditions pour qu'une composition entre deux sous-arbres induits donne un sous-arbre induit sont énoncées au besoin.

1.3.1 NP-complétude

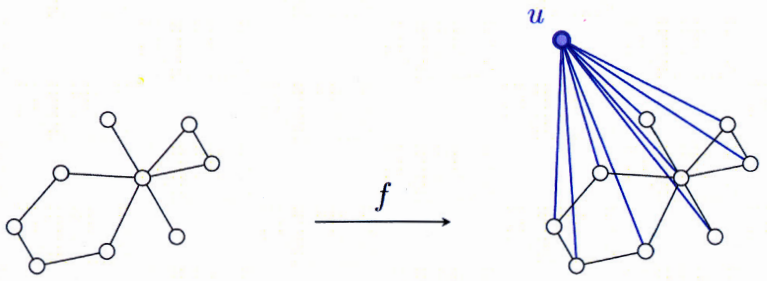
À présent, nous étudions la complexité de la résolution du problème SAIF. Pour ce faire, nous énonçons et démontrons le théorème suivant,

Théorème 1. Le problème SAIF de déterminer s'il existe ou non un sous-arbre induit ayant i sommets et ℓ feuilles dans un graphe donné est NP-complet.

Une ébauche de l'idée centrale de la preuve qui suit a été citée dans (Blondin Massé *et al.*, 2017) mais sans être rédigée. Ici, nous donnons une démonstration complète et détaillée.

Démonstration. Nous utilisons une *réduction polynomiale* du problème STABLE au problème SAIF dans cette démonstration. Pour plus de détails sur cette technique, voir (Garey et Johnson, 1990; Cormen *et al.*, 2009).

Appartenance à la classe NP. De simples parcours (temps linéaire) d'un ensemble T de sommets candidat à une instance positive de SAIF, suffisent pour



(a) (b)

Figure 1.11: Exemple de la correspondance créée par la fonction de réduction polynomiale f. (a) Le graphe $G = (V, E)$ et (b) son image par f le graphe $G' = (V', E')$.

décider si T est un sous-arbre induit et pour compter le nombre de sommets et de feuilles. Donc SAIF est bien dans la classe NP.

Nous notons une *instance* de SAIF, par le triplet (G, i, ℓ) où G est un graphe, i le nombre de sommets et ℓ le nombre de feuilles, et nous notons une instance de STABLE, par (G', i') où G' est un graphe et i' un entier positif.

Fonction de réduction. Soit la fonction de réduction f qui fait correspondre à chaque instance (G, i) de STABLE l'instance $(G', i+1, i)$ de SAIF où $G' = (V', E')$ est le graphe construit à partir de G en ajoutant un sommet universel u à V, donc $V' = V \cup \{u\}$ et les arêtes le reliant à tous les sommets de V, donc $E' = E \cup \{\{u, v\} \mid v \in V\}$. La figure 1.11 illustre la correspondance créée par la fonction f sur un exemple.

Il est clair que f est une fonction de réduction polynomiale. En effet, après la construction de V' par l'ajout du sommet u (temps constant), il suffirait d'un algorithme qui parcourt V (de complexité linéaire donc) pour construire E' et obtenir le graphe $G' = (V', E')$, au plus en temps polynomial.

STABLE vers SAIF. Si (G, i) est une instance positive de STABLE, c'est-à-dire qu'il

Figure 1.12: Correspondance d'instances : (a) un stable prouvant que l'instance $(G, 5)$ de STABLE est positive (b) le sous-arbre induit correspondant prouvant que l'instance $(G', 6, 5)$ de SAIF est positive

existe un stable de taille supérieure ou égale à i dans G, alors $f(G, i) = (G', i+1, i)$ est une instance positive de SAIF. En effet, s'il existe un stable de taille supérieure ou égale à i dans G, en particulier, il en existe un de taille i, soit S ce stable. Donc S est un sous-graphe induit de G ne contenant aucune arête de E. À présent, dans le graphe G', nous ajoutons le sommet universel u à l'ensemble S ; nous notons l'ensemble résultant $S' = S \cup \{u\}$. Il est clair que le sous-graphe induit par l'ensemble S' dans le graphe G' est un arbre, car les seules arêtes que nous devons ajouter au sous-graphe $G'[S']$ sont des arêtes de la forme $\{u, s\}$ où $s \in S$. Par conséquent, $G'[S']$ est un sous-arbre induit de G' ayant exactement $i+1 = |S|+1$ sommets dont $i = |S|$ feuilles, ce qui fait de $(G', i+1, i)$ une instance positive de SAIF. La figure 1.12 illustre un exemple de construction du sous-arbre induit dans G' à partir d'un stable de G.

SAIF vers STABLE. Inversement, si $f(G, i) = (G', i+1, i)$ est une instance positive de SAIF, c'est-à-dire qu'il existe un sous-arbre induit dans G' ayant $i+1$ sommets et i feuilles, alors (G, i) est une instance positive de STABLE. En effet, soit T un sous-arbre induit de G' ayant $i+1$ sommets et i feuilles. Sans restreindre la

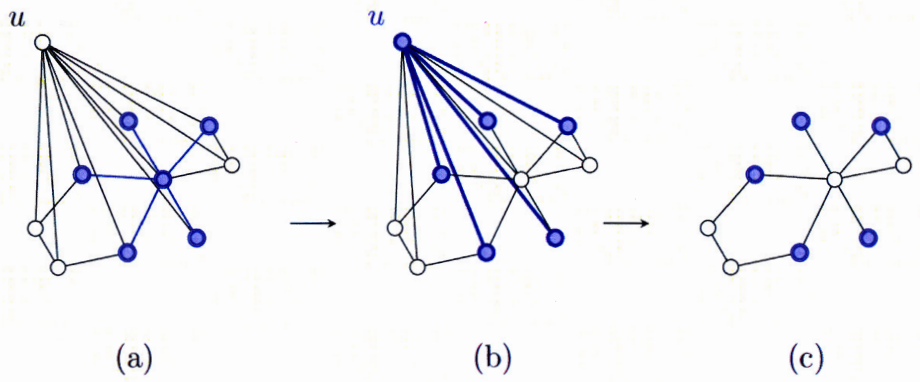


Figure 1.13: Correspondance d'instances : (a) un sous-arbre induit prouvant que l'instance $(G', 6, 5)$ de SAIF est positive, (b) le sous-arbre induit "équivalent", contenant le sommet universel u et prouvant que l'instance $(G', 6, 5)$ est positive et (c) le stable correspondant prouvant que l'instance $(G, 5)$ de STABLE est positive

généralité, nous supposons que $u \in T$, car si T ne contient pas u, nous pouvons toujours construire un autre sous-arbre induit T' de G' contenant u et ayant $i+1$ sommets dont i feuilles. En effet, il suffit de construire T' à partir de tous les sommets qui sont des feuilles dans T et nous leur ajoutons le sommet u, ainsi que toutes les arêtes reliant u à ces feuilles. T' est évidemment un sous-arbre induit (aucune arête entre les feuilles, sinon ça contredirait le fait que T est un sous-arbre induit) et les feuilles de T restent des feuilles dans T' (car il y a une seule arête qui lie chaque sommet à u). Donc T' a $i+1$ sommets dont i feuilles, ce qui en fait un candidat légitime pour l'instance positive $f(G, i) = (G', i+1, i)$. Donc nous pouvons supposer que $u \in T$.

Comme le graphe G ne contient pas u, il ne restera de T que ses i feuilles si nous le restreignons à G, étant donné qu'il n'existe aucune arête entre ces feuilles, le sous-graphe induit $G[T \cap V]$ est un stable ayant i sommets, ce qui fait de (G, i) une instance positive de STABLE. La figure 1.13 illustre un exemple de construction d'un stable dans G à partir d'un sous-arbre induit de G'.

D'où, STABLE $\leq$ SAIF et donc SAIF est NP-complet. $\square$

CHAPITRE II

SOUS-ABRES INDUITS DANS LES GRAPHES SÉRIES-PARALLÈLES

Ayant introduit toutes les définitions et les notions générales dont nous avons besoin, dans la section 2.2 nous énonçons celles plus spécifiques à nos résultats. Auparavant, à la section 2.1, nous discutons de quelques exemples pour schématiser le raisonnement suivi et l'intuition sous-jacente. Aussi, nous y démontrons un résultat d'optimalité crucial pour notre solution, qui stipule que la caractéristique de sous-arbre induit pleinement feuillu, sous certaines conditions, est propagée et parfois même préservée lors de compositions de graphes SP. Enfin, dans la section 2.3 nous énonçons notre résultat principal, un ensemble d'équations récursives qui nous permettent de calculer la fonction feuille pour différentes tailles.

Cette section a fait l'objet d'une publication à la conférence *GASCom 2018* qui a été acceptée et selectionnée pour une présentation (Abdenbi *et al.*, 2018). À noter qu'une amélioration a été apportée aux équations récursives, le nombre de cas a été réduit par rapport à la publication originale.

2.1 Propriété d'optimalité

Nous commençons par expliquer l'idée sous-tendante à notre approche et par introduire de nouvelles définitions pour formaliser cette approche. Soient les deux graphes SP G et H de la figure 2.1. Le graphe (c) est leur composition en série et

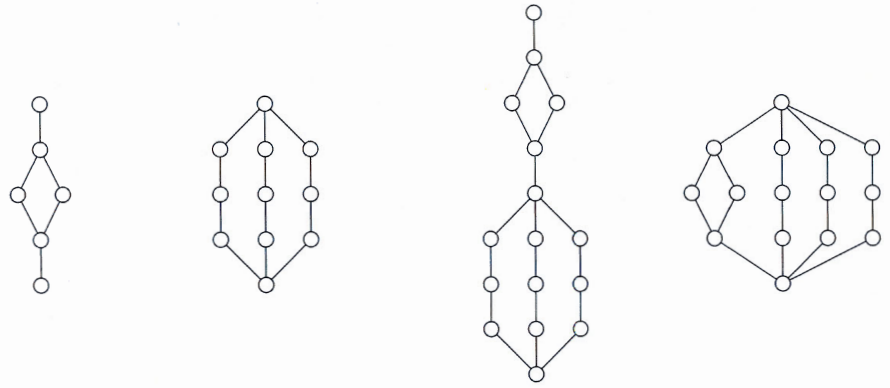
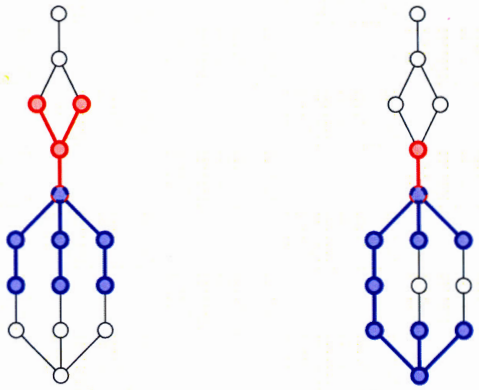


Figure 2.1: Graphes SP et leurs compositions. (a) et (b) deux graphes SP G et H respectivement. (c) La composition en série $G \bowtie H$, dans cet ordre. (d) La composition en parallèle $G \parallel H$.

le graphe (d) est leur composition en parallèle.

D'abord, en terme de nombre total de sommets, nous remarquons qu'il y a une perte dans le décompte de sommets. En effet, après une composition, due à la fusion du puits de G et de la source de H pour le graphe (c), nous perdons 1 sommet. Les fusions des sources de G et H et de leurs puits, causent la perte de 2 sommets.

À présent, dans la figure 2.2 nous voyons deux sous-arbres induits (les sommets et les arêtes coloriés), sur le même graphe, de taille $i = 10$ avec $\ell = 5$ feuilles. Ces deux sous-arbres induits sont pleinement feuillus. Une remarque importante est que ces sous-arbres sont composés, dans les deux cas de la figure 2.2, de sous-arbres induits qui ont un nombre de feuilles maximal et qui contiennent les sommets de compositions, le puits pour le sous-arbre induit en rouge et la source pour le sous-arbre induit en bleu. En effet, dans le graphe G les sous-arbres induits représentés en rouge dans la figure 2.2 ont un nombre de feuilles maximal de $\ell = 3$ et $\ell = 2$ pour des tailles $i = 4$ et $i = 2$ respectivement, de même les sous-arbres induits



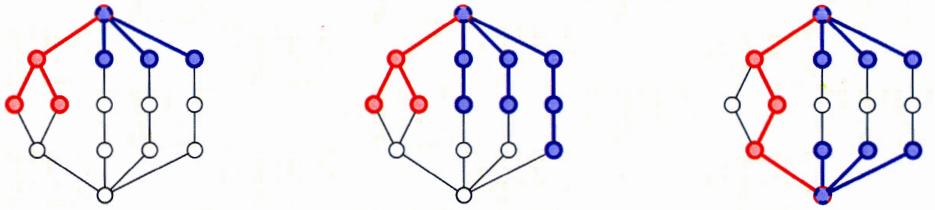
(a) (b)

Figure 2.2: Graphe SP construit par composition en série. (a) Sous-arbre induit de taille $i = 10$ et nombre de feuilles $\ell = 5$. De même pour (b).

représentés en bleu dans la figure 2.2 ont un nombre maximal de feuilles dans le graphe H, $\ell = 3$ et $\ell = 4$ pour les tailles $i = 7$ et $i = 9$ respectivement.

Nous avons ici l'une des idées principales de notre approche : *un sous-arbre induit pleinement feuillu d'un graphe SP G est composé, dans certains cas, de sous-arbres induits ayant un nombre maximal de feuilles et contenant des sommets de compositions des graphes SP qui composent G.*

Il est intéressant de remarquer que dans une composition, en plus de perdre un sommet dans le décompte total, le rôle de feuille que peuvent jouer la source ou le puits dans chacun des deux sous-arbres induits composés, va souvent se transformer en sommet interne, si la source et le puits en question sont des sommets de composition. Ce qui implique une baisse du nombre de feuilles dans le décompte total, c'est strictement inférieur à la somme du nombre de feuilles de ces deux sous-arbres induits. Par contre, un sommet qui est interne dans un sous-arbre induit le restera après une composition. En effet, par exemple, le sous-arbre induit en rouge du graphe G a $\ell = 3$ pour une taille de $i = 4$ et le sous-arbre induit en bleu du graphe H a $\ell = 3$ pour une taille de $i = 7$. Après la composition, nous avons



(a) (b) (c)

Figure 2.3: Graphe SP construit par composition en parallèle. (a) Sous-arbre induit de taille $i = 7$ et nombre de feuilles $\ell = 5$. (b) Sous-arbre induit de taille $i = 11$ et nombre de feuilles $\ell = 5$. (c) Sous-arbre induit de taille $i = 11$ et nombre de feuilles $\ell = 6$.

un sous-arbre induit avec $\ell = 5 = (3+3) - 1$ pour une taille $i = 10 = (4+7) - 1$.

Comme dit plus haut, cette approche de choisir des sous-arbres induits ayant un nombre maximal de feuilles et contenant des sommets de composition, n'est que partielle, elle est concluante dans certains cas, mais pas tous, notamment lors d'une composition parallèle. En effet, considérant le graphe (d) de la figure 2.1 construit par composition parallèle. Il est clair que le sous-arbre induit dans le graphe (a) de la figure 2.3 est pleinement feuillu et on constate qu'il est composé du sous-arbre induit avec le maximum de feuilles de taille $i = 4$ contenant la source du graphe G (en rouge) et du sous-arbre induit avec le maximum de feuilles de taille $i = 4$ contenant la source du graphe H (en bleu). Par contre, le sous-arbre induit dans le graphe (b) de la figure 2.3, construit de la même façon, n'est pas pleinement feuillu, alors que le sous-arbre induit dans le graphe (c) de la figure 2.3 l'est.

Donc pour compléter l'approche énoncée plus haut, nous devons l'élargir pour couvrir ce dernier cas de figure. Pour cela, nous allons considérer, en plus de la composition de sous-arbres induits, la possibilité de composer un sous-arbre induit et une forêt induite avec deux composantes. La forêt induite avec 2 composantes que nous considérons ici est particulière. En effet, il s'agit d'une forêt qui peut

être vide, c'est-à-dire que ses deux composantes sont vides ou peuvent être réduite à 2 sommets, un par composante. En revanche, nous excluons le cas où une des composantes est vide, qui revient au cas d'un sous-arbre induit. Et surtout, l'une des composantes doit contenir la source et l'autre composante doit contenir le puits afin de permettre la composition. Enfin, évidemment, elle doit avoir le nombre maximal de feuilles pour sa taille.

Comme nous travaillons uniquement avec des forêts induites avec 2 composantes, nous omettons dans la suite de préciser le nombre de composantes et nous utilisons *forêt induite* pour désigner une forêt induite avec 2 composantes.

En prenant en compte ces forêts induites, nous englobons toutes les possibilités de construction des sous-arbres induits pleinement feuillus s'ils existent.

À présent, nous formalisons les sous-arbres induits et les forêts induites particuliers que nous avons énoncés plus haut et que nous utilisons dans la suite.

Définition 4. Soient $G = (V, E, s, t)$ un graphe SP et un entier positif i. Soient les ensembles de sous-arbres induits et de forêt induite de taille i suivants,

1. $\mathcal{T}_G^s(i)$ l'ensemble des sous-arbres induits contenant s et ne contenant pas t. Nous distinguons les 2 sous-ensembles suivants,
   - $\text{N}\mathcal{T}_G^s(i) = \{T \in \mathcal{T}_G^s(i) \mid T \cap N_G(t) \neq \emptyset\}$
   - $\text{F}\mathcal{T}_G^s(i) = \{T \in \mathcal{T}_G^s(i) \mid T \cap N_G(t) = \emptyset\}$
2. $\mathcal{T}_G^t(i)$ l'ensemble des sous-arbres induits contenant t et ne contenant pas s. De même, nous distinguons les 2 sous-ensembles suivants,
   - $\text{N}\mathcal{T}_G^t(i) = \{T \in \mathcal{T}_G^t(i) \mid T \cap N_G(s) \neq \emptyset\}$
   - $\text{F}\mathcal{T}_G^t(i) = \{T \in \mathcal{T}_G^t(i) \mid T \cap N_G(s) = \emptyset\}$
3. $\mathcal{T}_G^{st}(i)$ l'ensemble des sous-arbres induits contenant s et contenant t
4. $\mathcal{T}_G(i)$ l'ensemble des sous-arbres induits ne contenant ni s ni t

5. $\mathcal{F}^{st}_G(i)$ l'ensemble des forêts induites dont l'une des composantes contient s et l'autre composante contient t.

Pour chacun de ces ensembles nous définissons l'ensemble maximal lui correspondant. Nous avons alors les ensembles suivants.

1. $\mathcal{FLT}^s_G(i) = \{T \in \mathcal{T}^s_G(i) \mid |T|_{\mathscr{D}} \geq |A|_{\mathscr{D}}, \forall A \in \mathcal{T}^s_G(i)\}$. Ainsi que les sous-ensembles,
   - $\text{N}\mathcal{FLT}^s_G(i) = \{T \in \text{N}\mathcal{T}^s_G(i) \mid |T|_{\mathscr{D}} \geq |A|_{\mathscr{D}}, \forall A \in \text{N}\mathcal{T}^s_G(i)\}$
   - $\text{F}\mathcal{FLT}^s_G(i) = \{T \in \text{F}\mathcal{T}^s_G(i) \mid |T|_{\mathscr{D}} \geq |A|_{\mathscr{D}}, \forall A \in \text{F}\mathcal{T}^s_G(i)\}$
2. $\mathcal{FLT}^t_G(i) = \{T \in \mathcal{T}^t_G(i) \mid |T|_{\mathscr{D}} \geq |A|_{\mathscr{D}}, \forall A \in \mathcal{T}^t_G(i)\}$. Ainsi que les sous-ensembles,
   - $\text{N}\mathcal{FLT}^t_G(i) = \{T \in \text{N}\mathcal{T}^t_G(i) \mid |T|_{\mathscr{D}} \geq |A|_{\mathscr{D}}, \forall A \in \text{N}\mathcal{T}^t_G(i)\}$
   - $\text{F}\mathcal{FLT}^t_G(i) = \{T \in \text{F}\mathcal{T}^t_G(i) \mid |T|_{\mathscr{D}} \geq |A|_{\mathscr{D}}, \forall A \in \text{F}\mathcal{T}^t_G(i)\}$
3. $\mathcal{FLT}^{st}_G(i) = \{T \in \mathcal{T}^{st}_G(i) \mid |T|_{\mathscr{D}} \geq |A|_{\mathscr{D}}, \forall A \in \mathcal{T}^{st}_G(i)\}$
4. $\mathcal{FLT}_G(i) = \{T \in \mathcal{T}_G(i) \mid |T|_{\mathscr{D}} \geq |A|_{\mathscr{D}}, \forall A \in \mathcal{T}_G(i)\}$
5. $\mathcal{FLF}^{st}_G(i) = \{F \in \mathcal{F}^{st}_G(i) \mid |F|_{\mathscr{D}} \geq |A|_{\mathscr{D}}, \forall A \in \mathcal{F}^{st}_G(i)\}$

Enfin nous notons, $\text{U}\mathcal{FLT}_G(i) = \mathcal{FLF}_G(i) \cup (\text{N}\mathcal{FLT}^s_G(i) \cup \text{F}\mathcal{FLT}^s_G(i)) \cup (\text{N}\mathcal{FLT}^t_G(i) \cup \text{F}\mathcal{FLT}^t_G(i)) \cup \mathcal{FLT}^{st}_G(i)$

La lettre $\mathcal{T}$ est pour le mot anglais *Tree* qui veut dire arbre, $\mathcal{F}$ est pour le mot anglais *Forest* qui veut dire forêt, les prefixes N, F et U veulent respectivement dire *near* : proche, *far* : loin et *union* : réunion et enfin $\mathcal{FL}$ représente *fully leafed* qui veut dire pleinement feuillu.

Remarque 2. De cette définition découlent les remarques suivantes,

- les ensembles $\mathcal{T}^s_G(i)$, $\mathcal{T}^t_G(i)$, $\mathcal{T}^{st}_G(i)$ et $\mathcal{T}_G(i)$ forment une partition de l'ensemble de tous les sous-arbres induits de taille i. Aussi, les ensembles $\text{N}\mathcal{T}^s_G(i)$ et $\text{F}\mathcal{T}^s_G(i)$ forment une partition de $\mathcal{T}^s_G(i)$, de même pour les ensembles $\text{N}\mathcal{T}^t_G(i)$, $\text{F}\mathcal{T}^t_G(i)$ et $\mathcal{T}^t_G(i)$,

— si l'un des ensembles $\mathcal{T}^s_G(i)$, $\mathcal{T}^t_G(i)$, $\mathcal{T}^{st}_G(i)$, $\mathcal{T}_G(i)$ ou $\mathcal{F}^{st}_G(i)$ est non vide alors son ensemble maximal est non vide et réciproquement. De même pour les ensembles $\text{N}\mathcal{T}^s_G(i)$, $\text{F}\mathcal{T}^s_G(i)$, $\text{N}\mathcal{T}^t_G(i)$, $\text{F}\mathcal{T}^t_G(i)$ et leurs ensembles maximaux $\text{N}\mathcal{FLT}^s_G(i)$, $\text{F}\mathcal{FLT}^s_G(i)$, $\text{N}\mathcal{FLT}^t_G(i)$, $\text{F}\mathcal{FLT}^t_G(i)$,

— si un sous-arbre induit pleinement feuillu T de taille i existe, donc $|T|_{\leaf} = L_G(i)$, alors $T \in \text{U}\mathcal{FLT}_G(i)$,

— pour toute $F \in \mathcal{FLF}^{st}_G(i)$ ses deux composantes vérifient $F_s \in \text{F}\mathcal{FLT}^s_G(i_1)$ et $F_t \in \text{F}\mathcal{FLT}^t_G(i_2)$, avec $i_1 + i_2 = i$,

— contrairement à la relation $\mathcal{T}^s_G(i) = \text{N}\mathcal{T}^s_G(i) \cup \text{F}\mathcal{T}^s_G(i)$ nous n'avons pas toujours $\mathcal{FLT}^s_G(i) = \text{N}\mathcal{FLT}^s_G(i) \cup \text{F}\mathcal{FLT}^s_G(i)$. Nous avons plutôt

$$\mathcal{FLT}^s_G(i) = \begin{cases} \text{N}\mathcal{FLT}^s_G(i) \cup \ \text{F}\mathcal{FLT}^s_G(i) & \text{si } ||\text{N}\mathcal{FLT}^s_G(i)||_{\leaf} = ||\text{F}\mathcal{FLT}^s_G(i)||_{\leaf} \\ \text{N}\mathcal{FLT}^s_G(i) & \text{si } ||\text{N}\mathcal{FLT}^s_G(i)||_{\leaf} > ||\text{F}\mathcal{FLT}^s_G(i)||_{\leaf} \\ \text{F}\mathcal{FLT}^s_G(i) & \text{si } ||\text{N}\mathcal{FLT}^s_G(i)||_{\leaf} < ||\text{F}\mathcal{FLT}^s_G(i)||_{\leaf} \end{cases}$$

avec $||X||_{\leaf}$ qui vaut le nombre de feuilles maximal que les sous-arbres de l'ensemble X peuvent réaliser si $X \neq \emptyset$ et vaut $-\infty$ si $X = \emptyset$

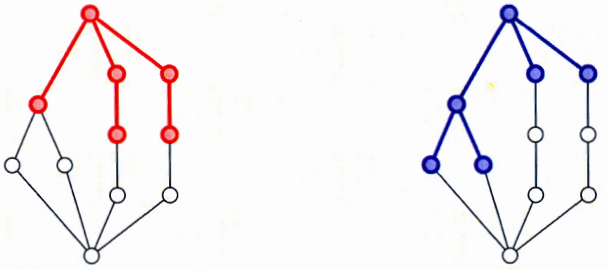
La figure 2.4 nous montre un cas où l'égalité n'est pas vérifiée.

Il en va de même pour les ensembles $\mathcal{FLT}^t_G(i)$, $\text{N}\mathcal{FLT}^t_G(i)$ et $\text{F}\mathcal{FLT}^t_G(i)$.

Avant d'aller plus loin et pour garantir la justesse de l'approche exposée plus haut (composer des sous-arbres entre eux ou un sous-arbre avec une forêt avec deux composantes), nous énonçons le lemme suivant,

Lemme 1. Soit $G = (V, E, s, t)$ un graphe SP composé, en série ou en parallèle, de deux graphes SP $G_1 = (V_1, E_1, s_1, t_1)$ et $G_2 = (V_2, E_2, s_2, t_2)$. Si $A = A_1 \cup A_2$ est un sous-arbre induit pleinement feuillu de G, où $A_1 \subseteq G_1$ et $A_2 \subseteq G_2$, alors

1. soit $A_1 \in \text{U}\mathcal{FLT}_{G_1}(i_1)$ et $A_2 \in \text{U}\mathcal{FLT}_{G_2}(i_2)$;
2. soit $A_1 \in \text{U}\mathcal{FLT}_{G_1}(i_1)$ et $A_2 \in \mathcal{FLF}^{st}_{G_2}(i_2)$ ou $A_1 \in \mathcal{FLF}^{st}_{G_1}(i_1)$ et $A_2 \in \text{U}\mathcal{FLT}_{G_2}(i_2)$



(a) (b)

Figure 2.4: Un graphe SP G pour lequel $\mathcal{FLT}^s_G(6) \neq \text{N}\mathcal{FLT}^s_G(6) \cup \text{F}\mathcal{FLT}^s_G(6)$. (a) En rouge le seul sous-arbre induit de $\text{F}\mathcal{T}^s_G(6)$, donc $\text{F}\mathcal{FLT}^s_G(6) = \text{F}\mathcal{T}^s_G(6)$ avec un nombre de feuilles de 3. (b) En bleu le seul sous-arbre induit de $\text{N}\mathcal{FLT}^s_G(6)$ avec un nombre de feuilles de 4. Il est clair que $\mathcal{FLT}^s_G(6) = \text{N}\mathcal{FLT}^s_G(6)$, nous excluons $\text{F}\mathcal{FLT}^s_G(6)$ car les sous-arbres induits de $\text{N}\mathcal{FLT}^s_G(6)$ ont plus de feuilles.

où $i_1 = |A_1|$ et $i_2 = |A_2|$

Démonstration. Nous commençons par discuter de la nature acyclique, connexe et induite de A_1 et A_2. Il est clair que A_1 et A_2 sont acycliques et induits dans G_1 et G_2 respectivement. En effet, si A est acyclique et induit dans G alors sa restriction à G_1 ou G_2 restera acyclique et induite, car $V_1, V_2 \subseteq V$ et $E_1, E_2 \subseteq E$.

Ensuite, A_1 et A_2 ont au plus 2 sommets en commun. En effet, lors d'une composition, nous n'avons que les 3 cas possibles suivants :

1. soit A_1 et A_2 n'ont aucun sommet en commun, ce qui est possible seulement si $A_1 = \emptyset$ donc $A = A_2 \subseteq G_2$, ou si $A_2 = \emptyset$ donc $A = A_1 \subseteq G_1$. La figure 2.5 donne une schématisation de ce cas ;
2. soit A_1 et A_2 ont exactement un sommet en commun, dépendamment de si nous avons
   - une composition en série, auquel cas nous aurons $t_1 \in A_1$ et $s_2 \in A_2$, car la composition passe par ces sommets, qui fusionneront pour devenir un

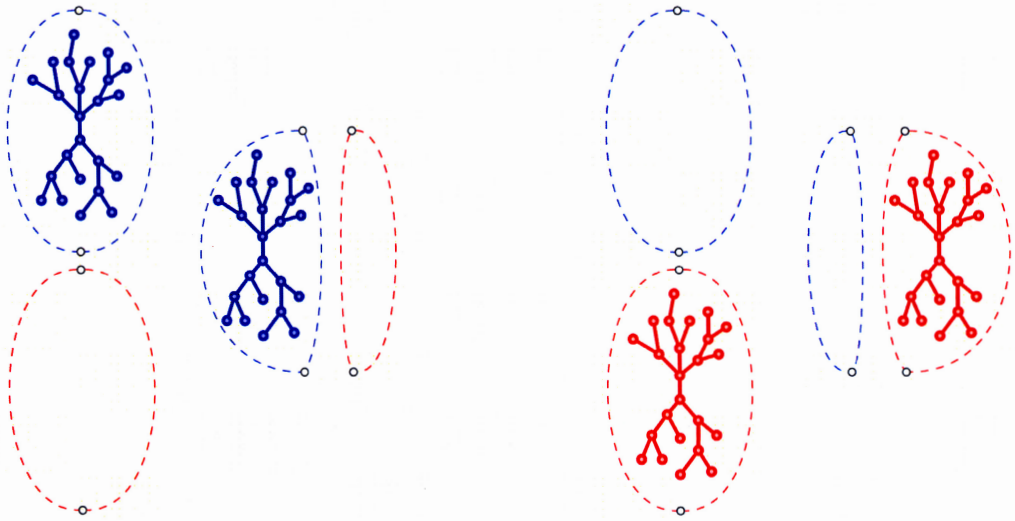
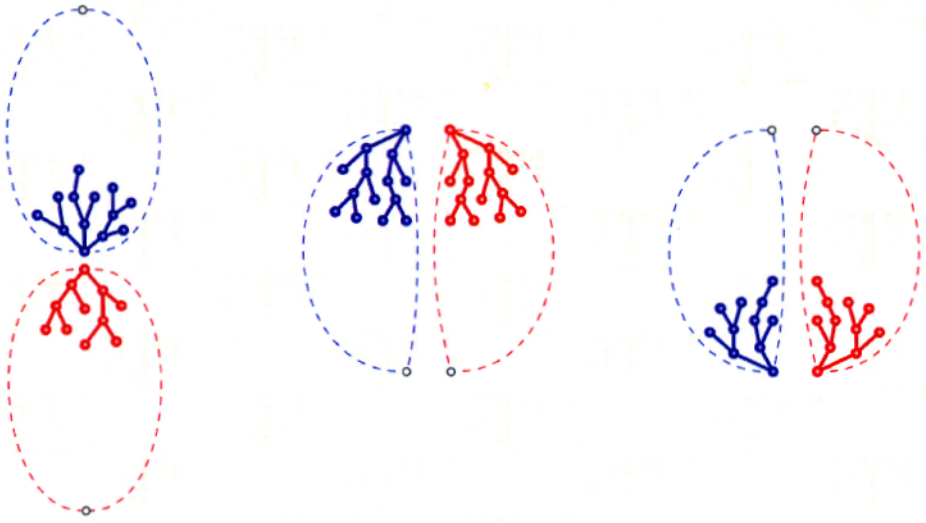


Figure 2.5: Le cas A_1 ou A_2 vide. En pointillés bleu une schématisation du graphe SP G_1 et en pointillés rouge celle du graphe SP G_2. L'arbre en bleu est la représentation de A_1 et en rouge celle de A_2. (a) Le cas $A = A_1 \subseteq G_1$ et $A_2 = \emptyset$ pour une composition en série et en parallèle. (b) Le cas analogue Le cas $A = A_2 \subseteq G_2$ et $A_1 = \emptyset$ pour une composition en série et en parallèle.



(a) (b) (c)

Figure 2.6: Les cas où A_1 et A_2 ont un sommet en commun. En bleu une représentation de G_1 et A_1 et en rouge une représentation de G_2 et A_2. (a) $A = A_1 \cup A_2$ dans le cas d'une composition en série, nous voyons que le puits de G_1 fusionne avec la source G_2 et forme le sommet commun. (b) et (c) A_1 et A_2 dans le cas d'une composition en parallèle, ils partagent un sommet en commun, la source ou le puits.

sommet interne dans G. Voir la figure 2.6 (a) pour une illustration de ce cas

— une composition en parallèle, auquel cas nous aurons soit $s_1 \in A_1$ et $s_2 \in A_2$ qui sont des sommets de composition et qui fusionnent pour donner s, ou soit $t_1 \in A_1$ et $t_2 \in A_2$ qui sont aussi des sommets de composition qui fusionnent pour donner t. Les figures 2.6 (b) et (c) illustrent ces cas

3. soit A_1 et A_2 ont exactement 2 sommets en commun à savoir $s_1 \in A_1$, $s_2 \in A_2$ qui deviendront s et $t_1 \in A_1$, $t_2 \in A_2$ qui deviendront t, cas possible uniquement dans une composition en parallèle. Voir la figure 2.7 pour illustrer ce cas.

Dans le premier cas, A reste un sous-arbre induit pleinement feuillu dans G_1 ou

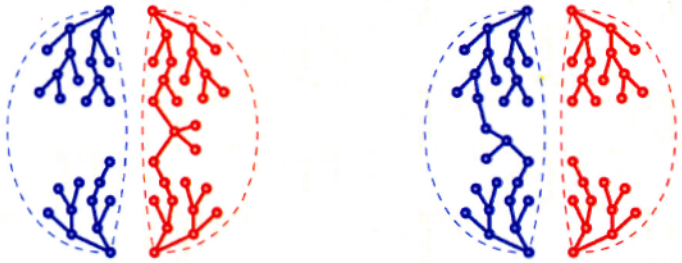


Figure 2.7: Les cas où A_1 et A_2 ont deux sommets en commun. De même, en bleu une représentation de G_1 et A_1 et en rouge une représentation de G_2 et A_2. Les deux sommets communs possibles sont donc la source et le puits. Nous n'avons donc pas le choix d'avoir soit A_1, soit A_2 qui aura 2 composantes connexes.

G_2 et d'après la remarque 2, $A = A_1 \in \text{U}\mathcal{FLT}_{G_1}(i_1)$ ou $A = A_2 \in \text{U}\mathcal{FLT}_{G_2}(i_2)$, ce qui démontre le lemme pour ce cas.

Dans le deuxième cas, nous avons un sommet en commun et comme A est connexe, le "couper" ou le scinder au niveau de ce sommet nous donnera exactement 2 composantes connexes A_1 et A_2, acycliques et induites, donc des sous-arbres induits. C'est similaire pour le troisième cas, nous avons 2 sommets communs s et t, A se divise donc en 3 composantes connexes, l'une des composantes contiendra la source et le puits d'un même graphe et sera donc dans son ensemble $\mathcal{T}^{st}()$, les deux autres seront dans l'autre graphe, l'une des deux contiendra la source et l'autre contiendra le puits, ce qui en fait un élément de $\mathcal{F}^{st}()$ de ce graphe. Ce qui se traduit par, soit $A_1 \in \mathcal{T}^{st}_{G_1}(i_1)$ et $A_2 \in \mathcal{F}^{st}_{G_2}(i_2)$, soit $A_1 \in \mathcal{F}^{st}_{G_1}(i_1)$ et $A_2 \in \mathcal{T}^{st}_{G_2}(i_2)$.

Maintenant que nous savons que A_1 et A_2 sont tous deux des sous-arbres induits, ou un sous-arbre induit et une forêt induite, il ne nous reste à démontrer que A_1 et A_2 appartiennent aux ensembles maximaux leur correspondant.

Pour ce faire nous raisonnons par l'absurde. Nous supposons donc que A est un sous-arbre induit pleinement feuillu dans G, mais que $A_1 \notin \text{U}\mathcal{FLT}_{G_1}(i_1)$ ou $A_2 \notin \text{U}\mathcal{FLT}_{G_2}(i_2)$ pour le premier cas du lemme et pour le deuxième cas $A_1 \notin$

$\text{U}\mathcal{FLT}_{G_1}(i_1)$ ou $A_2 \notin \mathcal{FLF}^{st}_{G_2}(i_2)$, ou $A_1 \notin \mathcal{FLF}^{st}_{G_1}(i_1)$ ou $A_2 \notin \text{U}\mathcal{FLT}_{G_2}(i_2)$.

Pour la suite, nous procédons suivant le nombre de sommets communs entre A_1 et A_2. Le cas où nous n'avons pas de sommet en commun étant traité plus haut. Nous traitons le cas où nous avons un seul sommet en commun en premier, puis le cas de deux sommets.

Un sommet de composition. L'hypothèse de l'absurde nous donne les cas suivants à traiter,

- $A_1 \notin \text{U}\mathcal{FLT}_{G_1}(i_1)$ et $A_2 \in \text{U}\mathcal{FLT}_{G_2}(i_2)$
- $A_1 \in \text{U}\mathcal{FLT}_{G_1}(i_1)$ et $A_2 \notin \text{U}\mathcal{FLT}_{G_2}(i_2)$
- $A_1 \notin \text{U}\mathcal{FLT}_{G_1}(i_1)$ et $A_2 \notin \text{U}\mathcal{FLT}_{G_2}(i_2)$

Au lieu de traiter ces trois cas un par un, nous traitons le cas général où l'un des sous-arbres n'appartient pas à l'ensemble maximal lui correspondant et l'autre est quelconque. En effet, cette approche nous permet de démontrer, tour à tour, que $A_1 \in \text{U}\mathcal{FLT}_{G_1}(i_1)$ puis que $A_2 \in \text{U}\mathcal{FLT}_{G_2}(i_2)$.

Sans restreindre la généralité, nous faisons les suppositions suivantes,

- $A_1 \notin \text{U}\mathcal{FLT}_{G_1}(i_1)$ (hypothèse de l'absurde) et que A_2 est quelconque
- t_1 et t_2 sont les sommets de compositions
- $s_1 \notin A_1$

Par conséquent, $A_1 \in \mathcal{T}^t_{G_1}(i_1)$. Il est important de garder le même "éloignement" qu'a A_1 de s_1. En effet, dans une composition en parallèle, si par exemple $s_2, t_2 \in A_2$, dans ce cas il est nécessaire que $A_1 \in \text{F}\mathcal{T}^t_{G_1}(i_1)$ (il n'a aucun sommet voisin de s_1), sinon nous aurons un cycle. Nous travaillons donc soit sur $\text{N}\mathcal{T}^t_{G_1}(i_1)$ ou sur $\text{F}\mathcal{T}^t_{G_1}(i_1)$ dépendemment de A_1. Encore une fois, sans restreindre la généralité, nous supposons que $A_1 \in \text{F}\mathcal{T}^t_{G_1}(i_1)$. L'hypothèse de l'absurde devient alors $A_1 \notin \text{F}\mathcal{FLT}^t_{G_1}(i_1)$.

Soit $T_1 \in \text{F}\mathcal{FLT}^t_{G_1}(i_1)$ (nous savons que $\text{F}\mathcal{FLT}^t_{G_1}(i_1) \neq \emptyset$, car $A_1 \in \text{F}\mathcal{T}^t_{G_1}(i_1)$),

nous avons donc $|T_1|_{\mathscr{D}} > |A_1|_{\mathscr{D}}$. Considérons maintenant T la composition de T_1 et A_2 au travers du sommet commun $t_1 = t_2$, nous sommes sûr que T est un sous-arbre induit. En effet, T_1 a le même éloignement de s_1 que A_1, donc aucun risque que T contienne un cycle.

$t_1 \in T_1$ et $t_2 \in A_2$ sont les deux sommets de composition pour construire T. Il est clair que si t_1 est une feuille dans T_1, dans T ça sera un sommet interne et nous perdrons donc une unité dans le décompte total de feuilles de T. Par contre, si t_1 est un sommet interne dans T_1 alors nous gardons toutes les feuilles de T_1. Il en va de même pour le "rôle" du sommet t_2 dans A_2. Étant commun à A et T, A_2 aura le même effet sur le décompte de feuilles. Nous supposons donc que t_2 est un sommet interne dans A_2.

Nous avons donc,

$$\begin{aligned} |A|_{\mathscr{D}} \leq |A_1|_{\mathscr{D}} + |A_2|_{\mathscr{D}} &< |T_1|_{\mathscr{D}} + |A_2|_{\mathscr{D}} \\ &\leq |T_1|_{\mathscr{D}} - 1 + |A_2|_{\mathscr{D}} \leq |T|_{\mathscr{D}} \end{aligned}$$

Ce qui contredit le fait que A est pleinement feuillus. Donc notre hypothèse d'absurde est fausse et nous avons bien $A_1 \in \text{F}\mathcal{FLT}^t_{G_1}(i_1) \subseteq \text{U}\mathcal{FLT}_{G_1}(i_1)$.

Avant de continuer, nous revenons brièvement sur les différentes suppositions que nous avons faites. L'approche et les calculs restent valides dans tous les cas possibles. En effet, nous avions fait les suppositions suivantes,

- $A_1 \in \text{F}\mathcal{T}^t_{G_1}(i_1)$. L'autre possibilté est de considérer $A_1 \in \text{N}\mathcal{T}^t_{G_1}(i_1)$, nous aurions pris $T_1 \in \text{N}\mathcal{FLT}^t_{G_1}(i_1)$ et nous arriverions à la même contradiction.
- $s_1 \notin A_1$. L'autre possibilité est de considérer $s_1 \in A_1$ et il aurait suffit de prendre $T_1 \in \mathcal{FLT}^{st}_{G_1}(i_1)$, ayant toujours un sommet de composition, les calculs resteront valides et meneront à la même conclusion.
- t_1 et t_2 sommets de composition. Les autres possibiltés sont s_1 et s_2 ou t_1 et s_2. La première est un cas analogue à celui que nous venons de traiter,

suffirait de considérer les ensembles contenant la source au lieu du puits. Dans la deuxième, il n'y a aucun changement dans les calculs et l'éloigement n'a aucune influence car nous serons dans une composition en série pour ce cas. Dans les deux cas, ça aboutira à la même conclusion, car il s'agira toujours de composer 2 sous-arbres induits au travers d'un seul sommet.

Enfin, en suivant la même démarche nous pouvons démontrer que $A_2 \in \cup\mathcal{FLT}_{G_2}(i_1)$. Ce qui démontre le premier cas du lemme.

Deux sommets de composition. L'hypothèse de l'absurde pour le deuxième cas du lemme nous donne

- $A_1 \notin \cup\mathcal{FLT}_{G_1}(i_1)$ et $A_2 \in \mathcal{FLF}^{st}_{G_2}(i_2)$
- $A_1 \in \cup\mathcal{FLT}_{G_1}(i_1)$ et $A_2 \notin \mathcal{FLF}^{st}_{G_2}(i_2)$
- $A_1 \notin \cup\mathcal{FLT}_{G_1}(i_1)$ et $A_2 \notin \mathcal{FLF}^{st}_{G_2}(i_2)$

ou

- $A_1 \notin \mathcal{FLF}^{st}_{G_1}(i_1)$ et $A_2 \in \cup\mathcal{FLT}_{G_2}(i_2)$
- $A_1 \in \mathcal{FLF}^{st}_{G_1}(i_1)$ et $A_2 \notin \cup\mathcal{FLT}_{G_2}(i_2)$
- $A_1 \notin \mathcal{FLF}^{st}_{G_1}(i_1)$ et $A_2 \notin \cup\mathcal{FLT}_{G_2}(i_2)$

Il est à noter, qu'au niveau des sommets de compositions, il s'agit toujours de compositions de deux sous-arbres induits (une forêt étant la réunion de deux sous-arbres). Donc, nous nous ramenons au cas traité plus haut. Par conséquent, le cas de deux sommets de compositions se réduit à un l'application deux fois de suite du cas d'un sommet de composition, démontrait plus haut. Ce qui conclut cette démonstration. □

Les forêts induites étant constituées de sous-arbres induits, nous pouvons étendre le lemme 1 à ces dernières.

Corollaire 1. Soit $G = (V, E, s, t)$ un graphe composé, en série ou en parallèle, de deux graphes SP $G_1 = (V_1, E_1, s_1, t_1)$ et $G_2 = (V_2, E_2, s_2, t_2)$. Si $F = F_1 \cup F_2 \in$

$\mathcal{FLF}^{st}_{G}(i)$, où $F_1 \subseteq G_1$ et $F_2 \subseteq G_2$, alors

1. soit $F_1 \in \text{U}\mathcal{FLT}_{G_1}(i_1)$ et $F_2 \in \text{U}\mathcal{FLT}_{G_2}(i_2)$
2. soit $F_1 \in \text{U}\mathcal{FLT}_{G_1}(i_1)$ et $F_2 \in \mathcal{FLF}^{st}_{G_2}(i_2)$, ou $F_1 \in \mathcal{FLF}^{st}_{G_1}(i_1)$ et $F_2 \in \text{U}\mathcal{FLT}_{G_2}(i_2)$
3. soit $F_1 \in \mathcal{FLF}^{st}_{G_1}(i_1)$ et $F_2 \in \mathcal{FLF}^{st}_{G_2}(i_2)$ (vrai uniquement dans une composition parallèle)

où $i_1 = |F_1|$ et $i_2 = |F_2|$.

Nous ne donnons pas la démonstration de ce corollaire, car elle découle directement du lemme 1. En effet, dans les trois cas du corollaire 1, au niveau des sommets de composition, la source et le puits, il s'agira toujours de composer deux sous-arbres induits et donc nous pouvons nous ramener à l'un des cas traités dans la démonstration du lemme 1.

À présent, grâce au lemme 1, nous pouvons calculer récursivement la fonction feuille, en suivant les compositions qui construisent un graphe SP. Pour ce faire, nous avons besoin de définir de nouveaux paramètres qui nous permettront de sauvegarder des informations, comme le critère d'éloignement cité plus haut, pour chaque composition.

2.2 Encodage de l'information

Dans la démonstration du lemme 1 nous nous sommes intéressés au critère d'éloignement pour nous assurer, lors d'une composition en parallèle, qu'il n'y ait pas de cycle qui se forme. La figure 2.8 nous donne un exemple de l'importance de ce critère d'éloignement dans une composition en parallèle. Le cycle dans le graphe SP (c) de la figure 2.8 est dû au fait que le sous-arbre de (a) contient la source et le puits, et que le sous-arbre de (b) contient la source (sommet de composition) et

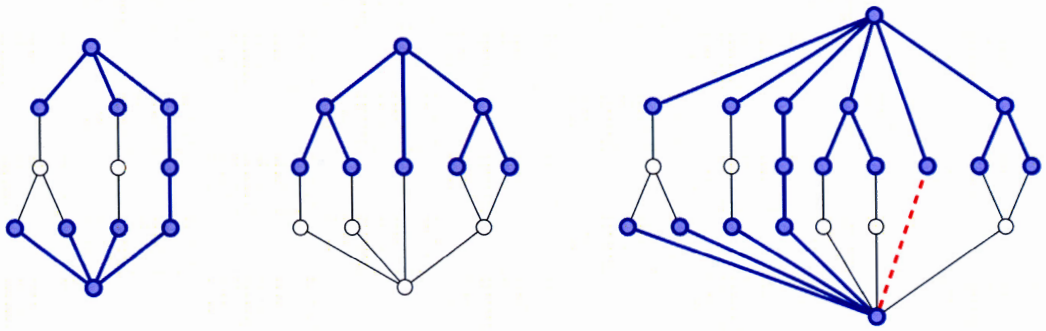
un sommet voisin du puits. Après la composition en parallèle, nous considérons la composition des sous-arbres induits pleinement feuillus de taille $i = 10$ et $i = 8$ pour avoir le sous-arbre induit pleinement feuillu de taille $i = 17$ du graphe SP composé. Mais un cycle s'est formé dans la démarche. Il est donc important de savoir s'il y a une proximité entre les sommets d'un sous-arbre induit pleinement feuillu et les deux sommets potentiels de composition la source et le puits, avant de considérer une quelconque composition de sous-arbres induits.

Comme nous l'avons mentionné plus haut, le rôle de feuille ou de sommet interne que peuvent jouer la source et le puits dans un sous-arbre induit, influence le nombre de feuilles total après une composition. En effet, si, par exemple, la source d'un graphe SP est une feuille dans un sous-arbre induit, après une composition de ce sous-arbre, le nombre de feuilles va au moins baisser de 1, car la source deviendra forcément un sommet interne.

Comme la proximité et le rôle de sommet interne ou de feuille sont tous deux liés à la source et au puits, nous pouvons encoder toute l'information qu'il nous faut dans deux valeurs, que nous définissons ainsi,

Définition 5. Soit T un sous-arbre induit d'un graphe SP $G = (V, E, s, t)$. On définit σ et τ comme suit,

$$\sigma = \begin{cases} 0 & \text{si } s \in T, \deg_T(s) = 0; \\ 1 & \text{si } s \in T, \deg_T(s) = 1; \\ 2 & \text{si } s \in T, \deg_T(s) > 1; \\ 3 & \text{si } s \notin T, |N_G(s) \cap T| \neq 0; \\ 4 & \text{si } s \notin T, |N_G(s) \cap T| = 0, \end{cases}$$



(a) (b) (c)

Figure 2.8: Formation d'un cycle après une composition parallèle. (a) et (b) deux graphes SP et en bleu des sous-arbres induits pleinement feuillus, avec $i = 10$ sommets $\ell = 5$ feuilles et $i = 8$ sommets $\ell = 5$ feuilles, respectivement. (c) leur composition en parallèle. Pour le sous-arbre induit pleinement feuillu de taille $i = 17 = 18 - 1$, nous ne pouvons pas considérer la composition des deux sous-arbres induits pleinement feuillus de (a) et (b), en cause la notion d'"induit" qui nous oblige à ajouter l'arête en pointillés rouge à la composition résultante, ce qui va créer un cycle et par conséquent, nous n'aurons plus de sous-arbre induit mais plutôt un sous-graphe induit.

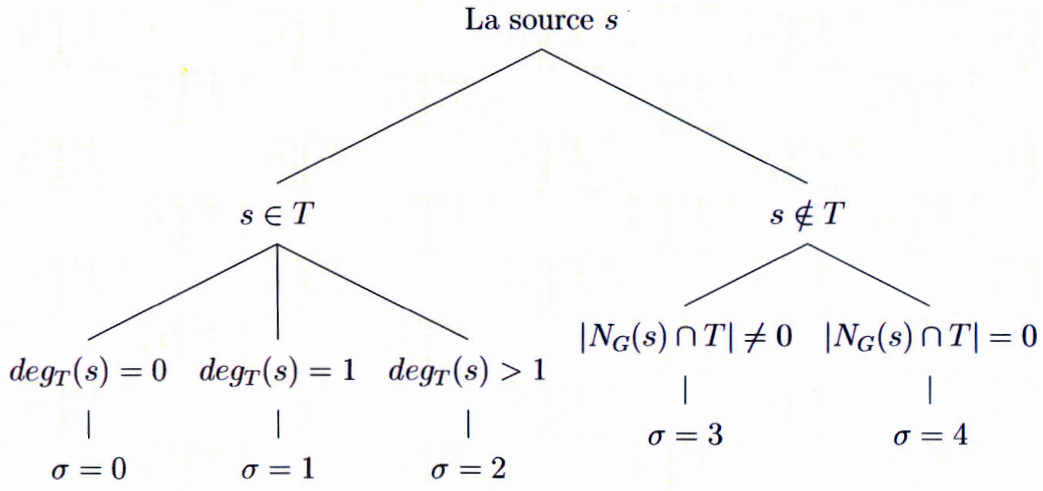


Figure 2.9: Arborescence des valeurs de σ

et

$$\tau = \begin{cases} 0 & \text{si } t \in T, \deg_T(t) = 0; \\ 1 & \text{si } t \in T, \deg_T(t) = 1; \\ 2 & \text{si } t \in T, \deg_T(t) > 1; \\ 3 & \text{si } t \notin T, |N_G(t) \cap T| \neq 0; \\ 4 & \text{si } t \notin T, |N_G(t) \cap T| = 0. \end{cases}$$

Les valeurs de σ nous renseignent donc sur le rôle de la source dans le sous-arbre induit T si elle y appartient, si non, sur l'éloignement des sommets de T de la source. De même, les valeurs de τ nous renseignent sur les mêmes critères, mais par rapport au rôle du puits dans T.

La figure 2.9 donne une représentation en arbre des valeurs de σ qui est aussi valide pour les valeurs de τ, qui peut aider à mieux voir les différents cas.

Une fois que les valeurs de σ et τ sont connues pour des sous-arbres induits, il est facile de savoir s'ils sont éloignés et de connaitre le nombre de feuilles que nous perdons après une composition. Plus formellement, la définition ci-dessous nous

permet d'exploiter ces valeurs pour avoir ces informations,

Définition 6. Soient T_1 et T_2 deux sous-arbres induits, avec σ_1 et τ_1 les valeurs associées à T_1 et σ_2 et τ_2 celles associées à T_2.

- Soit l'ensemble DistV $= \{3,4\} \times \{3,4\} \cup \{0,1,2\} \times \{4\} \cup \{4\} \times \{0,1,2\}$, nous disons que T_1 et T_2 sont éloignés au niveau de la source si $(\sigma_1, \sigma_2) \in$ DistV et éloignés au niveau du puits si $(\tau_1, \tau_2) \in$ DistV
- Pour $(a,b) \in \{0,1,2\} \times \{0,1,2\}$ nous définissons la *fonction de perte de feuilles* ρ par

$$\rho(a,b) = \begin{cases} 2 & \text{si } (a,b) = (1,1); \\ 1 & \text{si } (a,b) = (1,2) \text{ ou } (a,b) = (2,1); \\ 0 & \text{sinon.} \end{cases}$$

La fonction ρ englobe les cas suivants

- le cas où nous avons 2 feuilles, $(a,b) = (1,1)$, nous perdons effectivement 2 feuilles dans le décompte des feuilles après une composition si les sommets de composition sont lesdites feuilles,
- le cas où nous avons un sommet interne et une feuille, nous ne perdons en effet qu'une seule feuille dans le décompte de feuilles,
- il est facile de voir que dans tous les autres cas nous ne perdons aucune feuille.

À présent, nous nous intéressons à connaitre et à garder une trace du nombre maximal de feuilles que des sous-arbres induits ou des forêts induites peuvent réaliser, sous certaines contraintes. Pour ce faire, nous définissons la fonction L comme suit,

Définition 7. Soient $G = (V, E, s, t)$ un graphe SP et i un entier, avec $1 \leq i \leq |G|$. Nous notons $L(G, i, \sigma, \tau)$ le nombre maximal de feuilles qu'un sous-arbre induit T de taille i de G peut réaliser, avec σ et τ tels que définis dans la définition 2.2.

De même, soit $F(G, i, \sigma, \tau)$ le nombre maximal de feuilles qu'une forêt induite de taille i constituée de deux sous-arbres induits T_s et T_t contenant s et t respectivement puisse réaliser, avec

$$\sigma = \begin{cases} 0 & \text{si } deg_{T_s}(s) = 0; \\ 1 & \text{si } deg_{T_s}(s) = 1; \\ 2 & \text{si } \deg_{T_s}(s) > 1, \end{cases}$$

et

$$\tau = \begin{cases} 0 & \text{si } deg_{T_t}(t) = 0; \\ 1 & \text{si } deg_{T_t}(t) = 1; \\ 2 & \text{si } \deg_{T_t}(t) > 1. \end{cases}$$

À noter que, comme T_s et T_t contiennent la source et le puits, les valeurs de σ et τ vont donc être restreintes à 0 si nous avons affaire à un sommet isolé, 1 si c'est à une feuille ou à 2 si c'est un sommet interne dans la forêt. Contrairement aux valeurs σ et τ de la fonction L qui *code* l'information d'un seul sous-arbre, celles de F codent les informations de deux différents sous-arbres induits. Afin d'alléger la notation naturelle $F(G, i, (\sigma_{T_s}, 4), (4, \tau_{T_t}))$ où $(\sigma_{T_s}, 4)$ nous renseigne sur T_s et $(4, \tau_{T_t})$ sur T_t, nous omettons les valeurs $\tau_{T_s} = 4$ et $\sigma_{T_t} = 4$ dans la définition plus haut et nous remplaçons $(\sigma_{T_s}, 4)$ et $(4, \tau_{T_t})$ par $\sigma = \sigma_{T_s}$ et $\tau = \tau_{T_t}$ respectivement. Nous adoptons donc cette notation pour la suite.

Il est clair que la fonction feuille L_G de G satisfait

$$L_G(i) = \max_{0 \leq \sigma, \tau \leq 4} L(G, i, \sigma, \tau), \tag{2.1}$$

pour $i = 1, 2, 3, \ldots, |G|$. De là, il nous suffit de savoir comment $L(G, i, \sigma, \tau)$ et $F(G, i, \sigma, \tau)$ évoluent à travers les compositions successives de la construction d'un graphe SP, pour pouvoir résoudre le problème SAIF dans le cas des graphes séries-parallèles. Pour ce faire, nous allons étudier tous les cas possibles dans une composition en série et en parallèle de sous-arbres induits et aussi de forêt induite.

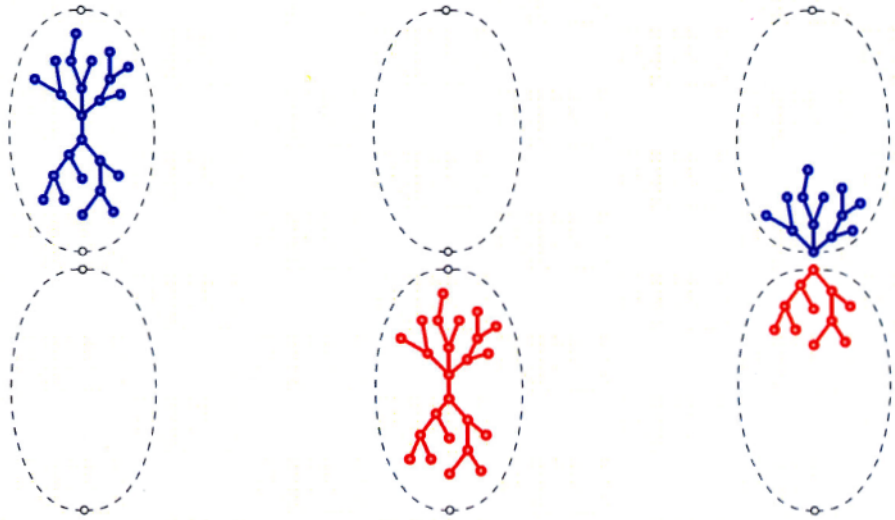


Figure 2.10: Illustrations des différents cas (SA). (a) Représente (SA1), nous voyons bien que l'éloignement de la source, σ, est conservé alors que l'éloignement du puits, τ, change et doit être réévalué. (b) Représente le cas (SA2) analogue à (SA1), le rôle est inversé entre σ et τ. (c) Le cas (SA3) où le sous-arbre fusionné garde le σ du premier sous-arbre induit le composant et le τ du deuxième.

2.3 Équations récursives

Soient $G = (V, E, s, t)$, $G_1 = (V_1, E_1, s_1, t_1)$ et $G_2 = (V_2, E_2, s_2, t_2)$ trois graphes SP, soit T un sous-arbre induit pleinement feuillu de taille i de G et F une fôret induite maximale de taille j, autrement dit $F \in \mathcal{FLF}^{st}_G(j)$.

2.3.1 Composition en série

Supposons que $G = G_1 \bowtie G_2$. On a trois cas à considérer. Une schématisation de ces cas est donnée dans la figure 2.10.

(SA1) *T est inclus dans G_1*. Nous définissons alors le nombre de feuilles de T par

(SA1)(G, i, σ, τ), tel que :

$$(SA1)(G, i, \sigma, \tau) = L(G_1, i, \sigma, \tau_1),$$

avec

$$\tau = \begin{cases} 3 & \text{si } \{s_2, t_2\} \in E_2 \text{ et } \tau_1 \in \{0, 1, 2\}\,; \\ 4 & \text{sinon.} \end{cases}$$

Comme T est entièrement inclus dans G_1, σ qui mesure sa proximité à la source s_1 qui devient s après la composition, reste inchangée dans G. Par contre, sa proximité par rapport au puits va changer. En effet, si T contient le puit de G_1, donc $\tau_1 \in \{0, 1, 2\}$, et qu'il existe une arête entre s_2 (qui fusionnera avec t_1) et t_2, T va avoir un sommet voisin du puits t_2 et donc de t, par conséquent $\tau = 3$. Dans tous les autres cas, nous sommes sûr que T sera complètement éloigné du puits t, ce qui nous donne $\tau = 4$. La figure 2.10 (a) illustre bien ce cas.

(SA2) *T est inclus dans G_2*. Nous définissons alors le nombre de feuilles de T par (SA2)(G, i, σ, τ), tel que :

$$(SA2)(G, i, \sigma, \tau) = L(G_2, i, \sigma_2, \tau)$$

avec

$$\sigma = \begin{cases} 3 & \text{si } \{s_1, t_1\} \in E_1 \text{ and } \sigma_2 \in \{0, 1, 2\}\,; \\ 4 & \text{sinon.} \end{cases}$$

De manière similaire à (SA1), la proximité vers le puits t_1 ou t restera inchangée, nous gardons donc la même valeur de τ. De même, la proximité de T de s dépend de sa proximité de s_2 dans G_2 et de l'existence d'une arête entre s_1 et t_1. Tel qu'illustré dans la figure 2.10 (b).

(SA3) *T est inclus dans G_1 et G_2*. Il existe donc un sous-arbre induit $T_1 \in \mathcal{FLT}^t_{G_1}(i_1)$ et un sous-arbre induit $T_2 \in \mathcal{FLT}^s_{G_2}(i_2)$, tel que $T = T_1 \diamond T_2$ et

$T_1 \cap T_2 = \{t_1\} = \{s_2\}$, avec $i + 1 = i_1 + i_2$. Par conséquent, le nombre de feuilles de T vaut

$$(SA3)(G, i, \sigma, \tau) = \max_{\substack{(i_1,i_2)\vdash i+1 \\ \tau_1,\sigma_2\in\{0,1,2\}}} \{L(G_1, i_1, \sigma, \tau_1) + L(G_2, i_2, \sigma_2, \tau) - \rho(\tau_1, \sigma_2)\}$$

Remarquer que (SA3) hérite de la proximité à la source du sous-arbre induit dans G_1 et de la proximité au puits du sous-arbre induit de G_2. Ainsi nous avons la même valeur σ dans la fonction L de G_1 et la même valeur de τ dans la fonction L de G_2, que les valeur dans (SA3). Évidemment, comme il y a fusion de sommets qui appartiennent aux sous-arbres induits, il y a risque de perte de feuilles, d'où la soustraction de $\rho(\tau_1, \sigma_2)$. Cas illustré par la figure 2.10 (c).

La combinaison des trois cas nous donne la valeur de la fonction feuille pour le sous-arbre induit pleinement feuillu de taille i de G, ce qui se traduit par,

$$L(G, i, \sigma, \tau) = \max_{j\in\{1,2,3\}} \{(SAj)(G, i, \sigma, \tau)\} \tag{2.2}$$

À présent, supposons que F est constituée des deux sous-arbres induits T_s et T_t, avec $s \in T_s$ et $t \in T_t$. Nous avons trois cas à considérer. La figure 2.11 schématise ces cas.

(SF1) *T_s est inclus dans G_1 et T_t est inclus dans G_2.* Autrement dit, $T_s \in \mathcal{FLT}^s_{G_1}(i_1)$ et $T_t \in \mathcal{FLT}^t_{G_2}(i_2)$. Le nombre de feuilles de F vaut alors

$$(SF1)(G, i, \sigma, \tau) = \max_{\substack{(i_1,i_2)\vdash i \\ (\tau_1,\sigma_2)\in\text{DistV}}} \{L(G_1, i_1, \sigma, \tau_1) + L(G_2, i_2, \sigma_2, \tau)\}$$

(SF2) *T_s s'étale de G_1 vers G_2 et T_t est inclus dans G_2.* Le nombre de feuilles vaut

$$(SF2)(G, i, \sigma, \tau) = \max_{\substack{(i_1,i_2)\vdash i+1 \\ \tau_1,\sigma_2\in\{0,1,2\}}} \{L(G_1, i_1, \sigma, \tau_1) + F(G_2, i_2, \sigma_2, \tau) - \rho(\tau_1, \sigma_2)\}$$

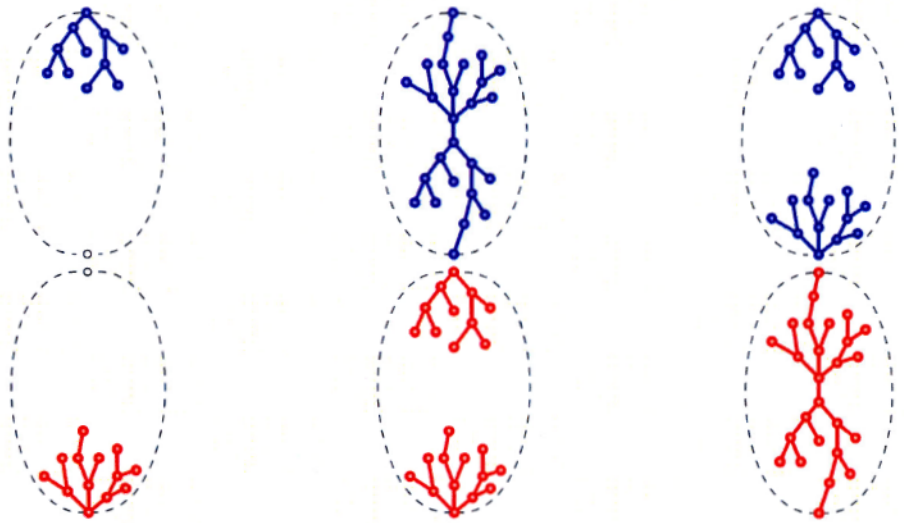


Figure 2.11: Illustrations des différents cas (SF). (a) Représente (SF1), nous voyons que la seule condition pour considérer des sous-arbres induits c'est qu'ils soient assez éloignés, ce qui se traduit par $(\tau_1, \sigma_2) \in \text{DistV}$. (b) Représente le cas (SF2) où nous disposons d'un sommet de fusion, il ne reste qu'à s'assurer s'avoir 2 composantes pour former une forêt. (c) Le cas (SF3) est analogue au cas (SF2).

Encore une fois, comme nous avons un sommet de composition, il est possible que nous perdions $\rho(\tau_1, \sigma_2)$ feuilles.

(SF3) *T_s est inclus dans G_1 et T_t s'étale de G_2 vers G_1*. Dans ce cas, le nombre de feuilles vaut

$$(SF3)(G, i, \sigma, \tau) = \max_{\substack{(i_1,i_2)\vdash i+1 \\ \tau_1,\sigma_2\in\{0,1,2\}}} \{F(G_1, i_1, \sigma, \tau_1) + L(G_2, i_2, \sigma_2, \tau) - \rho(\tau_1, \sigma_2,)\}$$

De même, nous perdons $\rho(\tau_1, \sigma_2)$ feuilles.

En combinant les trois expressions, nous obtenons

$$F(G, i, \sigma, \tau) = \max_{j=1,2,3} \{(SFj)(G_1 \bowtie G_2, i, \sigma, \tau)\} \tag{2.3}$$

Il convient de noter que, même si nous n'avons pas utilisé de forêt dans le calcul de la fonction feuilles dans l'expression 2.2, nous avons quand même besoin de maintenir le nombre de feuilles de la forêt induite de G à jour pour une éventuelle future composition en parallèle.

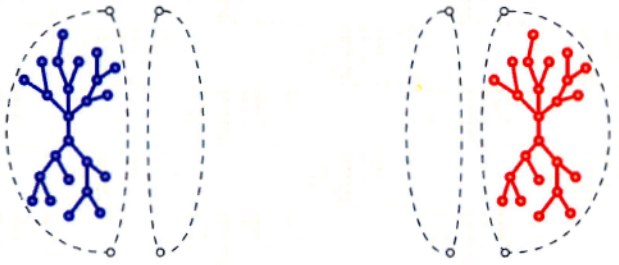
2.3.2 Composition en parallèle

La composition est plus complexe à appréhender, ceci dit, la même approche d'énumération de toutes les possibilités donne des résultats complets. Donc nous supposons que $G = G_1 \parallel G_2$.

Nous commençons par les cas liés à un sous-arbre induit pleinement feuillu. De même, nous avons T un sous-arbre induit pleinement feuillu de taille i de G et $F \in \mathcal{FLF}^{st}_G(i)$. Nous distinguons 6 cas possibles, la figure 2.12 illustre les deux premiers cas.

(PA1) *T est inclus dans G_1*. Un cas simple, le nombre de feuilles vaut

$$(PA1)(G, i, \sigma, \tau) = L(G_1, i, \sigma_1, \tau_1)$$



(a) (b)

Figure 2.12: Illustrations des cas (PA1) et (PA2). (a) Le cas (PA1). (b) Le cas (PA2)

Il faut accorder une attention particulière à la mise à jour des valeurs de σ et τ. En effet, même s'il n'y a aucune modification qui est apportée au sous-arbre induit, la composition de G_1 avec G_2 peut apporter des modifications à la proximité avec la source et le puits de G, ou même faire disparaitre certains sous-arbres induits de G_1 dans G. C'est pour cela, qu'il est nécessaire de vérifer les conditions suivantes et de mettre à jour les valeurs de σ et de τ en conséquence,

— si $\{s_2, t_2\} \in E_2$, dans ce cas il est important de s'assurer que le sous-arbre induit pleinement feuillu que nous considérons n'ait pas $\sigma_1 \in \{0,1,2\}$ *et* $\tau_1 \in \{0,1,2\}$ car dans ce cas un cycle sera créé après la composition (ce sous-arbre induit n'existera plus après la composition). Donc pour que nous puissions considérer un sous-arbre induit pleinement feuillu de G_1 comme un candidat pour rester induit et pleinement feuillu dans G également, il doit avoir $\sigma_1 \in \{3,4\}$ ou $\tau_1 \in \{3,4\}$, auquel cas nous avons

$$\sigma = \begin{cases} 3 & \text{si } \tau_1 \in \{0,1,2\} \\ \sigma_1 & \text{sinon.} \end{cases}$$

et

(a) (b)

Figure 2.13: Exemple de mise à jour pour (PA1). (a) La composition en parallèle de deux graphes SP. En bleu un sous-arbre induit pleinement feuillus, avec $\sigma_1 = 4$ et $\tau_1 = 2$. (b) Après la composition, le sous-arbre induit en bleu reste pleinement feuillus, mais à cause de l'arête en rouge et le fait que le puits appartient au sous-arbre induit ($\tau_1 \in \{0, 1, 2\}$) la proximité à la source change dans le graphe composé $\sigma = 3$ et $\tau = \tau_1 = 2$ reste inchangé.

$$\tau = \begin{cases} 3 & \text{si } \sigma_1 \in \{0, 1, 2\} \\ \tau_1 & \text{sinon.} \end{cases}$$

La figure 2.13 donne un exemple d'une telle mise à jour.

— si $\{s_2, t_2\} \notin E_2$, dans ce cas il n'y a aucun risque, nous pouvons considérer le sous-arbre induit pleinement feuillus de G_1 pour G et nous avons $\sigma = \sigma_1$ et $\tau = \tau_1$

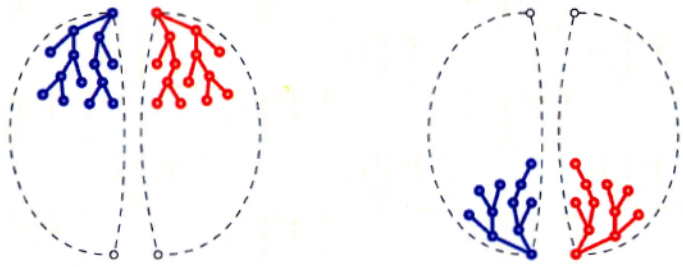
(PA2) *T est inclus dans G_2.* Tout aussi simple que (PA1) et le nombre de feuilles est

$$(PA2)(G, i, \sigma, \tau) = L(G_2, i, \sigma_2, \tau_2)$$

De même, nous avons les même conditions

— si $\{s_1, t_1\} \in E_1$ et si $\sigma_2 \in \{3, 4\}$ ou $\tau_2 \in \{3, 4\}$, alors

$$\sigma = \begin{cases} 3 & \text{si } \tau_2 \in \{0, 1, 2\} \\ \sigma_2 & \text{sinon.} \end{cases}$$



(a) (b)

Figure 2.14: Illustrations des cas (PA3) et (PA4). (a) Représente (PA3), nous voyons que la seule condition pour considérer des sous-arbres induits c'est qu'ils soient assez éloignés au niveau du puits. (b) De même pour le cas (PA4) où nous nous assurons que les deux sous-arbres induits considérés sont assez éloignés au niveau de la source.

et

$$\tau = \begin{cases} 3 & \text{si } \sigma_2 \in \{0,1,2\} \\ \tau_2 & \text{sinon.} \end{cases}$$

— si $\{s_1, t_1\} \notin E_1$, alors $\sigma = \sigma_2$ et $\tau = \tau_2$

dans les cas autres que ceux que nous avons cité plus haut, le sous-arbre induit n'existera plus après la composition.

Dans le reste des cas T *est inclus dans* G_1 *et est aussi inclus dans* G_2, donc T est composé de deux parties.

Dans les cas (PA3) et (PA4), illustrés par la figure 2.14, nous supposons que la première partie T_1 de taille i_1 est dans G_1 et la deuxième partie T_2 de taille i_2 est incluse dans G_2.

(PA3) $T_1 \cap T_2 = \{s\}$. Donc le sommet de composition est s et c'est le seul, ce qui correspond à $T_1 \in \mathcal{FLT}^s_{G_1}(i_1)$ et $T_2 \in \mathcal{FLT}^t_{G_2}(i_2)$. Dans ce cas le nombre de

feuilles est

$$(PA3)(G,i,\sigma,\tau) = \max_{\substack{(i_1,i_2)\vdash i+1\\ \sigma_1,\sigma_2\in\{0,1,2\}\\ (\tau_1,\tau_2)\in \mathrm{DistV}}} \{L(G_1,i_1,\sigma_1,\tau_1) + L(G_2,i_2,\sigma_2,\tau_2) - \rho(\sigma_1,\sigma_2)\}$$

comme un sommet isolé ($\deg = 0$) n'a aucune influence ça implique que

$$\sigma = \begin{cases} \max\{\sigma_1,\sigma_2\} & \text{si } \sigma_1 = 0 \text{ ou } \sigma_2 = 0 \\ 2 & \text{sinon.} \end{cases}$$

et $\tau = \min\{\tau_1,\tau_2\}$, c'est-à-dire que T hérite de la proximité au puits de la proximité la plus proche au puits entre celles de T_1 et T_2.

(PA4) $T_1 \cap T_2 = \{t\}$. Cas analogue à (PA3), nous avons $T_1 \in \mathcal{FLT}^t_{G_1}(i_1)$ et $T_2 \in \mathcal{FLT}^t_{G_2}(i_2)$, le nombre de feuilles vaut,

$$(PA4)(G,i,\sigma,\tau) = \max_{\substack{(i_1,i_2)\vdash i+1\\ \tau_1,\tau_2\in\{0,1,2\}\\ (\sigma_1,\sigma_2)\in \mathrm{DistV}}} \{L(G_1,i_1,\sigma_1,\tau_1) + L(G_2,i_2,\sigma_2,\tau_2) - \rho(\tau_1,\tau_2)\}$$

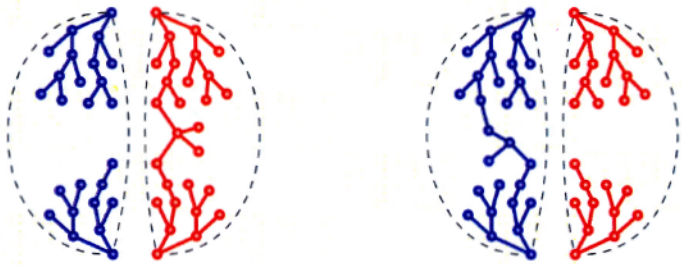
de même, avec

$$\tau = \begin{cases} \max\{\tau_1,\tau_2\} & \text{si } \tau_1 = 0 \text{ ou } \tau_2 = 0 \\ 2 & \text{sinon.} \end{cases}$$

et $\sigma = \min\{\sigma_1,\sigma_2\}$ nous prendons la valeur qui se rapproche le plus de la source.

Dans (PA5) et (PA6) nous supposons qu'une partie de T est un sous-arbre induit et l'autre partie est une forêt induite pleinement feuillus. La figure 2.15 schématise les deux cas (PA5) et (PA6).

(PA5) *T est composé d'une forêt F_1 incluse dans G_1 et d'un sous-arbre T_2 inclus dans G_2*. Ce qui se traduit par $F_1 \in \mathcal{FLF}^{st}_{G_1}(i_1)$ et $T_2 \in \mathcal{FLT}^{st}_{G_2}(i_2)$. Dans ce cas



(a) (b)

Figure 2.15: Illustrations des cas (PA5) et (PA6). (a) Représente (PA5). (b) représente le cas (PA6). À noter que comme les forêts peuvent avoir des composantes réduites à un seul sommet, nous reviendrons à des cas de composition de sous-arbres, soit au niveau de la source soit au niveau du puits.

nous avons deux sommets de composition la source et le puits, donc $s, t \in F_1$ et $s, t \in T_2$. Par conséquent, le nombre de feuilles de T est

$$(PA5)(G, i, \sigma, \tau) = \max_{\substack{(i_1,i_2)\vdash i+2 \\ \sigma_1,\tau_1,\sigma_2,\tau_2\in\{0,1,2\}}} \{F(G_1, i_1, \sigma_1, \tau_1) + L(G_2, i_2, \sigma_2, \tau_2) - \rho(\tau_1, \tau_2) - \rho(\sigma_1, \sigma_2)\}$$

avec

$$\sigma = \begin{cases} \max\{\sigma_1, \sigma_2\} & \text{si } \sigma_1 = 0 \text{ ou } \sigma_2 = 0 \\ 2 & \text{sinon.} \end{cases}$$

et

$$\tau = \begin{cases} \max\{\tau_1, \tau_2\} & \text{si } \tau_1 = 0 \text{ ou } \tau_2 = 0 \\ 2 & \text{sinon.} \end{cases}$$

À noter que nous avons une perte potentielle de feuilles au niveau de la source $\rho(\sigma_1, \sigma_2)$ et au niveau du puits $\rho(\tau_1, \tau_2)$, car les deux sont des sommets de composition.

(PA6) *T est composé d'un sous-arbre T_1 inclus dans G_1 et d'une forêt F_2 incluse dans G_2.* Un cas analogue à (PA5), nous avons $T_1 \in \mathcal{FLT}^{st}_{G_1}(i_1)$ et $F_2 \in \mathcal{FLF}^{st}_{G_2}(i_2)$, le nombre de feuilles vaut

$$(PA6)(G,i,\sigma,\tau) = \max_{\substack{(i_1,i_2)\vdash i+2 \\ \sigma_1,\tau_1,\sigma_2,\tau_2\in\{0,1,2\}}} \{L(G_1,i_1,\sigma_1,\tau_1) + F(G_2,i_2,\sigma_2,\tau_2) - \rho(\tau_1,\tau_2) - \rho(\sigma_1,\sigma_2)\}$$

avec

$$\sigma = \begin{cases} \max\{\sigma_1,\sigma_2\} & \text{si } \sigma_1 = 0 \text{ ou } \sigma_2 = 0 \\ 2 & \text{sinon.} \end{cases}$$

et

$$\tau = \begin{cases} \max\{\tau_1,\tau_2\} & \text{si } \tau_1 = 0 \text{ ou } \tau_2 = 0 \\ 2 & \text{sinon.} \end{cases}$$

En combinant tous les cas, on obtient

$$L(G,i,\sigma,\tau) = \max_{1\le j\le 6}\{(PAj)(G,i,\sigma,\tau)\} \tag{2.4}$$

Comme pour le cas d'une composition en série, nous énonçons l'unique cas de mise à jour pour la forêt induite. Supposons que $F \in \mathcal{FLF}^{st}_{G}(i)$ est une forêt induite de $G = G_1 \parallel G_2$, nous écrivons $F = F_1 \cup F_2$, où F_1 est incluse dans G_1 et F_2 est incluse dans G_2.

(PF) *F_1 et F_2 sont deux forêts non vides.* Donc, nous avons au moins deux sommets dans chaque forêt, un sommet par composante. Ça nous donne

$$(PF)(G,i,\sigma,\tau) = \max_{\substack{(i_1,i_2)\vdash i+2 \\ \sigma_1,\sigma_2,\tau_1,\tau_2\in\{0,1,2\}}} \{F(G_1,i_1,\sigma_1,\tau_1) + F(G_2,i_2,\sigma_2,\tau_2) - \rho(\sigma_1,\sigma_2) - \rho(\tau_1,\tau_2)\}$$

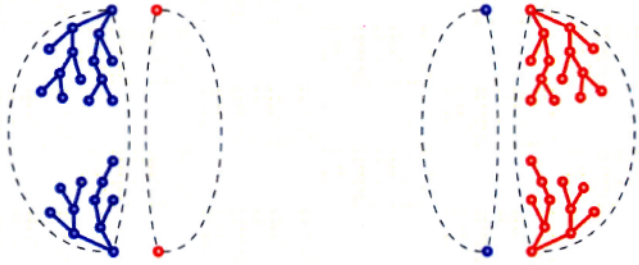


Figure 2.16: F est entièrement incluse dans G_1 ou G_2.

avec

$$\sigma = \begin{cases} \max\{\sigma_1, \sigma_2\} & \text{si } \sigma_1 = 0 \text{ ou } \sigma_2 = 0 \\ 2 & \text{sinon.} \end{cases}$$

et

$$\tau = \begin{cases} \max\{\tau_1, \tau_2\} & \text{si } \tau_1 = 0 \text{ ou } \tau_2 = 0 \\ 2 & \text{sinon.} \end{cases}$$

Cette expression englobe tous les cas suivants,

1. *F est entièrement incluse dans G_1 ou G_2.* Ce cas équivaut à F_1 ou F_2 réduite à 2 sommets. Comme les deux forêts doivent contenir la source et le puits, que l'une des deux forêts soit réduite à ces deux sommets est équivalent à dire que F est entièrement incluse dans un des deux graphes SP. La figure 2.16 donne une illustration de ces deux cas.

2. *F_2 a une composante réduite à un sommet.* Le cas où F est construite en fusionnant une forêt est un sous-arbre induit. Ce qui se traduit par F_2 a une composante réduite soit à la source soit au puit. La figure 2.17 illustre ce cas.

3. *F_1 a une composante réduite à un sommet.* Cas analogue au précédent. Voir figure 2.18 pour une illustration.

4. *F_1 et F_2 ont chacune une composante réduite à un sommet.* Ça représente le cas où la forêt F est principalement construite par deux sous-arbres induits.

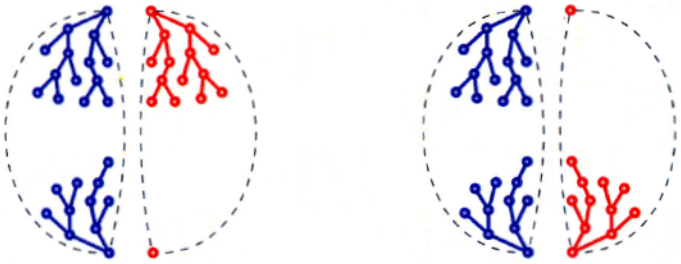


Figure 2.17: F_2 a une composante réduite à un sommet.

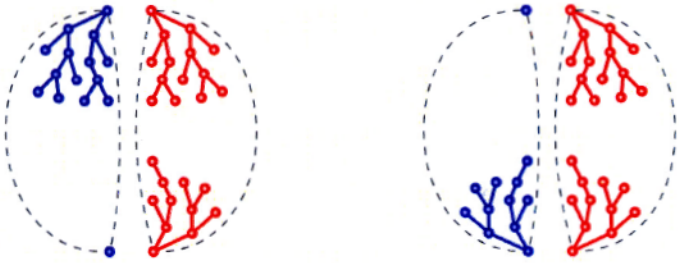


Figure 2.18: F_1 a une composante réduite à un sommet.

Figure 2.19

5. *F_1 et F_2 n'ont aucune composante réduite à un sommet.* Le cas où F est la fusion de deux forêts. Figure 2.20

6. *F a ses deux composantes réduites à un sommet.* Cas trivial où F_1 et F_2 sont réduites chacune à deux sommets.

2.3.3 Cas de base

Soit B_T le graphe SP de base illustré dans la figure 2.21 (a). Il est facile de voir que la fonction L vaut,

$$L(B_T, i, \sigma, \tau) = \begin{cases} 2 & \text{si } i = 2 \text{ et } \sigma = \tau = 1 \\ -\infty & \text{sinon.} \end{cases} \tag{2.5}$$

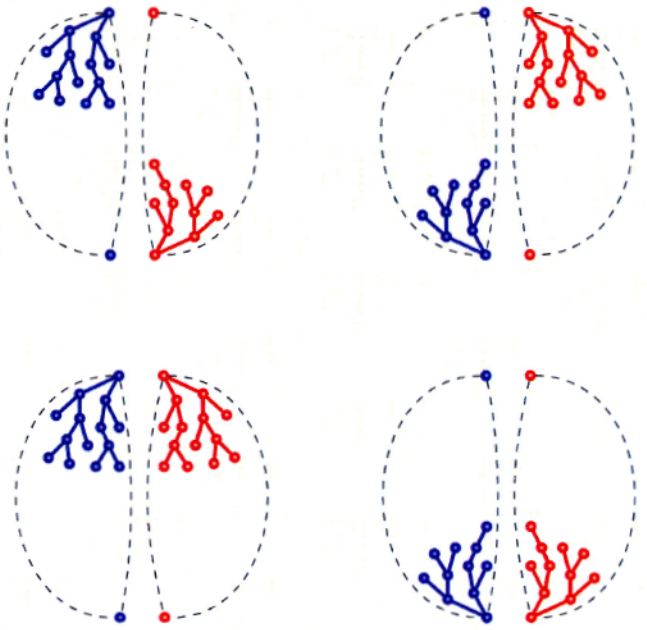


Figure 2.19: F_1 et F_2 ont chacune une composante réduite à un sommet.

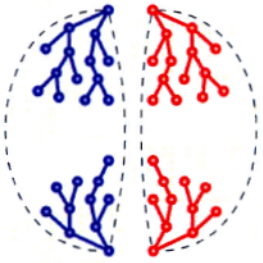


Figure 2.20: F_1 et F_2 n'ont aucune composante réduite à un sommet.

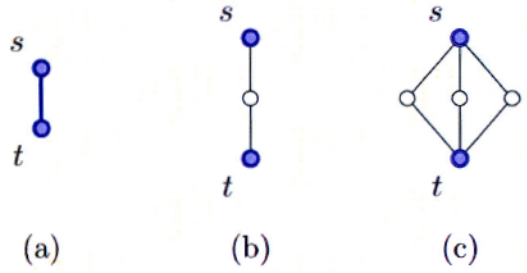


Figure 2.21: Cas de bases. (a) Graphe SP de base. (b) Le plus petit graphe SP contenant une forêt induite de base. (c) Forêt minimale dans un graphe avec uniquement des chaines de longueur 2 entre s et t.

D'une manière générale, pour n'importe quel graphe SP G, nous avons toujours

$$L(G, 2, 1, \tau) = L(G, 2, \sigma, 1) = 2 \tag{2.6}$$

pour tout $\sigma, \tau \in \{1, 2, 3\}$.

Enfin, soit B_F le graphe SP de 3 sommets illustré dans la figure 2.21 (b) et F la forêt induite en bleu. Alors,

$$F(B_F, i, \sigma, \tau) = \begin{cases} 0 & \text{si } i = 2 \text{ et } \sigma = \tau = 0 \\ -\infty & \text{sinon.} \end{cases} \tag{2.7}$$

Ainsi, pour tout graphe SP $G \neq B_F$ si nous sommes dans l'un des cas suivants,

— G est de taille inférieure à 3

— G contient une arête $\{s, t\}$

pour tout $i \in \{0, 1, \ldots, |G|\}$ et $\sigma, \tau \in \{0, 1, 2\}$, nous avons

$$F(G, i, \sigma, \tau) = -\infty \tag{2.8}$$

Il est à noter aussi que les graphes SP de forme similaire au graphe (c) de la figure 2.21 n'ont qu'une forêt minimale à deux sommets. Ce sont des graphes qui n'ont que des chaines de longueur 2 entre la source et le puits.

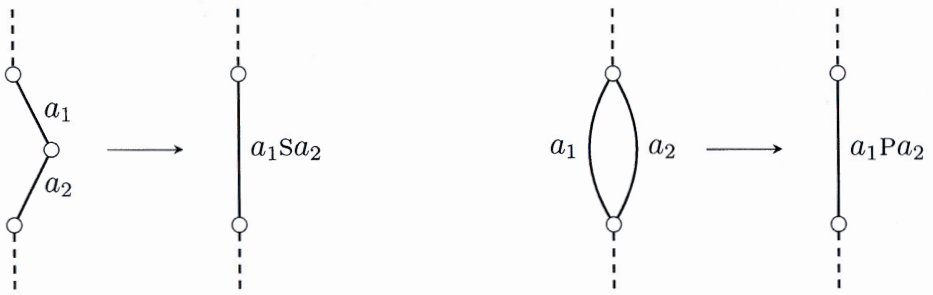
CHAPITRE III

ASPECTS ALGORITHMIQUES

Dans ce chapitre nous présentons l'implémentation de notre résultat. Il s'agit de plusieurs algorithmes avec une certaine dépendance entre eux et nous démontrons qu'ils ont une complexité temporelle polynomiale. Nous commençons en introduisant dans la section 3.1 les arbres de construction des graphes SP qui constituent l'entrée des algorithmes. Par la suite, dans la section 3.2 nous énonçons les structures de données, les méthodes et les algorithmes qui implémentent les équations récursives. Enfin, dans la section 3.3 nous discutons des algorithmes qui énumèrent les sous-arbres induits pleinement feuillus d'un graphe SP.

3.1 Arbre de construction d'un graphe SP

Il existe plusieurs algorithmes de construction d'arbre d'un graphe SP que nous trouvons par exemple dans (Valdes *et al.*, 1982; Eppstein, 1992; Schoenmakers, 1995; Bodlaender et de Fluiter, 1996b). Nous n'en citerons pas dans ce mémoire, en revanche, nous expliquons le point commun de leur structure qui permet de construire ces arbres qui sauvegardent l'ordre des différentes compositions inhérentes aux graphe SP. La principale cause est que dans ces travaux les arbres de construction ne sont qu'un résultat sous-jacent à la problématique de reconnaissance des graphes SP, c'est-à-dire, qu'étant donné un graphe quelconque décider



(a) (b)

Figure 3.1: Les deux réductions de base d'un graphe SP. (a) Réduction en série. (b) Réduction en parallèle.

s'il est un graphe série-parallèle ou non. Nous omettons donc la partie de reconnaissance et nous nous concentrons sur l'arbre de construction d'un graphe SP. Par conséquent, dans la suite nous supposons que nous avons un graphe SP et nous nous intéressons à la façon d'obtenir son arbre de construction.

Dans la plupart des algorithmes, la reconnaissance des graphes SP passe par la notion de *réduction d'un graphe*, de manière symétrique à la définition 1. Plus précisément, il s'agit d'appliquer autant de fois que possible et dans n'importe quel ordre les deux règles suivantes.

Réduction série. Tout sommet de degré 2, autre que la source et le puits, ayant donc deux arêtes adjacentes a_1 et a_2 peut être remplacé par une nouvelle arête $a_1 S a_2$ entre ses deux sommets voisins.

Réduction parallèle. Deux arêtes a_1 et a_2 ayant les mêmes extremités, seront remplacées par une seule nouvelle arête $a_1 P a_2$.

La figure 3.1 illustre les deux mécanismes cités plus haut.

Le principe est donc d'appliquer successivement ces deux opérations de réduction jusqu'à ce que le graphe SP soit réduit à un graphe SP de base. Si nous reprenons

l'exemple du graphe de la figure 1.5, nous pouvons suivre les étapes successives de consruction de l'arbre dans la figure 3.2 pour la réduction du graphe SP et la figure 3.3 pour les représentations en arbre de ces réductions. Chacun des 6 graphes de la figure 3.2 correspond à un ensemble de réductions du même type. Comme nous considérons des graphes SP simples, il n'y aura pas de réduction parallèle sur le graphe initial, donc nous cherchons en premier toutes les réductions en série possibles, puis à chaque réduction nous vérifions si nous avons de possibles réductions parallèles.

La figure 3.3 nous montre les arbres créés pour chaque réduction effectuée sur le graphe.

Nous constatons que le graphe SP considéré pour cet exemple admet au moins 2 arbres de construction différents, l'arbre (b) de la figure 1.5 et l'arbre que nous venons juste de construire. En effet, à part les points communs suivants,

- le nombre de feuilles, qui correspond au nombre d'arêtes du graphe SP initial,
- le nombre de noeuds internes, qui correspond au nombre d'opérations et
- le nombre de compositions en série et le nombre de composition en parallèle,

la forme de l'arbre peut varier dépendamment de l'ordre des compositions. C'est dû au fait que les compositions en série et en parallèle sont toutes deux des opérations associatives.

Nous avons choisi délibérément des *arbres binaires entiers*, c'est-à-dire des arbres dont chaque noeud a exactement 2 enfants, sauf s'il est une feuille. En effet, nous aurions très bien pu considérer l'arbre de construction non binaire équivalent de la figure 3.4. Pour rester cohérent, nous allons donc considérer pour la suite uniquement les arbres de construction binaires, qui font office de paramètre d'entrée pour les algorithmes de la section suivante.

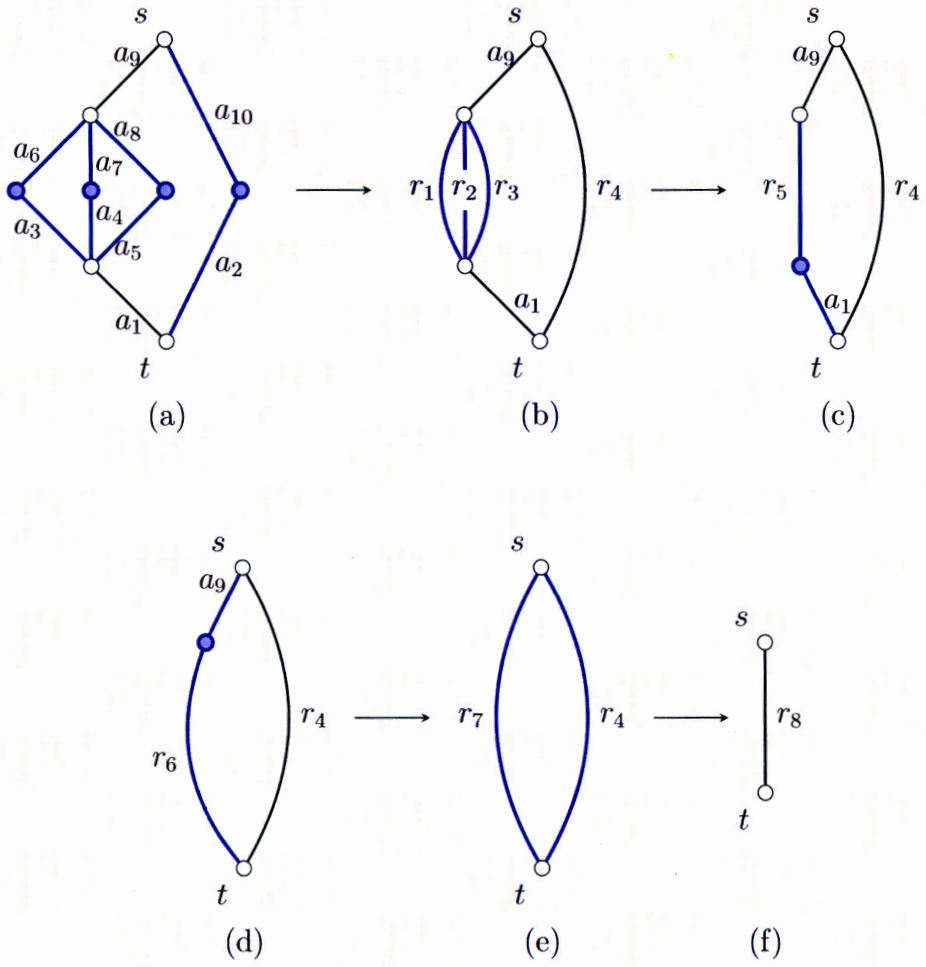


Figure 3.2: Exemple de réduction d'un graphe SP. (a) Les sommets et les arêtes adjacentes (en bleu) candidats pour une réduction en série. (b) Le graphe résultant après la réduction en série, avec $r_1 = a_6 \mathrm{S} a_3$, $r_2 = a_7 \mathrm{S} a_4$, $r_3 = a_8 \mathrm{S} a_5$ et $r_4 = a_{10} \mathrm{S} a_2$. En bleu les arêtes candidates pour une réduction en parallèle. (c) Le résultat de la réduction en parallèle et le sommet et ses arêtes candidats pour une réduction en série, avec $r_5 = r_1 \mathrm{P}(r_2 \mathrm{P} r_3) = (a_6 \mathrm{S} a_3)\mathrm{P}((a_7 \mathrm{S} a_4)\mathrm{P}(a_8 \mathrm{S} a_5))$. (d) De même, on a le résultat et les candidats pour la prochaine réduction, avec $r_6 = r_5 \mathrm{S} a_1 = ((a_6 \mathrm{S} a_3)\mathrm{P}((a_7 \mathrm{S} a_4)\mathrm{P}(a_8 \mathrm{S} a_5)))\mathrm{S} a_1$. (e) La dernière réduction en parallèle, avec $r_7 = a_9 \mathrm{S} r_6 = a_9 \mathrm{S}(((a_6 \mathrm{S} a_3)\mathrm{P}((a_7 \mathrm{S} a_4)\mathrm{P}(a_8 \mathrm{S} a_5)))\mathrm{S} a_1)$. (f) Enfin, on arrive au graphe SP de base, avec $r_8 = r_7 \mathrm{P} r_4 = (a_9 \mathrm{S}(((a_6 \mathrm{S} a_3)\mathrm{P}((a_7 \mathrm{S} a_4)\mathrm{P}(a_8 \mathrm{S} a_5)))\mathrm{S} a_1))\mathrm{P}(a_{10} \mathrm{S} a_2)$.

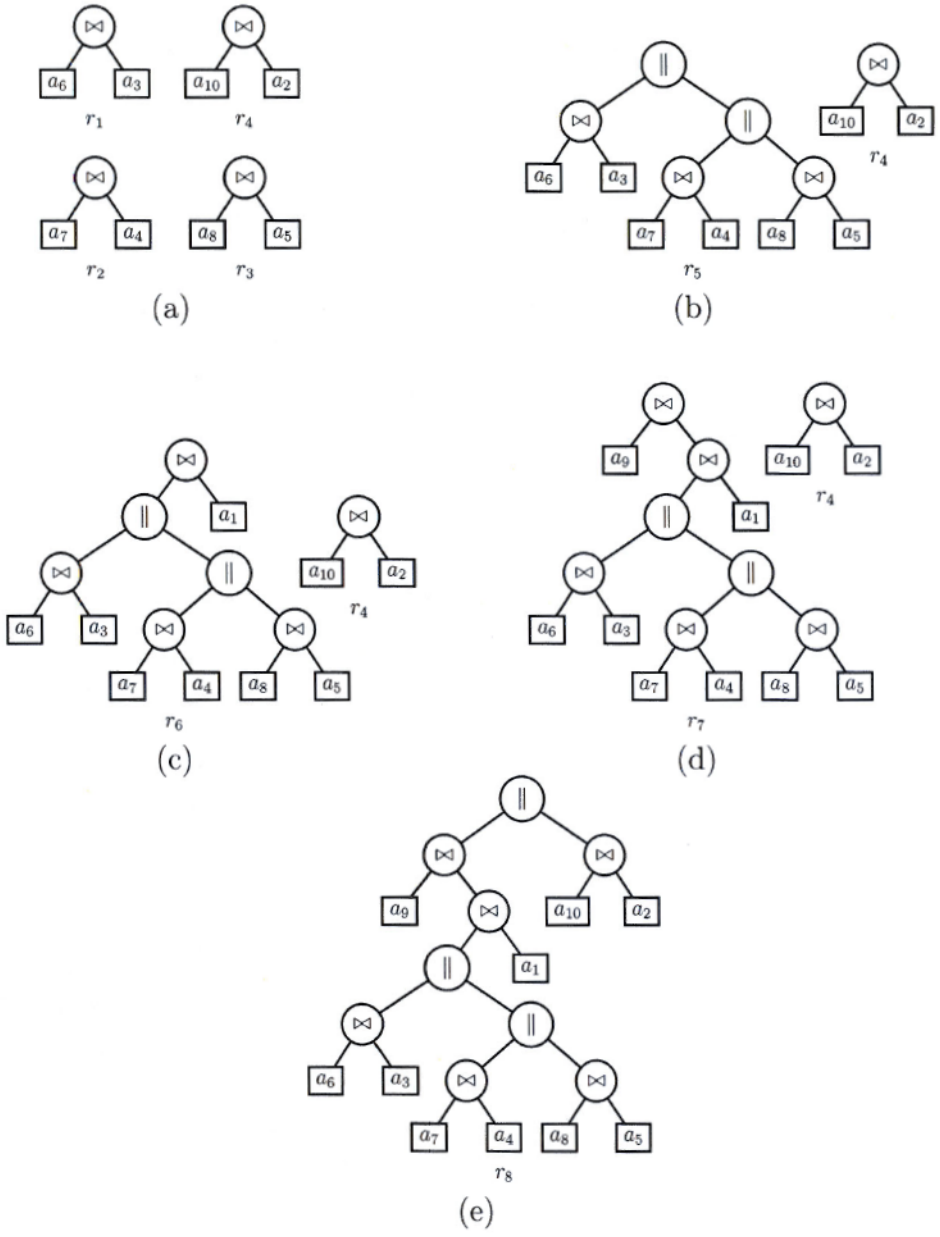


Figure 3.3: Les représentations en arbres des réductions de la figure 3.2. (a) Les réductions en série r_1, r_2, r_3 et r_4. (b) La double réduction parallèle r_5. (c) La réduction en série r_6. (d) La réduction r_7. (e) Enfin, la dernière réduction parallèle. Là, on a notre arbre de construction du graphe SP (a) de la figure 3.2.

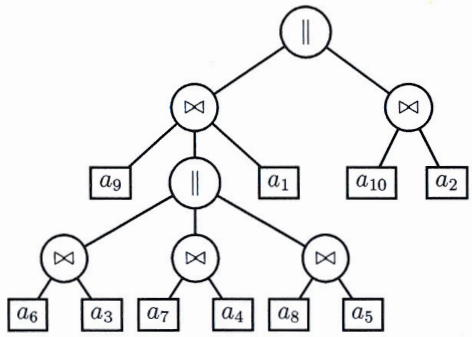


Figure 3.4: Un exemple d'arbre non binaire de construction pour le graphe (a) de la figure 3.2.

3.2 Algorithmes et complexité

Afin d'étudier la complexité des différents algorithmes que nous énonçons dans cette section, nous précisons quelques paramètres d'entrée pour mieux formuler nos résultats.

Soient un graphe SP $G = (V, E, s, t)$, avec $|G| = n$ et son arbre de construction A_G. Soient ℓ_A le nombre de feuilles de A_G (autrement dit, le nombre de graphes SP de base composant G), P_A le nombre d'opérations de composition en parallèle et S_A le nombre d'opérations de composition en série. Il est clair que $|E| = \ell_A$, car chaque arête de G est un graphe SP de base qui a été composé avec un autre graphe SP. Aussi, nous savons qu'à chaque composition en série nous perdons 1 sommet dans le décompte et lors d'une composition en parallèle nous en perdons 2, nous avons donc la relation suivante $|V| = n = 2\ell_A - S_A - 2P_A$. Par conséquent, il est clair que ce qui est linéaire sur la taille de A_G (qui vaut $\ell_A + S_A + P_A$) sera linéaire sur G de taille n.

Avant de donner les algorithmes qui implémentent les équations du chapitre 2, nous allons définir des structures de données dans la sous-section 3.2.1 et des fonctions auxiliaires de calcul et d'initialisation dans la sous-section 3.2.2 qui seront utilisées

par tous les algorithmes que nous présentons dans la sous-section 3.2.3 et la sous-section 3.2.4 plus bas.

3.2.1 Structures de données

Nous commençons par les structures de données. Nous définissons les trois structures suivantes,

Arbre. La structure d'arbre qui code l'information sur un sous-arbre induit dans un graphe SP, défini avec les attributs suivants

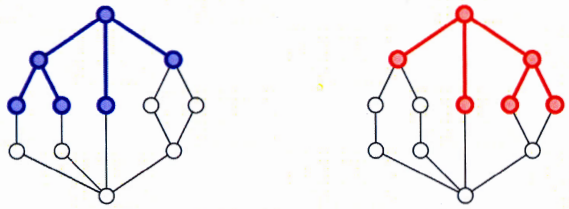
- i : la taille de l'arbre, avec $1 \leq i \leq n$
- ℓ : le nombre de feuilles
- σ : le codage du rôle de la source s dans l'arbre
- τ : le codage du rôle du puits t dans l'arbre

Foret. La structure de forêt qui code l'information sur une forêt de deux sous-arbres induits dans un graphe SP, défini comme suit,

- i : la taille de la forêt, avec $1 \leq i \leq n$
- ℓ : le nombre de feuilles
- σ : le codage du rôle de s dans la première composante
- τ : le codage du rôle de t dans la deuxième composante

Noeud. La structure d'un noeud de l'arbre A_G qui peut donc être une composition ou une feuille (un graphe SP de base), ayant les attributs

- *type :* un des trois types possibles *feuille*, *serie* ou *parallele*
- *gauche :* le noeud représentant la branche gauche du noeud courant, c'est-à-dire l'un des graphes SP composant le graphe courant
- *droit :* le noeud représentant la branche droite du noeud courant, c'est-à-dire l'autre graphe SP composant le graphe courant
- *arete_source_puits :* un booléen qui nous dit si une arête entre la source et le puits existe ou non à ce niveau de construction du graphe SP



(a) (b)

Figure 3.5: Un exemple de deux arbres différents avec la même représentation. (a) Un sous-arbre induit avec $i = 6$, $\ell = 4$, $\sigma = 0$ et $\tau = 2$. (b) Un autre sous-arbre induit avec la même représentation. Il est clair que les deux sous-arbres induits sont différents.

À noter que les structures Arbre et Foret, même si elles ont les mêmes attributs, nous garderons deux types différents pour les représenter pour des raisons de clarté et de lisibilté. Aussi, ces structures sont des représentations de sous-arbre induit ou de forêt, l'égalité de deux représentations n'implique pas l'égalité des sous-arbres induits représentés, cependant leur rôles dans les équations récursives demeurent identiques. En effet, les facteurs décisifs dans les différentes équations sont la taille i, le nombre de feuilles ℓ, σ et τ, donc deux sous-arbres ou forêts avec la même représentation donneront les mêmes résultats du point de vue des équations récursives. La figure 3.5 illustre deux sous-arbres induits différents avec des représentations identiques.

3.2.2 Fonctions auxiliaires

À présent, nous définissons les fonctions qui implémentent des critères ou des fonctions définies dans le chapitre 2 et initialisent l'attribut *arete_source_puits* de la structure de données Noeud. Soient donc les méthodes suivantes,

DISTV$(a, b : entiers)$ qui implémente le critère d'éloignement de la définition 6, qui retourne la valeur vrai si $(a, b) \in DistV$, faux sinon

RHO(a, b : *entiers*) qui implémente la fonction de perte de feuilles de la définition 6, qui retourne le nombre de feuilles qui seront perdues lors d'une composition

INITIALISERARETESP(r : Noeud) initialise récursivement, pour chaque noeud de A_G, la valeur de l'attribut *arete_source_puits* (voir algorithme 1).

Algorithme 1: Calcul de l'attribut *arete_source_puits*

```
1: procedure INITIALISERARETESP(r : Noeud)
2:     si r.type = feuille alors
3:         r.arete_source_puits ← Vrai
4:     sinon
5:         INITIALISERARETESP(r.gauche)
6:         INITIALISERARETESP(r.droit)
7:         si r.type = parallele alors
8:             g ← r.gauche.arete_source_puits
9:             d ← r.droit.arete_source_puits
10:            r.arete_source_puits ← g ou d
11:        sinon
12:            r.arete_source_puits ← Faux
13:        fin si
14:    fin si
15: fin procedure
```

AJOUTERARBRE(*dico* : *dictionnaire*, *arbre* : Arbre) ajoute un arbre à *dico* ou le réinitialise, suivant l'algorithme suivant,

La ligne 4 de la procédure AJOUTERARBRE() vérifie si la taille i de *arbre* n'est pas une clé du dictionnaire *dico*, auquel cas la procédure ajoute une nouvelle liste contenant uniquement *arbre* associée à la clé i. Dans la ligne 9 la condition $arbre \notin list$ se base sur l'égalité de représentations d'arbres. Enfin, la méthode AJOUTER() de la ligne 10, ajoute simplement un élément à une liste.

AJOUTERFORET($foret\_speciale$: *dictionnaire*, $foret$: **FORET**) cette procédure est identique à AJOUTERARBRE(), cependant au lieu d'ajouter un Arbre elle

Algorithme 2: Ajouter un arbre à un dictionnaire

```
1: procedure AJOUTERARBRE(dico : dictionnaire, arbre : Arbre)
2:     i ← arbre.i
3:     ℓ ← arbre.ℓ
4:     si i ∉ dico alors
5:         dico[i] ← [arbre]
6:     sinon
7:         list ← dico[i]
8:         rep ← list[0]
9:         si ℓ = rep.ℓ et arbre ∉ list alors
10:            list.AJOUTER(arbre)
11:        sinon si ℓ > rep.ℓ alors
12:            dico[i] ← [arbre]
13:        fin si
14:    fin si
15: fin procedure
```

ajoute une Foret en se basant sur exactement les mêmes critères.

Comme nous ajoutons uniquement les arbres ou les forêts qui ne sont pas déjà inclus, la liste des sous-arbres associée à une clé i du dictionnaire *dico* est de taille bornée. Une conséquence directe de ça est que AJOUTERARBRE() et AJOUTERFORET() ont une complexité constante. Ainsi que RHO() et DISTV() toutes deux de complexité constante.

En ce qui concerne la complexité de la méthode INITIALISERARETESP(), elle n'a besoin que d'un simple parcours de A_G pour faire les calculs nécessaires pour initiliaser la valeur de *arete_source_puits*, donc de complexité linéaire par rapport à la taille de A_G, autrement dit, $\mathcal{O}(\ell_A + P_A + S_A) = \mathcal{O}(n)$.

3.2.3 Algorithmes intermédiaires

Avant d'énoncer les algorithmes qui implémentent les équations (SA), (SF), (PA) et (PF), nous présentons quelques algorithmes intermédiaires qui mettent à jour

les sous-arbres des ensembles $\mathcal{FLT}^s_G(i)$, $\mathcal{FLT}^t_G(i)$ et $\mathcal{FLT}^{st}_G(i)$ avec $1 \leq i \leq n$, qui sont utilisés pour calculer les sous-arbres induits pleinement feuillus de G. Étant indispensables pour le calcul de la fonction feuille du chapitre 2, il est important de maintenir ces ensembles à jour après chaque composition. Nous traitons les cas de composition en série et en parallèle dans les mêmes algorithmes.

À noter que dans le cas d'une composition en parallèle de l'algorithme 3, ligne 25 à 45, le cas où le sous-arbre induit maximal contenant la source est inclus dans l'un des graphes SP correspondants aux sous-arbes de construction *r.gauche* ou *r.droit*, est implicitement considéré par les boucles des lignes 28 et 29. En effet, comme nous prenons en compte des sous-arbres induits ayant un seul sommet (dans ce cas la source seulement), les composer avec des sous-arbres induits de l'autre graphe, revient au même que de considérer uniquement ces derniers.

L'algorithme 4 est similaire à l'algorithme 3, l'intérêt est inversé entre la source et le puits (donc entre σ et τ).

La méthode COMPOSER() de l'algorithme 5 est pour éviter la redondance, elle parcourt toutes les compositions de sous-arbres induits susceptibles de donner des sous-arbres induits contenant la source et le puits.

Dans ces trois algorithmes, nous avons une complexité au plus quadratique, si nous ignorons les appels récursifs. En effet, nous avons au plus un niveau d'imbrication pour les boucles avec au plus $25n$ éléments à parcourir, ce qui nous donne une complexité $\mathcal{O}(n^2)$. Pour les appels récursifs, le calcul est fait une seule fois au niveau de chaque noeud de composition et ce pour chaque type maximal de sous-arbres induits. Plusieurs appels ne représentent aucun calcul effectif. Cette approche est facilement réalisable par des techniques de programmation dynamique (Cormen *et al.*, 2009). Pour des raisons de clarté et de lisibilité, les appels figurent sur les algorithmes, mais il s'agit d'un simple accès aux sous-arbres in-

Algorithme 3: Sous-arbres de $\mathcal{FLT}^{s}_{G}()$

```
1: fonction CalculerSAISource(r : noeud)
2:     courant ← dictionnaire vide
3:     si r.type = feuille alors
4:         courant[1] ← [Arbre(1, 0, 0, 3)]
5:         courant[2] ← [Arbre(2, 2, 1, 1)]
6:     sinon si r.type = serie alors
7:         sG ← CalculerSAISource(r.gauche)
8:         spG ← CalculerSAISourcePuits(r.gauche)
9:         sD ← CalculerSAISource(r.droit)
10:        pour g dans sG faire
11:            si r.droit.arete_source_puits et g.τ ∈ {0, 1, 2} alors
12:                τ ← 3
13:            sinon
14:                τ ← 4
15:            fin si
16:            AjouterArbre(courant, Arbre(g.i, g.ℓ, g.σ, τ))
17:        fin pour
18:        pour g dans spG faire
19:            pour d dans sD faire
20:                i ← g.i + d.i − 1
21:                ℓ ← g.ℓ + d.ℓ − rho(g.τ, d.σ)
22:                AjouterArbre(courant, Arbre(i, ℓ, g.σ, d.τ))
23:            fin pour
24:        fin pour
25:    sinon
26:        sG ← CalculerSAISource(r.gauche)
27:        sD ← CalculerSAISource(r.droit)
28:        pour g dans sG faire
29:            pour d dans sD faire
30:                si distV(g.τ, d.τ) alors
31:                    i ← g.i + d.i − 1
32:                    ℓ ← g.ℓ + d.ℓ − rho(g.σ, d.σ)
33:                    si g.σ = 0 alors
34:                        σ ← d.σ
35:                    sinon si d.σ = 0 alors
36:                        σ ← g.σ
37:                    sinon
38:                        σ ← 2
39:                    fin si
40:                    τ ← min{d.τ, g.τ}
41:                    AjouterArbre(courant, Arbre(i, ℓ, σ, τ))
42:                fin si
43:            fin pour
44:        fin pour
45:    fin si
46:    retourner courant
47: fin fonction
```

Algorithme 4: Sous-arbres de $\mathcal{FLT}^t_G()$

```
1: fonction CalculerSAIPuits(r : noeud)
2:     courant ← dictionnaire vide
3:     si r.type = feuille alors
4:         courant[1] ← [Arbre(1, 0, 3, 0)]
5:         courant[2] ← [Arbre(2, 2, 1, 1)]
6:     sinon si r.type = serie alors
7:         pG ← CalculerSAIPuits(r.gauche)
8:         spD ← CalculerSAISourcePuits(r.droit)
9:         pD ← CalculerSAIPuits(r.droit)
10:        pour d dans pD faire
11:            si r.gauche.arete_source_puits et d.σ ∈ {0, 1, 2} alors
12:                σ ← 3
13:            sinon
14:                σ ← 4
15:            fin si
16:            AjouterArbre(courant, Arbre(g.i, g.ℓ, σ, d.τ))
17:        fin pour
18:        pour g dans pG faire
19:            pour d dans spD faire
20:                i ← g.i + d.i − 1
21:                ℓ ← g.ℓ + d.ℓ − rho(g.τ, d.σ)
22:                AjouterArbre(courant, Arbre(i, ℓ, g.σ, d.τ))
23:            fin pour
24:        fin pour
25:    sinon
26:        pG ← CalculerSAIPuits(d.gauche)
27:        pD ← CalculerSAIPuits(d.droit)
28:        pour g dans pG faire
29:            pour d dans pD faire
30:                si distV(g.σ, d.σ) alors
31:                    i ← g.i + d.i − 1
32:                    ℓ ← g.ℓ + d.ℓ − rho(g.τ, d.τ)
33:                    si g.τ = 0 alors
34:                        τ ← d.τ
35:                    sinon si d.τ = 0 alors
36:                        τ ← g.τ
37:                    sinon
38:                        τ ← 2
39:                    fin si
40:                    σ ← min{d.σ, g.σ}
41:                    AjouterArbre(courant, Arbre(i, ℓ, σ, τ))
42:                fin si
43:            fin pour
44:        fin pour
45:    fin si
46:    retourner courant
47: fin fonction
```

Algorithme 5: Sous-arbres de $\mathcal{FLT}_G^{st}()$

```
1: fonction CalculerSAISourcePuits(r : noeud)
2:     courant ← dictionnaire vide
3:     si r.type = feuille alors
4:         couran[2] ← [Arbre(2, 2, 1, 1)]
5:     sinon si r.type = serie alors
6:         spG ← CalculerSAISourcePuits(r.gauche)
7:         spD ← CalculerSAISourcePuits(d.droit)
8:         pour g dans spG faire
9:             pour d dans spD faire
10:                i ← g.i + d.i − 1
11:                ℓ ← g.ℓ + d.ℓ − rho(g.τ, d.σ)
12:                ajouterArbre(courant, Arbre(i, ℓ, g.σ, d.τ))
13:            fin pour
14:        fin pour
15:    sinon
16:        spG ← CalculerSAISourcePuits(r.gauche)
17:        sD ← CalculerSAISource(r.droit)
18:        pD ← CalculerSAIPuits(r.droit)
19:        fD ← CalculerForets(r.droit)
20:        composer(courant, spG, sD, pD, fD, r.droit.arete_source_puits)
21:        spD ← CalculerSAISourcePuits(r.droit)
22:        sG ← CalculerSAISource(r.gauche)
23:        pG ← CalculerSAIPuits(r.gauche)
24:        fG ← CalculerForets(r.gauche)
25:        composer(courant, spD, sG, pG, fG, r.gauche.arete_source_puits)
26:    fin si
27:    retourner courant
28: fin fonction
```

```
29: procedure COMPOSER(courant, SP, S, P, F : dicionnaire, condition : booléen)
30:     pour g dans SP faire
31:         pour d dans S faire
32:             si DISTV(g.τ, d.τ) alors
33:                 i ← g.i + d.i − 1
34:                 ℓ ← g.ℓ + d.ℓ − RHO(g.σ, d.σ)
35:                 si d.σ = 0 alors σ ← g.σ sinon σ ← 2
36:                 AJOUTERARBRE(courant, Arbre(i, ℓ, σ, g.τ))
37:             fin si
38:         fin pour
39:         pour d dans P faire
40:             si DISTV(g.σ, d.σ) alors
41:                 i ← g.i + d.i − 1
42:                 ℓ ← g.ℓ + d.ℓ − RHO(g.τ, d.τ)
43:                 si d.τ = 0 alors τ ← g.τ sinon τ ← 2
44:                 AJOUTERARBRE(courant, Arbre(i, ℓ, g.σ, τ))
45:             fin si
46:         fin pour
47:         pour f dans F faire
48:             i ← g.i + d.i − 2
49:             ℓ ← g.ℓ + d.ℓ − RHO(g.τ, d.τ) − RHO(g.σ, d.σ)
50:             si d.σ = 0 alors σ ← g.σ sinon σ ← 2
51:             si d.τ = 0 alors τ ← g.τ sinon τ ← 2
52:             AJOUTERARBRE(courant, Arbre(i, ℓ, σ, τ))
53:         fin pour
54:         si ⌉condition alors
55:             AJOUTERARBRE(courant, g)
56:         fin si
57:     fin pour
58: fin procedure
```

duits maximaux déjà calculés par un autre appel. Par conséquent, nous pouvons considérer uniquement l'appel récursif à lui-même de chaque algorithme. Donc nous avons $|A_G|$ appels récursifs dans chaque algorithme, ce qui nous donne une complexité de $\mathcal{O}(n^2 \times |A_G|) = \mathcal{O}(n^3)$ pour les trois algorithmes.

3.2.4 Algorithmes principaux

Nous commençons par l'algorithme 6 qui implémente les cas (SF) et (PF)

Dans l'algorithme 6 nous donnons l'implémentation des différents cas (SF1), (SF2), (SF3) et (PF). À noter que le cas de base d'une forêt, tel que défini dans la sous-section 2.3.3, est couvert par le cas (SF1). En effet, lors d'une composition en série, nous sommes sûr qu'il n'y aura pas d'arête entre la source et le puits du graphe résultant, ce qui implique que nous aurons au moins une forêt de base qui se formera. De même, le cas où une composition en parallèle avec un graphe SP de base. Donc, n'ayant aucune forêt (car il existe une arête entre la source et le puits), cela fera que l'une des boucle des lignes 34 ou 35 de l'algorithme 6 sera vide, donc un dictionnaire vide sera retourné.

En terme de complexité, de même que les algorithmes de la sous-section 3.2.3, elle est quadratique pour les différents cas (SF) et (PF), en plus du parcours de A_G, l'algorithme 6 est de complexité $\mathcal{O}(n^3)$.

À présent que nous disposons de tous les outils préliminaires nécessaires pour calculer la fonction feuille L du chapitre 2, nous énonçons l'algorithme 7, qui la calcule.

Comme la boucle de la ligne 5 de la fonction feuille est linéaire, la complexité de la fonction feuille est égale à celle de la fonction CALCULERSAIPF() qui vaut $\mathcal{O}(n^2 \times |A_G|) = \mathcal{O}(n^3)$. En effet, de même que pour les algorithmes précédents, les

Algorithme 6: Forêts de $\mathcal{FLF}^{st}()$

```
1: fonction CalculerForets(r : noeud)
2:     courant ← dictionnaire vide
3:     si r.type = serie alors
4:         sG ← CalculerSAISource(r.gauche)
5:         pD ← CalculerSAIPuits(r.droit)
6:         pour g dans sG faire                                   ▷ (SF1)
7:             pour d dans pD faire
8:                 si distV(g.τ, d.σ) alors
9:                     ajouterForet(courant, Foret(g.i + d.i, g.ℓ + d.ℓ, g.σ, d.τ))
10:                fin si
11:            fin pour
12:        fin pour
13:        spG ← CalculerSAISourcePuits(r.gauche)
14:        fD ← CalculerForets(r.droit)
15:        pour g dans spG faire                                  ▷ (SF2)
16:            pour f dans fD faire
17:                i ← g.i + f.i − 1
18:                ℓ ← g.ℓ + f.ℓ − rho(g.τ, f.σ)
19:                ajouterForet(courant, Foret(i, ℓ, g.σ, f.τ))
20:            fin pour
21:        fin pour
22:        spD ← CalculerSAISourcePuits(r.droit)
23:        fG ← CalculerForets(r.gauche)
24:        pour d dans spD faire                                  ▷ (SF3)
25:            pour f dans fG faire
26:                i ← d.i + f.i − 1
27:                ℓ ← d.ℓ + f.ℓ − rho(d.σ, f.τ)
28:                ajouterForet(courant, Foret(i, ℓ, f.σ, d.τ))
29:            fin pour
30:        fin pour
```

```
31:     sinon si r.type = parallele alors
32:         fG ← CalculerForets(r.gauche)
33:         fD ← CalculerForets(r.droit)
34:         pour g dans fG faire                                   ▷ (PF)
35:             pour d dans fD faire
36:                 i ← g.i + d.i − 2
37:                 ℓ ← g.ℓ + d.ℓ − rho(g.σ, d.σ) − rho(g.τ, d.τ)
38:                 si g.σ ≠ 0 et d.σ ≠ 0 alors
39:                     σ ← 2
40:                 sinon
41:                     σ ← max{g.σ, d.σ}
42:                 fin si
43:                 si g.τ ≠ 0 et d.τ ≠ 0 alors
44:                     τ ← 2
45:                 sinon
46:                     τ ← max{g.τ, d.τ}
47:                 fin si
48:                 AjouterForet(courant, Foret(i, ℓ, σ, τ))
49:             fin pour
50:         fin pour
51:     fin si
52:     retourner courant
53: fin fonction
```

Algorithme 7: Fonction feuille L

```
1: fonction FonctionFeuille(r : Noeud)
2:     initialiserAreteSP(r)
3:     saipf ← CalculerSAIPF(r)
4:     L ← dictionnaire vide
5:     pour a dans saipf faire
6:         L[a.i] ← a.ℓ
7:     fin pour
8:     retourner L
9: fin fonction
10: fonction CalculerSAIPF(r : Noeud)
11:     courant ← dictionnaire vide
12:     si r.type = feuille alors
13:         courant[1] ← [Arbre(1, 0, 3, 0), Arbre(1, 0, 0, 3)]
14:         courant[2] ← [Arbre(2, 2, 1, 1)]
15:     sinon si r.type = serie alors
16:         saipfG ← CalculerSAIPF(r.gauche)
17:         saipfD ← CalculerSAIPF(r.droit)
18:         pG ← CalculerSAIPuits(r.gauche)
19:         sD ← CalculerSAISource(r.droit)
20:         pour a dans saipfG faire                                  ▷ (SA1)
21:             si r.droit.arete_source_puits et a.τ ∈ {0, 1, 2} alors
22:                 τ ← 3
23:             sinon
24:                 τ ← 4
25:             fin si
26:             ajouterArbre(courant, Arbre(a.i, a.ℓ, a.σ, τ))
27:         fin pour
28:         pour a dans saipfD faire                                  ▷ (SA2)
29:             si r.gauche.arete_source_puits et a.σ ∈ {0, 1, 2} alors
30:                 σ ← 3
31:             sinon
32:                 σ ← 4
33:             fin si
34:             ajouterArbre(courant, Arbre(a.i, a.ℓ, σ, a.τ))
35:         fin pour
36:         pour g dans pG faire                                      ▷ (SA3)
37:             pour d dans sD faire
38:                 i ← g.i + d.i − 1
39:                 ℓ ← g.ℓ + d.ℓ − rho(g.τ, d.σ)
40:                 ajouterArbre(courant, Arbre(i, ℓ, g.σ, d.τ))
41:             fin pour
42:         fin pour
```

```
43:     sinon
44:         saipfG ← CalculerSAIPF(r.gauche)
45:         saipfD ← CalculerSAIPF(r.droit)
46:         PA1-PA2(courant, saipfG, r.droit.arete_source_puits)        ▷ (PA1)
47:         PA1-PA2(courant, saipfD, r.gauche.arete_source_puits)       ▷ (PA2)
48:         sG ← CalculerSAISource(r.gauche)
49:         sD ← CalculerSAISource(r.droit)
50:         pour g dans sG faire                                        ▷ (PA3)
51:             pour d dans sD faire
52:                 si distV(g.τ, d.τ) alors
53:                     i ← g.i + d.i − 1
54:                     ℓ ← g.ℓ + d.ℓ − rho(g.σ, d.σ)
55:                     si g.σ ≠ 0 et d.σ ≠ 0 alors
56:                         σ ← 2
57:                     sinon
58:                         σ ← max{g.σ, d.σ}
59:                     fin si
60:                     τ ← min{g.τ, d.τ}
61:                     AjouterArbre(courant, Arbre(i, ℓ, σ, τ))
62:                 fin si
63:             fin pour
64:         fin pour
65:         pG ← CalculerSAIPuits(r.gauche)
66:         pD ← CalculerSAIPuits(r.droit)
67:         pour g dans pG faire                                        ▷ (PA4)
68:             pour d dans pD faire
69:                 si distV(g.σ, d.σ) alors
70:                     i ← g.i + d.i − 1
71:                     ℓ ← g.ℓ + d.ℓ − rho(g.τ, d.τ)
72:                     si g.τ ≠ 0 et d.τ ≠ 0 alors
73:                         τ ← 2
74:                     sinon
75:                         τ ← max{g.τ, d.τ}
76:                     fin si
77:                     σ ← min{g.σ, d.σ}
78:                     AjouterArbre(courant, Arbre(i, ℓ, σ, τ))
79:                 fin si
80:             fin pour
81:         fin pour
82:         spG ← CalculerSAISourcePuits(r.gauche)
83:         fG ← CalculerForets(r.gauche)
84:         spD ← CalculerSAISourcePuits(r.droit)
85:         fD ← CalculerForets(r.droit)
86:         PA5-PA6(courant, spG, fD)                                   ▷ (PA5)
87:         PA5-PA6(courant, spD, fG)                                   ▷ (PA6)
88:     fin si
89:     retourner courant
90: fin fonction
```

```
91: procedure PA1-PA2(courant, SAIPF : dictionnaire, condition : booleen)
92:     pour a dans SAIPF faire
93:         si condition alors
94:             si a.σ ∈ {3,4} ou a.τ ∈ {3,4} alors
95:                 si a.τ ∈ {0,1,2} alors σ ← 3 sinon σ ← a.σ
96:                 si a.σ ∈ {0,1,2} alors τ ← 3 sinon τ ← a.τ
97:                 AJOUTERARBRE(courant, Arbre(a.i, a.ℓ, σ, τ))
98:             fin si
99:         sinon
100:            AJOUTERARBRE(courant, a)
101:        fin si
102:    fin pour
103: fin procedure
104: procedure PA5-PA6(courant, SP, F : dictionnaire)
105:    pour a dans SP faire
106:        pour f dans F faire
107:            i ← a.i + f.i − 2
108:            ℓ ← a.ℓ + f.ℓ − RHO(a.σ, f.σ) − RHO(a.τ, f.τ)
109:            si f.σ = 0 alors σ ← a.σ sinon σ ← 2
110:            si f.τ = 0 alors τ ← a.τ sinon τ ← 2
111:            AJOUTERARBRE(courant, Arbre(i, ℓ, σ, τ))
112:        fin pour
113:    fin pour
114: fin procedure
```

valeurs des différents sous-arbres induits et forêts maximaux, sont déjà calculées, la fonction CALCULERSAIPF() de l'algorithme 7 ne fait qu'y accéder, ce qui fait que la complexité de CALCULERSAIPF() est uniquement dépendante de ses boucles et des appels récursifs à elle-même, ce qui nous donne une complexité de $\mathcal{O}(n^2 \times |A_G|) = \mathcal{O}(n^3)$.

Ce qui démontre, effectivement, que le problème SAIPF se résout en temps polynomial dans le cas des graphes SP. Une implémentation de ces algorithmes avec le langage de programmation Python est disponible sur un dépôt public *Gitlab* (Abdenbi, 2018).

3.3 Combinatoire

Les méthodes AJOUTERARBRE() et AJOUTERFORET() écrasent toute représentation double. Il est très facile de modifier ces méthodes pour compter le nombre de représentations, en plus d'un attribut que nous ajoutons à la structure Noeud qui sauvegarde ce compte. Cette modification nous permettra non seulement d'avoir la fonction feuille, mais aussi de savoir exactement combien de représentations, donc de sous-arbres induits pleinement feuillus, nous avons pour chaque taille i s'il y a lieu, tout en gardant une complexité polynomiale. Car c'est de simples opérations d'incrémentation et d'affectation qui seront ajoutées.

Nous pouvons même aller plus loin. En étiquetant correctement les feuilles de l'arbre de construction, nous pouvons ajouter un attribut aux types Arbre et Foret qui nous permettra d'énumérer les sommets qui constituent l'arbre ou la forêt qu'ils représentent. En terme de complexité, elle augmentera mais les algorithmes restent à temps polynomial. En effet, les ensembles des sous-arbres induits peuvent être partitionnés, pour une taille i et un nombre de feuilles maximal ℓ, en fonction des valeurs de σ et de τ en un nombre maximal de 25, ce qui équivaut au nombre

de valeurs à vérifier lors des calculs.

CONCLUSION

Dans ce mémoire nous avons étudié le problème SAIF qui interroge l'existence de sous-arbres induits ayant une certaine taille et un certain nombre de feuilles. Nous avons notamment démontré qu'il est NP-complet dans le cas d'un graphe quelconque par une réduction polynomiale du problème du STABLE au problème SAIF. Nous nous sommes naturellement intéressés au problème d'optimisation associé, à savoir le problème SAIPF qui cherche le nombre maximal de feuilles pour toutes les tailles possibles. Évidemment ce problème, dans le cas général, appartient à la famille des problèmes dit NP-difficiles, ce qui se traduit souvent par des algorithmes de résolution avec des complexités temporelles exponentielles.

À l'instar de l'approche des auteurs dans (Blondin Massé *et al.*, 2017) où ils ont considéré la famille des graphes arbres où le problème SAIPF s'est avéré résoluble en temps et espace polynomiaux, nous avons considéré la famille des graphes séries-parallèles comme candidate potentielle pour des algorithmes de résolution en temps polynomial. Nous avons, en effet, réussi à résoudre le problème SAIPF dans un temps polynomial sur cette famille de graphes. Au travers d'équations récursives construites notamment grâce à un résultat d'optimalité sur les graphes séries-parallèles, qui stipule que le nombre maximal de feuilles dans les sous-arbres induits, notion fondamentale pour le problème SAIFP, se transmettait et était conservé, sous certaines conditions, par les compositions inhérentes aux graphes séries-parallèles. L'ensemble des algorithmes qui implémentent ces équations sont tous polynomiaux, ce qui est l'objectif de ce mémoire à savoir identifier une famille de graphes où le problème SAIPF se résout en temps polynomial.

Comme perspective aux résultats de ce mémoire, la famille des graphes planaires, dont les graphes séries-parallèles sont une sous-famille, est une excellente famille de graphes à considérer. En effet, de nombreux problèmes NP-difficiles ont des solutions polynomiales, comme justement nous venons de le voir pour les graphes séries-parallèles. Il serait donc intéressant d'explorer les possibilités que cette famille, ou l'une de ses sous-familles, offre pour le problème SAIPF. Notamment, dans la continuité des résultats de ce mémoire, nous pouvons considérer pour un future travail, la famille des graphes *trois-terminaux séries-parallèles*. De façon analogue à la famille des graphes *deux-terminaux séries-parallèles* l'autre appellation des graphes séries-parallèles, les graphes trois-terminaux se composent au travers de 3 sommets particuliers identifiés pour chaque graphe (Nishizeki et Saito, 1975). On y retrouve les compositions en série et en parallèle, ainsi qu'une définition récurrente de ces graphes basée sur ces compositions, ce qui en fait une excellente famille pour une généralisation de nos résultats.

Enfin, dans une plus large perspective, il serait intéressant de considérer le problème à partir d'un paramètre appelé *largeur arborescente* d'un graphe. Sans entrer dans le formalisme de sa définition, nous pouvons voir la largeur arborescente d'un graphe, comme un entier qui mesure une distance entre n'importe quel graphe simple et un arbre (Williamson et Shmoys, 2011). Il a été prouvé que la largeur arborescente est bornée pour plusieurs familles de graphes (Robertson et Seymour, 1986; Bodlaender et de Fluiter, 1996a) :

- les arbres, avec une largeur arborescente bornée par 1 ;
- les graphes séries-parallèles, avec une largeur arborescente bornée par 2 ;
- les graphes planaires extérieurs, avec une largeur arborescente aussi bornée par 2.

Ainsi, ayant des algorithmes en temps polynomial qui résolvent le problème SAIPF pour les arbres et les graphes séries-parallèles, il est tout naturel de considérer la

largeur arborescente comme un autre paramètre d'entrée. Dans ce cas, nous aurons à faire une analyse de *complexité paramétrée*. Cette technique ne considère pas uniquement la taille d'un problème comme unique paramètre d'analyse de la complexité, mais y ajoute un ou plusieurs paramètres de l'instance d'entrée (Cygan *et al.*, 2015). En l'occurrence, pour le problème SAIPF, considérer la largeur arborescente d'un graphe comme un paramètre d'analyse de complexité, afin de développer un algorithme paramétré pour une solution plus générale. Ce mémoire et d'autres travaux sur les sous-arbres induits pleinement feuillus, constitueront un pas vers une telle solution.

RÉFÉRENCES

Abdenbi, M. (2018). The maximal number of leaves in induced subtrees of series-parallel graphs. `https://gitlab.com/moussa.abdenbi/flis-graphs-SP`.

Abdenbi, M., Blondin Massé, A. et Goupil, A. (2018). On the maximal number of leaves in induced subtrees of series-parallel graphs. *ceur-ws*, *2113*(1), 24 – 31. Récupéré de `http://ceur-ws.org/Vol-2113/paper1.pdf`

Alekseev, V. E., Lozin, V., Malyshev, D. et Milanič, M. (2008). The maximum independent set problem in planar graphs. Dans E. Ochmański et J. Tyszkiewicz (dir.). *Mathematical Foundations of Computer Science 2008*, 96–107., Berlin, Heidelberg. Springer Berlin Heidelberg.

AlSudairy, N. M. K., Raghavan, V. V., Hafez, A. M. et Mathkour, H. I. (2011). Connection subgraphs : A survey. *Journal of Applied Sciences*, *11*, 3221 – 3232. `http://dx.doi.org/https://scialert.net/abstract/?doi=jas.2011.3221.3232`. Récupéré de `https://scialert.net/abstract/?doi=jas.2011.3221.3232`

Blondin Massé, A., de Carufel, J., Goupil, A., Lapointe, M., Nadeau, É. et Vandomme, É. (2017). Fully leafed induced subtrees.

Blondin Massé, A., de Carufel, J., Goupil, A. et Samson, M. (2018). Fully leafed tree-like polyominoes and polycubes. Dans *Combinatorial Algorithms*, volume 10765 de *Lect. Notes Comput. Sci.* 28th International Workshop, IWOCA 2017, Newcastle, NSW, Australia, Springer.

Bodlaender, H. L. et de Fluiter, B. (1996a). Parallel algorithms for series parallel graphs. Dans J. Diaz et M. Serna (dir.). *Algorithms — ESA '96*, 277–289., Berlin, Heidelberg. Springer Berlin Heidelberg.

Bodlaender, H. L. et de Fluiter, B. (1996b). Parallel algorithms for series parallel graphs. Dans J. Diaz et M. Serna (dir.). *Algorithms — ESA '96*, 277–289., Berlin, Heidelberg. Springer Berlin Heidelberg.

Brylawski, T. (1971). A combinatorial model for series-parallel networks. *American Mathematical Society*, *154*.

Cai, H., Zheng, V. W. et Chang, K. C.-C. (2017). A comprehensive survey of graph embedding : Problems, techniques and applications. *CoRR, abs/1709.07604*.

Chen, G. H., Kuo, M. T. et Sheu, J. P. (1988). An optimal time algorithm for finding a maximum weight independent set in a tree. *BIT Numerical Mathematics*, *28*(2), 353–356.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. et Stein, C. (2009). *Introduction to Algorithms, Third Edition* (3rd éd.). The MIT Press.

Cuissart, B. et Hébrard, J.-J. (2005). A direct algorithm to find a largest common connected induced subgraph of two graphs. Dans L. Brun et M. Vento (dir.). *Graph-Based Representations in Pattern Recognition*, 162–171., Berlin, Heidelberg. Springer Berlin Heidelberg.

Cygan, M., Fomin, F. V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M. et Saurabh, S. (2015). *Parameterized Algorithms* (1st éd.). Springer Publishing Company, Incorporated.

de Mier, A. et Noy, M. (2009). On the maximum number of cycles in outerplanar and series-parallel graphs. *Electronic Notes in Discrete Mathematics*, *34*, 489 – 493. European Conference on Combinatorics, Graph Theory and Applications (EuroComb 2009), `http://dx.doi.org/https://doi.org/10.1016/j.endm.2009.07.081`. Récupéré de `http://www.sciencedirect.com/science/article/pii/S1571065309001231`

Diestel, R. (2010). *Graph theory* (fourth éd.), volume 173 de *Graduate Texts in Mathematics*. Springer, Heidelberg.

Duffin, R. (1965). Topology of series-parallel networks. *Journal of Mathematical Analysis and Applications*, *10*(2), 303 – 318. `http://dx.doi.org/https://doi.org/10.1016/0022-247X(65)90125-3`. Récupéré de `http://www.sciencedirect.com/science/article/pii/0022247X65901253`

Eppstein, D. (1992). Parallel recognition of series-parallel graphs. *Inf. Comput.*, *98*, 41–55.

Erdős, P., Saks, M. et Sós, V. T. (1986). Maximum induced trees in graphs. *J. Combin. Theory Ser. B*, *41*(1), 61–79.

Garey, M. R. et Johnson, D. S. (1990). *Computers and Intractability; A*

Guide to the Theory of NP-Completeness. New York, NY, USA : W. H. Freeman & Co.

Giacomo, E. D., Liotta, G. et Meijer, H. (2005). Computing straight-line 3d grid drawings of graphs in linear volume. *Computational Geometry, 32*(1), 26 – 58. http://dx.doi.org/https://doi.org/10.1016/j.comgeo.2004.11.003. Récupéré de http://www.sciencedirect.com/science/article/pii/S092577210500012X

Golumbic, M. C. (1985). Interval graphs and related topics. *Discrete Mathematics, 55*(2), 113 – 121. http://dx.doi.org/https://doi.org/10.1016/0012-365X(85)90039-1. Récupéré de http://www.sciencedirect.com/science/article/pii/0012365X85900391

Hong, S.-H., Eades, P. et Lee, S.-H. (2000). Drawing series parallel digraphs symmetrically. *Computational Geometry, 17*(3), 165 – 188. http://dx.doi.org/https://doi.org/10.1016/S0925-7721(00)00020-1. Récupéré de http://www.sciencedirect.com/science/article/pii/S0925772100000201

K. Takamizaw, T. N. et Saito, N. (1982). Linear-time computability of combinatorial problems on series-parallel graphs. *Journal of the ACM, 29*(3), 623–641.

Karp, R. M. (1972). *Reducibility among Combinatorial Problems*, Dans R. E. Miller, J. W. Thatcher, et J. D. Bohlinger (dir.). *Complexity of Computer Computations : Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department*, (p. 85–103). Springer US : Boston, MA

Kennedy, J. W., Quintas, L. V. et Sysło, M. M. (1985). The theorem on planar graphs. *Historia Mathematica, 12*(4), 356 – 368. http://dx.doi.org/https://doi.org/10.1016/0315-0860(85)90045-X. Récupéré de http://www.sciencedirect.com/science/article/pii/031508608590045X

Korenblit, M. et Levit, V. (2011). On the structure of maximum series-parallel graphs. *1389.*

Krissinel, E. B. et Henrick, K. (2004). Common subgraph isomorphism

detection by backtracking search. *Softw. Pract. Exper.*, *34*(6), 591–607. http://dx.doi.org/10.1002/spe.588. Récupéré de http://dx.doi.org/10.1002/spe.588

Maxwell, S., Chance, M. R. et Koyutürk, M. (2014). Efficiently enumerating all connected induced subgraphs of a large molecular network. Dans A.-H. Dediu, C. Martín-Vide, et B. Truthe (dir.). *Algorithms for Computational Biology*, 171–182., Cham. Springer International Publishing.

McKee, T. et McMorris, F. (1999). *Topics in Intersection Graph Theory.* Society for Industrial and Applied Mathematics. http://dx.doi.org/10.1137/1.9780898719802. Récupéré de https://epubs.siam.org/doi/abs/10.1137/1.9780898719802

Nishizeki, T. et Saito, N. (1975). Necessary and sufficient condition for a graph to be three-terminal series-parallel. *IEEE Transactions on Circuits and Systems*, *22*(8), 648–653. http://dx.doi.org/10.1109/TCS.1975.1084108

Nishizeki, T., Vygen, J. et Zhou, X. (2001). The edge-disjoint paths problem is np-complete for series–parallel graphs. *Discrete Applied Mathematics*, *115*(1), 177 – 186.

Robertson, N. et Seymour, P. (1986). Graph minors. II. algorithmic aspects of tree-width. *Journal of Algorithms*, *7*(3), 309–322. http://dx.doi.org/10.1016/0196-6774(86)90023-4. Récupéré de https://doi.org/10.1016%2F0196-6774%2886%2990023-4

Schoenmakers, L. A. (1995). *A New Algorithm for the Recognition of Series Parallel Graphs.* Rapport technique, Amsterdam, The Netherlands, The Netherlands.

Sun, Z., Wang, H., Wang, H., Shao, B. et Li, J. (2012). Efficient subgraph matching on billion node graphs. *Proc. VLDB Endow.*, *5*(9), 788–799. http://dx.doi.org/10.14778/2311906.2311907. Récupéré de http://dx.doi.org/10.14778/2311906.2311907

Takamizawa, K., Nishizeki, T. et Saito, N. (1982). Linear-time computability of combinatorial problems on series-parallel graphs. volume 29, 623–641., New York, NY, USA. ACM. http://dx.doi.org/10.1145/322326.322328. Récupéré de http://doi.acm.org/10.1145/322326.322328

Valdes, J., Tarjan, R. E. et Lawler, E. L. (1982). The recognition of series parallel digraphs. *SIAM Journal on Computing*, *11*(2), 298–313. http://dx.doi.org/10.1137/0211023. Récupéré de https://doi.org/10.1137/0211023

Wasa, K., Arimura, H. et Uno, T. (2014). Efficient enumeration of induced subtrees in a K-degenerate graph. In *Algorithms and computation*, volume 8889 de *Lect. Notes in Comput. Sci.* 94–102. Springer, Cham

Williamson, D. P. et Shmoys, D. B. (2011). *The Design of Approximation Algorithms* (1st éd.). New York, NY, USA : Cambridge University Press.

Woeginger, G. J. (2003). *Exact Algorithms for NP-Hard Problems : A Survey*, Dans M. Jünger, G. Reinelt, et G. Rinaldi (dir.). *Combinatorial Optimization — Eureka, You Shrink! : Papers Dedicated to Jack Edmonds 5th International Workshop Aussois, France, March 5–9, 2001 Revised Papers*, (p. 185–207). Springer Berlin Heidelberg : Berlin, Heidelberg

Zaki, M. J. (2002). Efficiently mining frequent trees in a forest. Dans *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, 71–80., New York, NY, USA. ACM. `http://dx.doi.org/10.1145/775047.775058`. Récupéré de `http://doi.acm.org/10.1145/775047.775058`