Contraintes de Partitionnement par des Arbres

Nicolas Beldiceanu, Xavier Lorca, Irit Katriel

Contraintes de Partitionnement par des Arbres

Nicolas Beldiceanu[1] **Irit Katriel**[2] **Xavier Lorca**[1]

[1] École des Mines de Nantes, LINA FRE CNRS 2729,
FR–44307 Nantes Cedex 3, France
[2] BRICS*, University of Aarhus, Åbogade 34, Århus, Denmark.

{nbeldiceanu,xlorca}@emn.fr irit@daimi.au.dk

Résumé

Nous présentons deux contraintes qui partitionnent les sommets d'un graphe non-orienté $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, où $|\mathcal{V}| = n$ et $|\mathcal{E}| = m$, en un ensemble d'arbres disjoints. La première contrainte, *resource-forest*, spécifie que chaque arbre dans la forêt doit contenir au moins un sommet *ressource*. L'ensemble des ressources est un sous-ensemble $\mathtt{R} \subseteq \mathcal{V}$. Cette contrainte est la contrepartie non-orienté de la contrainte d'arbre introduite dans [2], qui partitionne un graphe orienté en une forêt d'arbres orientés où seulement certains sommets peuvent être des racines. Nous décrivons un algorithme de consistance-hybride pour la contrainte *resource-forest* ayant une complexité de $\mathcal{O}(m + n)$. Ceci constitue donc une amélioration de la complexité en $\mathcal{O}(mn)$ connue pour le cas orienté. La seconde constrainte, *proper-forest*, est une variante de la première ne nécessitant pas que chaque arbre contienne une ressource. Cependant, tout arbre construit doit être un arbre *propre*, i.e., un arbre contenant au moins deux sommets. Nous avons développé un algorithme de consistance-hybride ayant une complexité en $\mathcal{O}(mn)$ au pire des cas, et en $\mathcal{O}(m\sqrt{n})$ dans la plupart des autres cas.

1 Introduction

Les contraintes décrivant les propriétés de graphes sont considérées comme une nouvelle étape de la recherche en programmation par contraintes. Quelques exemples connus sont les contraintes de *chemin Hamiltonien* et *d'arbre couvrant* proposées dans ALICE [9], qui ont été suivies des contraintes de *cycle* [1] et de *chemin* [14] introduites dans les premières versions de CHIP [7] et d'Ilog Solver [11]. Un exemple plus récent est la contrainte d'arbre *tree*(NTREE, VER) introduite par [2], qui prend en paramètres une variable entière NTREE et un graphe orienté décrit par sa liste de sommets VER. Certains sommets sont décrits comme des "racine potentielles" et la contrainte impose que le graphe construit contienne NTREE arbres orientés, chacun étant enraciné sur une "racine potentielle".

Un problème classique de design de réseaux consiste à considérer un graphe non-orienté $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ où $\mathtt{R} \subseteq \mathcal{V}$ un ensemble de sommets correspond à un type de ressources, par exemple des imprimantes. Les sommets restant représentent des tâches (ou clients, ou encore utilisateurs). Le problème est de couvrir tous les sommets du graphe par des arbres (réseaux) tels que chaque arbre contienne au moins un sommet ressource (ce qui signifie par exemple que chaque réseau doit avoir au moins une imprimante). Naturellement, il est possible de remplacer chaque arête par deux arcs maintenant l'équivalence sémantique (i.e., une arête (a, b) est remplacée par un arc (a, b) et un arc (b, a)), alors la contrainte d'arbre *tree*, introduite dans [2], autorise un filtrage complet avec une complexité en $\mathcal{O}(mn)$. Cependant, raisonner sur un graphe non-orienté est très souvent plus simple que de raisonner sur un équivalent orienté. En effet, nous allons montrer que la contrainte *resource-forest*, qui est la contrepartie non-orienté de la contrainte *tree*, effectue un filtrage équivalent pour une complexité en $\mathcal{O}(m + n)$.

Nous présentons maintenant une autre variante du problème, la contrainte *proper-forest*(NTREE, VER) qui spécifie que le graphe doit être une forêt contenant NTREE *arbres propres*, comme le définit A. CAYLEY en 1889 [6], un arbre propre est un graphe connexe, sans cycles, contenant au moins deux sommets. Il faut remarquer que dans le cas de la contrainte *proper-forest* la notion de ressources n'existe pas (ou, de manière équivalente on peut considérer que tout

*Basic Research in Computer Science, funded by the Danish National Research Foundation.

sommet est une ressource). Elle peut être utilisée pour le design de réseaux insensibles aux coupures, par exemple, chaque réseau doit contenir au moins deux ordinateurs de sorte qu'il puisse se soutenir l'un l'autre. La contrainte *proper-forest* est plus complexe que son pendant avec ressources. En effet, nous détaillerons un algorithme de filtrage ayant une complexité en $\mathcal{O}(mn)$, dominée par la complexité de déterminer les arêtes du graphe qui appartiennent à au moins un couplage de cardinalité maximum. Comme nous le verrons, ce pire cas se produit lorsque le domaine de `NTREE` est fixé à une certaine valeur. Dans tous les autres cas, l'algorithme est basé sur la recherche d'un couplage de cardinalité maximum dans le graphe, ce qui est effectué avec une complexité en $\mathcal{O}(m\sqrt{n})$.

Comme les deux contraintes mettent en jeu à la fois des variables entières et des variables ensemblistes, nos algorithmes de filtrage atteignent la *consistance-hybride*, qui est un type de consistance, introduite par Bessière *et al.* [5], particulièrement adaptée à ce contexte. Elle sera définie formellement dans la Section 2, mais intuitivement, la consistance-hybride signifie que toute variable entière est arc-consistante et toute variable ensembliste est consistante aux bornes.

Dans la suite, la Section 2 fournit les notions nécessaires de la programmation par contraintes et de la théorie des graphes. La Section 3 introduit ensuite les contraintes *resource-forest* et *proper-forest*. Les Sections 4 et 5 présentent respectivement les algorithmes de filtrage des contraintes *resource-forest* et *proper-forest*. Enfin, la Section 6 est un résumé des résultats théoriques connus sur le filtrage des contraintes de partitionnement par des arbres.

2 Préliminaires

Nous rappelons dans cette section quelques notions de programmation par contraintes et de théorie des graphes qui seront utilisées dans la suite de ce papier.

Definition 1. *Une* variable entière V_i *varie sur un ensemble fini d'entiers notés* $\mathcal{D}(V_i)$. *Les valeurs extrêmes de* $\mathcal{D}(V_i)$ *sont notées* $\min(V_i)$ *et* $\max(V_i)$.

Definition 2. *Le domaine d'une* variable ensembliste V_s *est un ensemble d'ensembles d'entiers. Il est décrit par sa* borne inférieure $\underline{V_s}$ *et sa* borne supérieure $\overline{V_s}$. *Tout ensemble de* $\underline{V_s}$ *est contenu dans* V_s *et tout ensemble de* V_s *est contenu dans* $\overline{V_s}$. *Lorsque la variable ensembliste est* fixée *alors* $\underline{V_s} = \overline{V_s}$. *Les valeurs dans* $\underline{V_s}$ *sont dites* valeurs obligatoires *et les valeurs de* $\overline{V_s} \setminus \underline{V_s}$ *sont dites* valeurs potentielles.

Definition 3 (Consistance-Hydride [5])**.** *Une contrainte* $\mathcal{C}$ *définie sur des variables entières* $V_1^d, \ldots, V_l^d$ *et l'ensemble des variables* $V_{l+1}^s, \ldots, V_n^s$ *est* hybride-consistante *ssi :*

1. *Pour chaque paire* (V^d, v) *telle que* V^d *est une variable entière de* $\mathcal{C}$ *et* $v \in \mathcal{D}(V^d)$*, il existe au moins une solution respectant* $\mathcal{C}$ *dans laquelle la valeur* v *est assignée à* V^d.
2. *Pour toute paire* (V^s, v) *telle que* V^s *est une variable ensembliste de* $\mathcal{C}$*, si* $v \in \underline{V^s}$ *alors* v *appartient à l'ensemble assigné à* V^s *dans* toutes *les solutions respectant* $\mathcal{C}$ *et si* $v \in \overline{V^s} \setminus \underline{V^s}$ *alors* v *appartient à l'ensemble assigné à* V^s *dans au moins une solution et est exclue de cet ensemble dans au moins une solution.*

Definition 4. Notion de théorie des graphes [4] :
Soit $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *un graphe non-orienté. Un* chemin *dans* $\mathcal{G}$ *est une séquence de sommets, tels que deux sommets consécutifs sont liés par une arête. Un chemin est dit* élémentaire *si tout sommet apparaît au plus une fois dans la séquence le définissant. Un* isthme *dans* $\mathcal{G}$ *est une arête* $e \in \mathcal{E}$ *dont le retrait augmente le nombre maximal de composantes connexes de* $\mathcal{G}$. *Un* couplage *dans* $\mathcal{G}$ *est un ensemble d'arêtes,* $M \in \mathcal{E}$*, telles que chaque sommet dans* $\mathcal{V}$ *est incident à au moins une arête de* M.

3 Les Contraintes *resource-forest* et *proper-forest*

Dans cette section, nous allons définir et motiver les contraintes *resource-forest* et *proper-forest*, introduire les graphes correspondant, les définir de manière formelle et fournir des exemples qui illustrent la sémantique de chaque contrainte ainsi que le problème du filtrage pour atteindre la consistance-hybride.

Dans beaucoup de problèmes de partitionnement de graphes, l'ensemble de sommets du graphe est l'union d'un ensemble de sommets *ressources* et d'un ensemble de sommets *tâches*. Indépendamment du pattern utilisé pour couvrir le graphe, cette distinction entre deux types de sommets traduit le besoin de maintenir l'existence d'au moins un sommet ressource dans chaque partition. Cette distinction entre les sommets ressources et les sommets tâches a, par exemple, déjà été introduite dans la contrainte de *cycle* [1]. Un exemple d'application pour la contrainte de *cycle* est le problème de planification de tournées de véhicules qui consiste à allouer un ensemble de camions (ressources) livrant des marchandises à un ensemble de magasins (tâches) en attente de produits. Dans notre cas, la contrainte *resource-forest* peut être utilisée pour modéliser le problème d'allocation de ressources matérielles partagées dans un réseau. Ici, une ressource représentera un type de matériel informatique qui doit être partagé sur un réseau (e.g., imprimantes) et une tâche représentera le matériel susceptible d'utiliser cette ressource (e.g., un ordinateur). Une solution (i.e., une forêt) est un réseau dans lequel chaque ordinateur sera connecté à au moins une imprimante.

En 1889, A. CAYLEY [6] introduisit la définition formelle d'un arbre comme celle d'un graphe connexe, sans cycles, contenant au moins deux sommets. Dans la suite, nous appellerons les arbres définis par Cayley des *arbres propres*. Alors, une *forêt propre* sera définie comme un ensemble d'arbres propres. Ainsi, la contrainte *proper-forest* partitionne les sommets d'un graphe non-orienté en un ensemble de sommets disjoints formant des arbres propres.

Formellement, les deux contraintes *resource-forest* et *proper-forest* sont définies sur une variable entière NTREE et un tableau VER qui est une représentation du graphe sous forme d'une liste d'adjacence. Chaque item $v \in$ VER possède les attributs suivants :

- I est un entier entre 1 et n qui peut être interprété comme le *nom* de v.
- N est une variable ensembliste dont les éléments sont des entiers (i.e., les noms des sommets) entre 1 et n. Les bornes inférieures et supérieures de N peuvent être interprétées respectivement comme l'ensemble des *voisins obligatoires* et l'ensemble des *voisins possibles ou obligatoires* de v.
- R (seulement pour la contrainte *resource-forest*) est un booléen qui est vrai si le sommet est un sommet ressource et faux si le sommet est une tâche.

Notation : Quelque soit $1 \leq i \leq n$, VER$[i]$ désigne le i-ème item de la collection VER, alors que VER$[i]$.I, VER$[i]$.N, et VER$[i]$.R représentent respectivement les attributs I, N et R de VER$[i]$.

Lorsque l'on aborde les contraintes globales, il est très souvent plus facile de raisonner directement sur le graphe modélisant cette dernière que de raisonner directement sur ses variables (voir, e.g., les contraintes *cycle* [1], *chemin* [13, 14], et *alldifferent* [12]). Dans le cadre des contraintes *resource-forest* et *proper-forest*, le modèle de graphe est évident : il s'agit d'un graphe non-orienté $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ dans lequel les sommets représentent des éléments de VER et les arêtes représentent la relation de voisinage entre ces derniers. Chaque arête du graphe est *typée* (solide ou pointillé) pour indiquer si elle représente une relation de voisinage obligatoire (solide) ou potentielle (pointillé).

Sachant qu'il s'agit d'un pré-traitement linéaire en temps et relativement évident, nous supposerons dans la suite du papier que le graphe associé ne contient aucune boucle et que l'ensemble des sommets de N est symétrique, i.e., $i \in$ VER$[j].\underline{\text{N}} \Leftrightarrow j \in$ VER$[i].\underline{\text{N}}$ (dans ce cas nous dirons que i et j sont des *voisins obligatoires*) et $i \in$ VER$[j].\overline{\text{N}} \Leftrightarrow j \in$ VER$[i].\overline{\text{N}}$ (dans ce cas, si i et j ne sont pas des voisins obligatoires alors se sont des *voisins possibles*). Il faut remarquer que cette étape de pré-traitement peut découvrir que la contrainte n'a pas de solution. Ceci peut se produire si $i \in$ VER$[i].\underline{\text{N}}$ (il y a une boucle obligatoire) ou $\exists i, j : i \in$ VER$[j].\underline{\text{N}} \wedge j \notin$ VER$[i].\overline{\text{N}}$ (i est un voisin obligatoire de j mais j n'est pas un voisin possible de i). Formellement, le graphe associé aux contraintes *resource-forest* et *proper-forest* est défini de la manière suivante :

Definition 5. *Quelque soit une contrainte* resource-forest *ou une contrainte* proper-forest*, le* graphe associé *est le graphe non-orienté* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *pour lequel* $\mathcal{V} = \{v_i : i \in [1, n]\}$ *et* $(v_i, v_j) \in \mathcal{E}$ *ssi* $i \in$ VER$[j].\overline{\text{N}} \wedge j \in$ VER$[i].\overline{\text{N}}$. *Nous distinguons les arêtes solides de celles en pointillé de la manière suivante : l'arête* $(v_i, v_j) \in \mathcal{E}$ *est* solide *si* i *et* j *sont des voisins obligatoires et elle sera en* pointillé *si* i *et* j *sont des voisins possibles. Pour finir, nous noterons le nombre d'arêtes,* $|\mathcal{E}|$*, dans le graphe par* m.

Dans le cas de la contrainte resource-forest*, nous ferons une distinction entre les sommets ressources et les sommets tâches ; l'ensemble des sommets ressources* R *est* $\{v_i :$ VER$[i]$.R $= \text{true}\}$. *Tout sommet dans* $\mathcal{V} \setminus$ R *est un sommet tâche.*

La contrainte *resource-forest* spécifie que le graphe associé doit être une forêt dans laquelle chaque arbre contient au moins une ressource. La contrainte *proper-forest* spécifie, quand à elle, que le graphe associée doit être une forêt propre. Formellement, on dira :

Definition 6. *Une contrainte* resource-forest(NTREE, VER) *est satisfaite ssi les conditions suivantes sont vérifiées :*

(1) $\forall i \in [1, n]$: VER$[i]$.I $= i$,

(2) $\forall i, j \in [1, n] : i \in$ VER$[j]$.N $\Leftrightarrow j \in$ VER$[i]$.N *(i.e., la relation de voisinage est symétrique),*

(3) *Le graphe* $\mathcal{G}$ *associé contient* NTREE *composantes connexes telles que chacune contient au moins un sommet de* R *et ne contient pas de cycles.*

Definition 7. *Une contrainte* proper-forest(NTREE, VER) *est satisfaite ssi es conditions suivantes sont vérifiées :*

(1) $\forall i \in [1, n]$: VER$[i]$.I $= i$,

(2) $\forall i, j \in [1, n] : i \in$ VER$[j]$.N $\Leftrightarrow j \in$ VER$[i]$.N *(i.e., la relation de voisinage est symétrique),*

(3) *Le graphe* $\mathcal{G}$ *associé est une forêt de* NTREE *arbres propres disjoints.*

L'exemple suivant sera utilisé tout au long de ce papier.

example 1. *La partie (A) de la Figure 1 montre le graphe* $\mathcal{G}$ *donné en entrée, où les arêtes obligatoires sont solides et les autres sont en pointillé. Les parties (B) et (C) de cette figure montrent deux solutions possibles pour la contrainte* resource--forest*, la première avec deux arbres et la deuxième avec trois. Les parties (B) et (D) montrent deux solutions possibles pour la contrainte* proper-forest*, avec respectivement deux et sept arbres propres.*

Un algorithme atteignant la consistance-hybride pour la contrainte resource-forest *sur* $\mathcal{G}$ *devrait découvrir qu'au vue des éléments présents dans le domaine de* NTREE*, l'arête marquée*

par M_r, i.e., l'arête $(6, 8)$, est obligatoire et que l'arête $(5, 7)$ (marquée par F_r) est interdite. De plus, il devrait restreindre le domaine de NTREE *à l'intersection entre ses valeurs précédentes et $\{2, 3\}$. Si, en entrée, $\mathcal{D}(\texttt{NTREE}) = \{2\}$, l'algorithme devrait aussi découvrir que l'arête marquée par J_r, i.e., l'arête $(13, 15)$, est obligatoire. Dans la suite, la Section 4 justifiera ce filtrage.*

Un algorithme atteignant la consistance-hybride pour la contrainte proper-forest *sur $\mathcal{G}$ devrait découvrir que l'arête $(13, 15)$ (marquée par M_p) est obligatoire et que l'arête $(5, 7)$ (marquée par F_p) est interdite. Ensuite, il devrait restreindre le domaine de* NTREE *à l'intersection entre ses valeurs précédentes et $\{2, 3, 4, 5, 6, 7\}$. Si, en entrée, $\mathcal{D}(\texttt{NTREE}) = \{7\}$, l'algorithme devrait aussi découvrir que les arêtes marquées par I_p, i.e., $(2, 3)$, $(3, 5)$, $(4, 7)$, $(6, 8)$, $(11, 13)$ et $(12, 14)$ sont interdites. Dans la suite, la Section 5 justifiera ce filtrage.*

Avant de pouvoir décrire les algorithmes de filtrage des contraintes *resource-forest* et *proper-forest*, nous devons définir le graphe obligatoire $\mathcal{G}_{TRUE}$ et le graphe possible $\mathcal{G}_{MAYBE}$ associé au graphe $\mathcal{G}$. Un exemple est fournit dans la Figure 2.

Definition 8. *(*Graphe obligatoire*) Etant donné une contrainte* resource-forest *ou une contrainte* proper-forest *et son graphe associé $\mathcal{G}$, le graphe $\mathcal{G}_{TRUE}$ contient toutes les arêtes qui doivent être dans la forêt. Formellement, $\mathcal{G}_{TRUE} = (\mathcal{V}, \mathcal{E}_{TRUE})$, où $\mathcal{E}_{TRUE}$ est l'ensemble des arêtes solides de $\mathcal{G}$.*

Definition 9. *(*Graphe possible*) Etant donné une contrainte* resource-forest *ou une contrainte* proper--forest *et son graphe associé $\mathcal{G}$, le graphe $\mathcal{G}_{MAYBE}$ contient le sous-graphe induit par les sommets qui ne sont pas incident aux arêtes obligatoires. Formellement, $\mathcal{G}_{MAYBE} = (\mathcal{V}_{MAYBE}, \mathcal{E}_{MAYBE})$, où $\mathcal{V}_{MAYBE}$ contient tous les sommets qui sont isolés dans $\mathcal{G}_{TRUE}$ et $\mathcal{E}_{MAYBE} = \mathcal{E} \cap (\mathcal{V}_{MAYBE} \times \mathcal{V}_{MAYBE})$.*

4 Filtrer une Contrainte *resource-forest*

4.1 Existence d'une Solution pour une Contrainte *resource-forest*

Le Théorème 1 fournit une condition nécessaire et suffisante quand à l'existence d'une solution pour la contrainte *resource-forest*. Les deux premières conditions assurent qu'il existe une partition du graphe en une forêt telle que chaque arbre contienne au moins un sommet ressource et la troisième assure que le nombre d'arbre dans la forêt soit dans le domaine de NTREE.

Theorem 1. *Il existe une solution pour la contrainte* resource-forest(NTREE, VER) *ssi les conditions suivantes sont vérifiées :*

(1) *$\mathcal{G}_{TRUE}$ ne contient aucun cycle.*

(2) *Toute composante connexe de $\mathcal{G}$ contient au moins un sommet ressource.*

(3) *$\mathcal{D}(\texttt{NTREE}) \cap [\texttt{MINTREE}, \texttt{MAXTREE}] \neq \emptyset$, où* MINTREE *est le nombre maximal de composantes connexes de $\mathcal{G}$ et* MAXTREE *est le nombre maximal de composantes connexes de $\mathcal{G}_{TRUE}$ qui contiennent au moins un sommet ressource.*

Démonstration. **Les conditions sont suffisantes :** Pour prouver que les trois conditions sont suffisantes, nous supposons qu'elles sont vérifiées et nous montrons que pour chaque valeur $k \in [\texttt{MINTREE}, \texttt{MAXTREE}]$, nous pouvons construire une forêt couvrante de $\mathcal{G}$ avec k arbres, chacun contenant un sommet ressource.
Cas 1 : $k =$ MAXTREE. Soit $\mathcal{T} = \{C_1, \cdots, C_p\}$ les composantes connexes de $\mathcal{G}_{TRUE}$. Par définition, exactement MAXTREE d'entre elles contiennent au moins un sommet ressource. Par la condition (2), chaque composante qui ne contient pas un sommet ressource est connectée par un chemin dans $\mathcal{G}$ à une composante en contenant au moins un. Pour obtenir une solution avec k arbres, nous fusionnons chaque composante qui ne contient pas de sommets ressources avec une qui en contient un, et nous renvoyons un arbre couvrant de chaque composante.
Cas 2 : $k <$ MAXTREE. Nous construisons d'abord une forêt de MAXTREE arbres comme dans le Cas 1 et ensuite nous fusionnons des arbres jusqu'à ce qu'il ne reste que k arbres : Tant qu'il y a trop d'arbres, nous en sélectionnons deux qui sont connectés par une arête e et nous les fusionnons en ajoutant e dans la forêt. Comme MINTREE est le nombre de composantes connexes dans $\mathcal{G}$, tant que $k >$ MINTREE nous avons la garantie de trouver deux arbres qui peuvent être fusionnés.

Les conditions sont nécessaires : Si $\mathcal{G}_{TRUE}$ contient un cycle, la solution ne peut pas être une forêt. Si $\mathcal{D}(\texttt{NTREE}) \cap [\texttt{MINTREE}, \texttt{MAXTREE}] = \emptyset$ alors nous avons $\max(\texttt{NTREE}) <$ MINTREE ou $\min(\texttt{NTREE}) >$ MAXTREE. Dans chaque cas, la contrainte est insatisfiable : Nous ne pouvons pas créer moins de MINTREE arbres car un arbre doit être connexe. Maintenant nous allons montrer que l'on ne peut pas créer plus de MAXTREE arbres. Remarquons d'abord qu'une composante connexe de $\mathcal{G}_{TRUE}$ ne peut pas être cassée, donc chaque composante ne peut pas appartenir à plus d'un arbre. De plus, les sommets d'une composante qui ne contient pas un sommet ressource doivent appartenir au même arbre que les sommets d'une composante qui contient un sommet ressource. En d'autres termes, une composante ne peut pas être dans un arbre de la forêt s'il ne contient pas un sommet ressource. □

4.2 *Consistance-Hybride* d'une Contrainte *resource-forest*

La Figure 1 montre un algorithme de filtrage pour la contrainte *resource-forest* atteignant la consistance--hybride. Premièrement, cet algorithme vérifie que la contrainte possède au moins une solution, en utilisant la

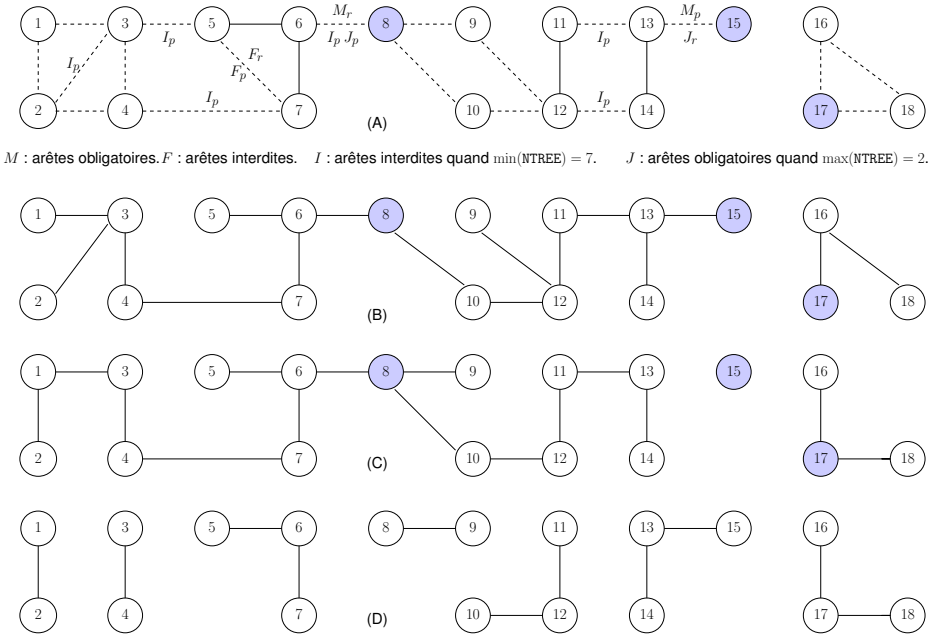

FIG. 1 – (A) Un graphe non-orienté avec 3 sommets ressources (en gris). (B) Une solution avec 2 arbres pour les contraintes *resource-forest* et *proper-forest*. (C) Une solution avec 3 arbres pour la contrainte *resource-forest*. (D) Une solution avec 7 arbres pour la contrainte *proper-forest*. Il faut remarquer que chaque sorte d'arête (M, F, I et J) est indicée par p dans le cas de la contrainte *proper-forest* et par r dans le cas de la contrainte *resource-forest*.

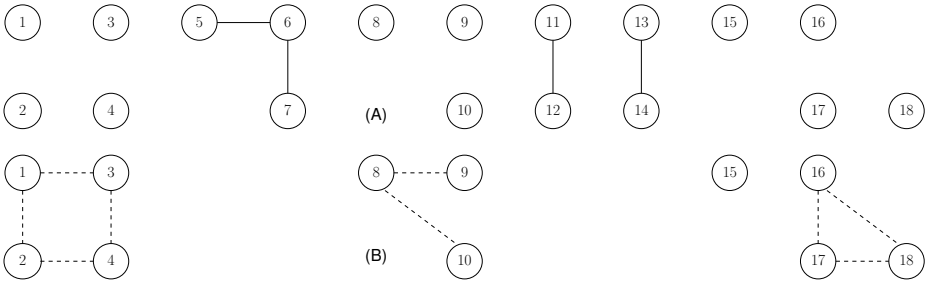

FIG. 2 – Les graphes (A) $\mathcal{G}_{TRUE}$ et (B) $\mathcal{G}_{MAYBE}$ associés avec le graphe de la Figure 1 en Partie (A).

caractérisation fournit par le Théorème 1. Les Lignes 3 à 7 effectuent un filtrage en accord avec le fait qu'une solution doit être une forêt (tout en ignorant la cardinalité de la forêt et la condition sur les ressources). A la Ligne 7 l'algorithme supprime toute arête en pointillé (u, v) où u et v sont connectés par une arête solide ; comme les arêtes solides doivent être dans une solution, cette arête en pointillé devrait créer un cycle (e.g., l'arête $(5, 7)$ de la Partie (A) de la Figure 1). Les Lignes 8 à 10 identifient les arêtes en pointillé qui doivent être dans une solution car leur retrait devrait séparer un ou plusieurs sommets des sommets ressources, et les bascule en arêtes solides (e.g., l'arête $(6, 8)$ de la Partie (A) de la Figure 1). Pour finir, la Ligne 11 met à jour le domaine de NTREE.

Les Lignes 12 à 17 sont exécutées uniquement lorsque le domaine de NTREE est fixé. Dans ce cas, le nombre d'arbres dans une solution est fixé et s'il est égal à MINTREE (comme définit dans le Théorème 1), tous les isthmes de $\mathcal{G}$ sont obligatoires et sont transformés en arêtes solides, sinon le nombre de composantes connexes du graphe, et par conséquent le nombre d'arbres de toute solution, est strictement supérieur à MINTREE (e.g., l'arête $(13, 15)$ de la Partie (A) de la Figure 1). D'un autre côté, si la valeur de NTREE est fixée à MAXTREE, alors toute composante connexe de $\mathcal{G}_{TRUE}$ contenant une ressource devrait appartenir à un arbre d'une solution, ainsi une arête en pointillé entre deux composantes de ce type ne doit pas être dans la forêt et est supprimée.

4.3 Correction

Dans la version complète de ce papier nous avons prouvé que l'algorithme atteint bien la consistance-hybride :

1. Aucune arête du graphe n'est supprimée si elle appartient à une solution et aucune valeur de NTREE n'est supprimée s'il existe une couverture de taille corres-

Algorithm 1 Algorithme de consistance-hybride pour la contrainte *resource-forest*.

```
1. Si la contrainte n'a pas de solutions
   (Théorème 1) alors
2.  sortir sur un échec ;
3. Calculer les composantes connexes (CCs)
   de G_TRUE ;
4. Pour chaque v ∈ V faire
5.  C(v) ← les CCs de G_TRUE contenant v ;
6. Pour chaque arête (u,v) ∈ E en pointillé faire
7.  Si C(u) = C(v) alors supprimer (u,v) du graphe ;
8. Pour chaque arête e en pointillé faire
9.  Si le retrait de e créé une CC de G sans
    sommets ressource alors
10.   Basculer e en arête solide ;
11. D(NTREE) ← D(NTREE) ∩ [MINTREE, MAXTREE] ;
12. Si D(NTREE) = {MINTREE} alors
13.  Pour chaque arête e en pointillé qui est
    un isthme de G faire
14.   Basculer e en arête solide ;
15. Si D(NTREE) = {MAXTREE} alors
16.  Pour chaque arête en pointillé e = (u,v)
    pour laquelle C(u) ≠ C(v) et C(u) et C(v)
    contiennent un sommet ressource faire
17.   Supprimer e du graphe ;
```

pondante.

2. Chaque arête restante dans $\mathcal{G}$ et chaque valeur de $\mathcal{D}(\texttt{NTREE})$ participe à au moins une solution, et chaque arête en pointillé restante est exclue d'au moins une solution.
3. Chaque arête en pointillé transformée en arête solide appartient à toute solution.

4.4 Complexité

Toutes les étapes de l'algorithme, à l'exception des Lignes 8 à 10, requièrent la détection des cycles, le calcul des composantes connexes et l'identification des isthmes. Toutes ces étapes peuvent être effectuées en temps linéaire [8, p.18]. Nous allons maintenant montrer que les Lignes 8 à 10 peuvent, elles aussi être vérifiées en temps linéaire. Clairement, une arête, dont le retrait créé une composante connexe sans sommet ressource, est un isthme dans $\mathcal{G}$. Mais tous les isthmes n'ont pas cette propriété. Nous créons donc un graphe réduit $\mathcal{H}$ en contractant chaque composante 2-connexe de $\mathcal{G}$ en un unique sommet. Le graphe $\mathcal{H}$ est un arbre dont les arêtes sont toutes des isthmes de $\mathcal{G}$. Nous dirons qu'un sommet de $\mathcal{H}$ est une ressource si un des sommets de $\mathcal{G}$ qui a été contracté était une ressource, i.e., si la composante 2-connexe qu'il représente possédait une ressource. Nous devons donc identifier quelles arêtes de $\mathcal{H}$ sont des arêtes dont le retrait pourrait créer une composante connexe de $\mathcal{H}$ sans sommets ressources. En d'autres mots, nous avons réduit notre problème au même problème mais sur des arbres. Ceci peut se produire pour une arête dont un des sommets adjacent est une racine d'un sous-arbre sans ressources. Ainsi, nous sélectionnons arbitrairement un sommet ressource de $\mathcal{H}$ et on effectue un DFS de $\mathcal{H}$ depuis ce sommet ressource. Chaque fois que l'on fait un retour arrière depuis un sommet v, on communique à son parent p si une ressource a été rencontrée dans le sous-arbre enraciné sur v. Si ce n'est pas le cas, alors l'arête (p, v) est transformée en une arête solide si ce n'était pas déjà le cas.
Ainsi, nous venons de montrer que :

Theorem 2. *L'algorithme de la Figure 1 filtre la contrainte* resource-forest *jusqu'à la consistance-hybride avec une complexité en temps de* $\mathcal{O}(m + n)$.

5 Filtrer une Contrainte *proper-forest*

5.1 Existence d'une Solution pour une Contrainte *proper-forest*

Le Théorème 3 fournit les conditions d'existence d'une solution pour une contrainte *proper-forest*. Les deux premières conditions assurent qu'il est possible de partitionner le graphe en une forêt propre et la troisième que le nombre d'arbres propres dans la forêt propre soit dans le domaine de NTREE.

Theorem 3. *Il existe une solution pour la contrainte* proper-forest(NTREE, VER) *ssi les conditions suivantes sont vérifiées :*

(1) *$\mathcal{G}$ n'a pas de sommets isolés,*

(2) *$\mathcal{G}_{TRUE}$ ne contient aucun cycle,*

(3) *$\mathcal{D}(\texttt{NTREE}) \cap [\texttt{MINTREE}, \texttt{MAXTREE}] \neq \emptyset$, où* MINTREE *est le nombre de composantes connexes de $\mathcal{G}$ et* MAXTREE *est le nombre de composantes connexes de $\mathcal{G}_{TRUE}$ de taille au moins deux plus la taille d'un couplage de cardinalité maximum de $\mathcal{G}_{MAYBE}$.*

Démonstration. **Les conditions sont suffisantes :** Pour prouver que les trois conditions sont suffisantes, nous supposons qu'elles sont vérifiées et nous montrons que pour chaque valeur $k \in [\texttt{MINTREE}, \texttt{MAXTREE}]$, nous pouvons construire une forêt couvrante de $\mathcal{G}$ avec k arbres propres. Nous commençons avec $k = \texttt{MAXTREE}$ et nous choisissons ensuite une valeur arbitraire $k \in [\texttt{MINTREE}, \texttt{MAXTREE}]$.

- Soit $\mathcal{T} = \{T_1, \cdots, T_p\}$ une forêt couvrante maximale de $\mathcal{G}_{TRUE}$, i.e., que chaque T_i est un sommet isolé dans $\mathcal{G}_{TRUE}$ ou une forêt couvrante d'une composante connexe de $\mathcal{G}_{TRUE}$. Il faut remarquer qu'un arbre T_i de taille un est aussi un sommet de $\mathcal{G}_{MAYBE}$.
- Pour construire une forêt couvrante de cardinalité MAXTREE, il faut calculer un couplage de cardinalité maximum $\mathcal{M}$ dans $\mathcal{G}_{MAYBE}$ et modifier $\mathcal{T}$ de la manière suivante :
  - Par définition de $\mathcal{G}_{MAYBE}$, chaque arête du couplage connecte deux singletons T_i et T_j de $\mathcal{T}$.

Alors, il faut les fusionner dans un arbre de taille deux.
  - Pour chaque sommet $u \in \mathcal{V}_{MAYBE}$, correspondant à un arbre T_i de taille un, qui n'est pas saturé dans le couplage $\mathcal{M}$, sélectionner un voisin v de u et inclure l'arête (u, v) dans la forêt couvrante. En d'autres termes, fusionner l'arbre T_i avec l'arbre T_v auquel v appartient. La Condition 1 garantie que ceci est possible. Il trivial de voir que la forêt obtenue est composée d'exactement MAXTREE arbres.
- Si $k <$ MAXTREE, fusionner des arbres propres jusqu'à avoir k arbres propres : Tant qu'il y a trop d'arbres propres, en sélectionner deux qui sont liés par une arête e de $\mathcal{G}_{MAYBE}$ et les fusionner en incluant e dans la forêt propre. Comme MINTREE est le nombre de composantes connexes dans $\mathcal{G}$, tant que $k >$ MINTREE nous sommes assurés de trouver deux arbres propres qui peuvent être fusionnés.

Les conditions sont nécessaires : Clairement, si $\mathcal{G}$ contient un sommet isolé v, alors v n'appartient pas à un sous-graphe de $\mathcal{G}$ qui est un arbre propre et si $\mathcal{G}_{TRUE}$ contient un cycle, la solution doit contenir un cycle donc cela ne pourra pas être une forêt propre. Finalement, si $\mathcal{D}(\texttt{NTREE}) \cap [\texttt{MINTREE}, \texttt{MAXTREE}] = \emptyset$ alors on a $\max(\texttt{NTREE}) <$ MINTREE ou $\min(\texttt{NTREE}) >$ MAXTREE. Dans chaque cas, la contrainte n'a pas de solution : Nous ne pouvons, en effet, avoir moins de MINTREE arbres propres car un arbre propre doit être connexe. Pour voir que l'on ne peut pas créer plus de MAXTREE arbres propres, il faut remarquer que le nombre d'arbres propres est au plus le nombre de composantes connexes de $\mathcal{G}_{TRUE}$ (nous ne pouvons pas casser une composante connexe de $\mathcal{G}_{TRUE}$) et qu'un sommet de $\mathcal{G}_{MAYBE}$ peut soit former un nouvel arbre propre avec un autre sommet de $\mathcal{G}_{MAYBE}$, soit être fusionné avec un arbre propre existant (et donc ne pas augmenter le nombre d'arbres propres). Clairement, un couplage de cardinalité maximum dans $\mathcal{G}_{MAYBE}$ permet de maximiser le nombre d'arbres propres construit. □

5.2 *Consistance-Hybride* d'une Contrainte *proper-forest*

La Figure 2 montre l'algorithme de filtrage associé à une contrainte *proper-forest* atteignant la consistance-hybride. Premièrement, il vérifie que la contrainte possède au moins une solution, en utilisant la caractérisation fournie par le Théorème 3. Les Lignes 3 à 7 et 18 à 20 filtrent en accord avec le fait qu'une solution doit être un arbre propre (en ignorant la taille de la forêt). A la Ligne 7 l'algorithme supprime toute arête (u, v) en pointillé (e.g., l'arête $(5, 7)$ de la Partie (A) de la Figure 1) où u et v sont connectés par un chemin d'arête solides ; comme les arêtes solides doivent être dans la solution, cette arête en pointillé va créer un cycle. Les Lignes 19-20 identifient les arêtes en pointillé qui doivent être dans la solution car leur retrait isolerait un sommet, et donc il faut les rendre solides (e.g., l'arête $(13, 15)$ de la Partie (A) de la Figure 1). Pour finir, la Ligne 8 met à jour le domaine de la variable NTREE.

Algorithm 2 Algorithme de consistance-hybride pour la contrainte *proper-forest*.

```
1. Si la contrainte n'a pas de solution (voir
   Theorem 3) alors
2.  Sortir sur un échec;
3. Calculer les composantes connexes (CCs)
   de G_TRUE.
4. Pour chaque v ∈ V faire
5.  C(v) ← les CCs of G_TRUE contenant v;
6. Pour chaque arête (u,v) ∈ E en pointillé faire
7.  Si C(u) = C(v) alors supprimer (u,v) du graphe;
8. D(NTREE) ← D(NTREE) ∩ [MINTREE, MAXTREE];
9. Si D(NTREE) = {MINTREE} alors
10.  Pour chaque arête (u,v) en pointillé qui est
     un isthme de G faire
11.    Basculer (u,v) en arête solide;
12. Si D(NTREE) = {MAXTREE} alors
13.  Pour chaque arête (u,v) en pointillé faire
14.    Supprimer (u,v) de G si une des conditions
       suivante est vérifiée :
15       a. |C(u)| > 1, |C(v)| > 1, et C(u) ≠ C(v).
16.      b. (u,v) ∈ E_MAYBE mais n'appartient
            à aucun couplage de cardinalité
            maximum dans G_MAYBE.
17.      c. |C(u)| > 1 et v est saturé dans tout
            couplage de cardinalité maximum
            de G_MAYBE.
18. Pour chaque arête (u,v) ∈ E en pointillé faire
19.  Si u est une feuille dans G alors
20.    Basculer (u,v) en arête solide;
```

Les Lignes 9 à 17 sont exécutées seulement lorsque le domaine de NTREE est fixé. Dans ce cas, le nombre d'arbres propres dans une solution est donc fixé et s'il est égal à MINTREE ou MAXTREE (ces valeurs sont définies dans le Théorème 3), on peut alors effectuer un filtrage supplémentaire : If $\mathcal{D}(\texttt{NTREE}) = \{\texttt{MINTREE}\}$ alors tous les isthmes de $\mathcal{G}$ sont obligatoires et sont transformés en arêtes solides, car sinon le nombre de composantes connexes du graphe, et donc le nombre d'arbres de toute solution, serait strictement supérieur à MINTREE. Dans l'exemple Partie (A) de la Figure 1, l'arête $(6, 8)$ (marquée par J_p) est obligatoire lorsque $\mathcal{D}(\texttt{NTREE}) = \{2\}$. Maintenant, si $\mathcal{D}(\texttt{NTREE}) = \{\texttt{MAXTREE}\}$ alors trois ensembles d'arêtes sont interdites et sont supprimées du graphe ; nous verrons qu'inclure l'une d'entre elles dans une solution va réduire le nombre d'arbres que l'on peut construire à une valeur inférieure à MAXTREE. Par exemple, si $\mathcal{D}(\texttt{NTREE}) = \{7\}$, les arêtes marquées par I_p dans la Partie (A) de la Figure 1 sont supprimées : $(11, 13)$ et $(12, 14)$ à la Ligne 15, $(2, 3)$ à la Ligne 16, et $(3, 5)$, $(4, 7)$ et $(8, 6)$ à la Ligne 17.

5.3 Correction

Pour prouver la correction de l'algorithme, nous allons montrer que :

1. Aucune arête du graphe n'est supprimée si elle appartient à une solution et aucune valeur de NTREE n'est supprimée s'il existe une couverture de taille correspondante (Lemma 1).
2. Chaque arête restante dans $\mathcal{G}$ et chaque valeur de $\mathcal{D}(\texttt{NTREE})$ participe à au moins une solution, et chaque arête en pointillé restante est exclue d'au moins une solution (Lemma 2).
3. Chaque arête en pointillé transformée en arête solide appartient à toute solution (Lemma 3).

Lemma 1. *L'algorithme de la Figure 2 ne supprime aucune arête du graphe ou aucune valeur de $\mathcal{D}(\texttt{NTREE})$ appartenant à une solution.*

Démonstration. Soit (u, v) une arête qui a été supprimée par l'algorithme. Supposons qu'il existe une solution S qui contienne (u, v). Si (u, v) a été supprimée à la Ligne 7, alors il existe un chemin d'arêtes solides depuis u jusqu'à v, et ses arêtes sont dans la solution. Mais alors la forêt S contient un cycle, c'est une contradiction. Donc il faut supposer que (u, v) a été supprimée aux Lignes 13 à 17. Dans ce cas, nous savons que le nombre d'arbres dans S est égal à MAXTREE (la nombre de composantes connexes de $\mathcal{G}_{TRUE}$ de cardinalités au moins deux plus la taille d'un couplage de cardinalité maximale dans $\mathcal{G}_{MAYBE}$). Nous verrons que s'il existe toujours une solution S' lorsque la contrainte est appliquée sur un graphe $\mathcal{G}'$, obtenu depuis $\mathcal{G}$ en rendant l'arête (u, v) solide, alors ceci viole les conditions du Théorème 3 car S' aurait plus que $\texttt{MAXTREE}'$ arbres (où $\texttt{MAXTREE}'$ est la valeur MAXTREE calculée sur le graphe $\mathcal{G}'$). Si (u, v) était supprimé à la Ligne 15, alors $\texttt{MAXTREE}' = \texttt{MAXTREE} - 1$ car u et v ne sont pas dans $\mathcal{G}_{MAYBE}$ et deux composantes connexes de $\mathcal{G}_{TRUE}$ ont été fusionnées. La solution $S' = S$ a MAXTREE (i.e., $>$ $\texttt{MAXTREE}'$) arbres. Si (u, v) était supprimé à la Ligne 16, alors u et v formerait une composante connexe de taille deux dans $\mathcal{G}'_{TRUE}$. Ceci augmente MAXTREE de un. D'un autre côté, la taille d'un couplage de cardinalité maximum dans $\mathcal{G}'_{MAYBE} = \mathcal{G}_{MAYBE} \setminus \{u, v\}$ est inférieure de deux à celle de $\mathcal{G}_{MAYBE}$, car sinon (u, v) appartiendrait à un couplage de cardinalité maximum. Ainsi, $\texttt{MAXTREE}' = \texttt{MAXTREE} - 1$ et $S' = S$ est une solution avec MAXTREE arbres. Au final, si (u, v) est supprimé à la Ligne 17, alors transformer (u, v) en une arête solide ajoute v dans la composante connexe contenant u dans $\mathcal{G}'_{TRUE}$. Comme v n'est pas dans $\mathcal{G}'_{MAYBE}$, la taille d'un couplage de cardinalité maximum dans $\mathcal{G}'_{MAYBE}$ est inférieure de un à celle de $\mathcal{G}_{MAYBE}$ (sinon $\mathcal{G}_{MAYBE}$ aurait un couplage de cardinalité maximum pour lequel v ne serait pas saturé, ce qui constituerait une contradiction). Supposons encore que $\texttt{MAXTREE}' = \texttt{MAXTREE} - 1$ et $S' = S$ soit une solution avec MAXTREE arbres. Si l'algorithme supprime une valeur utile de $\mathcal{D}(\texttt{NTREE})$, ceci contredit trivialement le Théorème 3. □

Lemma 2. *Après avoir appliqué l'Algorithme de la Figure 2, toute arête restante dans $\mathcal{G}$ et toute valeur restante dans $\mathcal{D}(\texttt{NTREE})$ participe à au moins une solution, et toute arête en pointillé est exclue d'au moins une solution.*

Démonstration. Nous avons déjà montré dans la preuve du Théorème 3 que chaque valeur dans $[\texttt{MINTREE}, \texttt{MAXTREE}]$ participe à une solution. Nous montrons maintenant que chaque arête (u, v) restante appartient à une forêt dans au moins une solution. Premièrement, nous construisons une solution S avec MAXTREE arbres comme précédemment. Si (u, v) appartient à la forêt, tout va bien. Sinon, soit $S' = S \cup (u, v)$. Si S' n'est pas une solution pour la contrainte, c'est soit à cause du fait que S' n'est pas une forêt ou soit parce que le nombre d'arbres dans S' n'est pas dans $\mathcal{D}(\texttt{NTREE})$. Si ce n'est pas une forêt, c'est parce que l'ajout de (u, v) a créé un cycle , mais dans ce cas (u, v) aurait été supprimée à la ligne 7. Ainsi, le nombre d'arbres, qui est $\texttt{MAXTREE} - 1$, n'est pas dans $\mathcal{D}(\texttt{NTREE})$. S'il existe une valeur dans $\mathcal{D}(\texttt{NTREE})$ qui est inférieure au nombre d'arbres de S', nous pouvons fusionner des arbres comme nous l'avons montré dans la preuve du Théorème 3 jusqu'à ce que l'on obtienne une solution. Sinon, on doit avoir $\mathcal{D}(\texttt{NTREE}) = \{\texttt{MAXTREE}\}$. Mais dans ce cas, (u, v) aurait du être supprimée à la Ligne 15.

Il reste à montrer que chaque arête en pointillé est exclue d'au moins une solution. Soit (u, v) une arête en pointillé. Nous avons prouvé ci-dessus qu'il existe une solution S qui utilise (u, v). Soit $S' = S \setminus \{(u, v)\}$. Si S' est une solution pour la contrainte alors tout va bien. Supposons, alors, que S' n'est pas une solution. Ceci peut se produire si S' contient un sommet isolé ou si elle est composée d'un trop grand nombre d'arbres.

Supposons que le retrait de (u, v) créé un arbre de taille un contenant le sommet u. Comme (u, v) n'a pas été transformée en arête solide à la Ligne 19, nous savons que u a un autre voisin $n_u \neq v$, avec lequel il est lié par une arête en pointillé. Si v n'était pas un arbre de taille un ou que le nombre d'arbres était strictement supérieur à MINTREE, nous aurions fusionné u avec un arbre voisin et obtenu ainsi une solution ne contenant pas l'arête (u, v).

Cependant, si le nombre d'arbres est exactement MINTREE et que à la fois u et v sont des sommets isolés, fusionner chacun d'entre eux dans un arbre existant nous laisserait $\texttt{MINTREE} - 1$ arbres. Heureusement, ce cas n'est pas possible. En effet, supposons qu'il se soit produit. nous savons que n_u appartient à un arbre de la solution qui contient au moins deux sommets et qui ne contient pas u ou v. Ainsi, la composante connexe de u (et v) dans $\mathcal{G}$ est présente dans au moins deux arbres de la solution, et la solution doit avoir

plus de MINTREE arbres.

Enfin, supposons qu'après le retrait de (u, v) il n'existe plus d'arbres de taille un, mais qu'il n'y est pas non plus un arbre de trop. Si l'on peut, nous fusionnons deux arbres en utilisant une arête e en pointillé autre que (u, v). Si ce n'est pas possible, alors c'est que le nombre d'arbres est égal au nombre de composantes connexes de $\mathcal{G} \setminus \{e\}$, qui est égal à MINTREE. Si (u, v) était un isthme, il aurait était transformé en une arête solide par la Ligne 11. Ainsi, il y a un cycle contenant (u, v). Comme nous ne sommes pas en mesure de fusionner deux arbres après la suppression de (u, v), c'est la même chose pour chaque arête en pointillé sur le cycle, donc les deux extrémités appartiennent au même arbre. Ceci implique que u et v sont dans le même arbre après la suppression de l'arête (u, v), ce qui signifie qu'il existe un cycle dans la forêt S, c'est une contradiction. □

Lemma 3. *Chaque arête que l'Algorithme de la Figure 2 bascule de pointillé à solide appartient à toute solution.*

Démonstration. Supposons qu'il existe une arête (u, v) transformée de pointillé à solide mais qui n'appartienne pas à une solution S. Si (u, v) est transformée en arête solide à la Ligne 19 de l'algorithme alors $\mathcal{G}$ a un sommet isolé, ainsi S ne peut pas être une forêt propre. Donc, la transformation a du être effectuée à la Ligne 11. Dans ce cas, nous savons que le domaine de NTREE est fixé à MINTREE, i.e., le nombre d'arbres dans S est égal au nombre de composantes connexes de $\mathcal{G}$. Mais alors tout isthme de $\mathcal{G}$, et par conséquent l'arête (u, v), doit appartenir à S ce qui constitue une contradiction. □

5.4 Complexité

La complexité de vérifier si une contrainte *proper-forest* a une solution est dominée par la complexité de calculer MAXTREE (tout le reste peut être effectué en temps linéaire). Pour trouver MAXTREE nous devons trouver la taille d'un couplage de cardinalité maximum dans $\mathcal{G}_{MAYBE}$ et la meilleure complexité théorique connue jusqu'ici est en $\mathcal{O}(m\sqrt{n})$ [10]. Les Lignes 3 à 9 sont effectuées en temps linéaire : Nous devons calculer les composantes connexes de $\mathcal{G}_{TRUE}$ et parcourir les arêtes en pointillé, ce qui coûte un temps constant pour chaque arête. Trouver tous les isthmes de $\mathcal{G}$ à la Ligne 10 et détecter les feuilles à la Ligne 19 occupe aussi un temps linéaire.

Jusqu'ici la complexité est dominée par le test de faisabilité qui coûte $\mathcal{O}(m\sqrt{n})$. Si le domaine de NTREE est fixé et contient la valeur MAXTREE, nous devons exécuter aussi les Lignes 13 à 17. La Ligne 15 est triviale. Pour la Ligne 16, nous devons déterminer quelles arêtes du graphe appartiennent à au moins un couplage de cardinalité maximum. Pour les graphes bipartites, ceci peut être fait en temps linéaire une fois un couplage de cardinalité maximum connu [12]. Cependant, nous devons effectuer cela sur des graphes arbitraires. Dans la version complète du papier, nous décrivons un algorithme qui effectue ce traitement en $\mathcal{O}(mn)$.

Enfin, pour la Ligne 19 nous avons un algorithme qui à partir d'un graphe et d'un couplage de cardinalité maximum, détecte les sommets du graphe qui sont saturés dans tout couplage de cardinalité maximum. Dans la version complète du papier, une solution en temps linéaire est décrite.

Theorem 4. *L'algorithme de la Figure 2 filtre la contrainte* proper-forest *jusqu'à la consistance-hybride avec une complexité en temps de $\mathcal{O}(mn)$ si $\mathcal{D}(\texttt{NTREE}) = \{\texttt{MAXTREE}\}$ et en $\mathcal{O}(m\sqrt{n})$ sinon.*

6 Résumé des Résultats sur les Contraintes de Couvertures par des Arbres

Cette section a pour but de mettre en avant les éléments communs ainsi que les différences entre les contraintes *tree* [2], *resource-forest* et *proper-forest*. toutes trois sont définies sur un graphe $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, orienté ou non, avec $|\mathcal{V}| = n$ et $|\mathcal{E}| = m$.

La Figure 3 résume les meilleurs résultats connus en terme de complexité temporelle pour vérifier l'existence d'une solution et pour atteindre la consistance-hybride pour chaque contrainte. La Figure 4 résume les principales propriétés de graphes utilisées pour déterminer les bornes sur le nombre d'arbres autorisés pour couvrir le graphe $\mathcal{G}$ ainsi que les conditions d'existence d'arbres bien formés en accord avec la définition de chaque contrainte. La dernière table indique que quatre propriétés de graphes usuelles définissent complètement ces contraintes : les composantes connexes (pour les graphes non-orientés), les composantes fortement connexes (pour les graphes orientés), les couplages de cardinalités maximum, l'existence de cycles. Pour chaque contraintes, des conditions nécessaires et des règles de filtrage ont été déduites avec des algorithmes connus (e.g., dfs, couplage de cardinalité maximum, détection des composantes connexes, etc.) ainsi que de nouveaux algorithmes (e.g., l'identification des sommets qui sont saturés dans tous les couplages de cardinalité maximum). Il faut remarquer que les bornes inférieure et supérieure MINTREE et MAXTREE pour la contrainte *proper-forest* correspondent exactement aux bornes inférieure et supérieure sur le nombre de composantes connexes qui ont été déterminées dans [3].

Notation 1. *Pour un graphe, H, le nombre de composantes connexes (CCs) dans H est donné par $CC(H)$, le couplage de cardinalité maximum de H est donné par $\mu(H)$, le nombre de composantes fortement connexes puits (SCCs) dans H et donné par $|SCC_{sink}(H)|$ et le nombre de racines potentielles dans H est donné par $|R_{potentiel}(H)|$.*

Pattern du Graphe	Anti-arborescences	Arbres	
	tree	*proper-forest*	*resource-forest*
faisabilité	$\mathcal{O}(n+m)$	$\mathcal{O}(m\sqrt{n})$	$\mathcal{O}(n+m)$
consistance-hybride	$\mathcal{O}(mn)$	$\mathcal{O}(mn)$ [au pire], $\mathcal{O}(m\sqrt{n})$ [typique]	$\mathcal{O}(n+m)$

FIG. 3 – Meilleures complexités connues pour les contraintes de couvertures par des arbres.

Pattern du Graphe	Anti-arborescences	Arbres	
	tree	*proper-forest*	*resource-forest*
MINTREE	$\lvert SCC_{sink}(G)\rvert$	$\lvert CC(\mathcal{G})\rvert$	$\lvert CC(\mathcal{G})\rvert$
MAXTREE	$\lvert R_{potentiel}(\mathcal{G})\rvert$	$\lvert CC(\mathcal{G}_{TRUE})\rvert +$ $\mu(\mathcal{G}_{MAYBE})$	$\lvert CC(\mathcal{G}_{TRUE})$ avec au moins une *resource*$\rvert$
Arbres bien formés	au moins une racine potentielle dans chaque SCC de $\mathcal{G}$	pas de cycle dans $\mathcal{G}_{TRUE}$ pas de sommets isolés dans $\mathcal{G}$	pas de cycle dans $\mathcal{G}_{TRUE}$ un sommet *ressource* dans chaque $CC(\mathcal{G}_{TRUE})$
Nombre d'arbres compatibles	$\mathcal{D}(\texttt{NTREE}) \cap [\texttt{MINTREE}, \texttt{MAXTREE}] \neq \emptyset$		

FIG. 4 – Propriétés caractérisant les solutions des trois contraintes de couverture par des arbres.

7 Conclusion

Dans ce papier nous avons proposé les caractérisations complètes de deux contraintes de partitionnement par des arbres *resource-forest* et *proper-forest*. Nous avons aussi fournit les algorithmes de filtrage polynomiaux permettant d'atteindre la consistance-hybride pour chacune d'entre elle. Pour finir, nous nous sommes intéressé à synthétiser les propriétés issues de la théorie des graphes qui ont permis d'aboutir à une caractérisation complète des contraintes de partitionnement de graphe par des arbres tant pour les graphes orientés que pour les graphes non-orientés.

Références

[1] N. Beldiceanu and E. Contejean. Introducing global constraint in CHIP. *Mathl. Comput. Modelling*, 20(12) :97–123, 1994.

[2] N. Beldiceanu, P. Flener, and X. Lorca. The *tree* Constraint. In *CP-AI-OR'05*, volume 3524 of *LNCS*, pages 64–78. Springer-Verlag, May 2005.

[3] N. Beldiceanu, T. Petit, and G. Rochart. Bounds of Graph Characteristics. In P. van Beek, editor, *CP 2005*, volume 3709 of *LNCS*, pages 742–746. Springer-Verlag, 2005.

[4] C. Berge. *Graphes*. Dunod, New York, 2nd edition, 1985. In French.

[5] C. Bessière, E. Hebrard, B. Hnich, Z. Kızıltan, and T. Walsh. The *range* and *roots* Constraints : Specifying Counting and Occurrence Problems. In *IJCAI-05*, pages 60–65, 2005.

[6] A. Cayley. A theorem on trees. *Quart. J. Math.*, 23 :376–378, 1889.

[7] M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. The constraint logic programming language CHIP. In *Int. Conf. on Fifth Generation Computer Systems (FGCS'88)*, pages 693–702, Tokyo, Japan, 1988.

[8] M. Gondran and M. Minoux. *Graphes et algorithmes*. Eyrolles, Paris, 2nd edition, 1985. In French.

[9] J.-L. Laurière. A language and a program for stating and solving combinatorial problems. *Artificial Intelligence*, 10 :29–127, 1978.

[10] S. Micali and V. V. Vazirani. An $\mathcal{O}(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matching in general graphs. In *FOCS 1980*, pages 17–27, New York, 1980. IEEE.

[11] J.-F. Puget. A C++ Implementation of CLP. In *Second Singapore International Conference on Intelligent Systems (SPICIS)*, pages 256–261, Singapore, November 1994.

[12] J.-C. Régin. A filtering algorithm for constraints of difference in CSP. In *AAAI-94*, pages 362–367, 1994.

[13] M. Sellmann. *Reduction techniques in Constraint Programming and Combinatorial Optimization*. PhD thesis, University of Paderborn, 2002.

[14] M. Sellmann. Cost-based filtering for shortest path constraints. In *CP 2003*, volume 2833 of *LNCS*, pages 694–708. Springer-Verlag, 2003.