HAL
archives-ouvertes.fr

Partitionnement de graphes par des arbres sous contraintes de degré

Nicolas Beldiceanu, Pierre Flener, Xavier Lorca

▶ **To cite this version:**

Nicolas Beldiceanu, Pierre Flener, Xavier Lorca. Partitionnement de graphes par des arbres sous contraintes de degré. Deuxièmes Journées Francophones de Programmation par Contraintes (JFPC06), 2006, Nîmes - Ecole des Mines d'Alès / France, 2006. <inria-00085802>

HAL Id: inria-00085802
https://hal.inria.fr/inria-00085802

Submitted on 14 Jul 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Partitionnement de graphes par des arbres sous contraintes de degré

Nicolas Beldiceanu [1] **Pierre Flener** [2] **Xavier Lorca** [1]

[1] École des Mines de Nantes, LINA FRE CNRS 2729,
FR – 44307 Nantes Cedex 3, France

[2] Department of Information Technology, Uppsala University,
Box 337, SE – 751 05 Uppsala, Suède

{nicolas.beldiceanu,xavier.lorca}@emn.fr pierref@it.uu.se

Résumé

Nous présentons une contrainte de partitionnement de graphe par des arbres sous contraintes de degré. Étant donné un graphe orienté $\mathcal{G}$, la contrainte spécifie que toute solution est formée par N arbres, tels que le nombre de fils d'un sommet quelconque de $\mathcal{G}$ appartienne à un ensemble donné de valeurs. Il existe pour la contrainte d'arbre pure, introduite dans [2], une caractérisation complète polynomialement évaluable. Cependant, la prise en compte du nombre de fils de chaque sommet rend le problème d'existence d'une solution NP-complet. Nous montrons donc comment intégrer une relaxation polynomiale de la contrainte de degré, prenant en compte les différents aspects de la contrainte d'arbre pure. Enfin, nous présentons une première série de résultats partiels pratiques sur les problèmes (NP-complets) d'existence d'un chemin Hamiltonien et d'un arbre binaire couvrant dans le cas de graphes orientés quelconques.

1 Introduction

Les contraintes décrivant des propriétés de graphes furent très tôt considérées par la communauté de programmation par contraintes. En effet, les problèmes de circuit Hamiltonien ou d'existence d'un arbre recouvrant furent initialement abordés dans ALICE [6]. Par la suite, suivirent les contraintes de *cycle* [1] et de *chemin* [3, 10, 11]. Curieusement, jusqu'à peu, aucune contrainte de partitionnement par des arbres n'a réellement été étudiée par la communauté alors qu'elle a fait l'objet d'une attention particulière en recherche opérationnelle [9]. Soulignons qu'une contrainte d'arbre constitue un premier pas vers une contrainte de partitionnement par des chemins.

L'objectif de cet article est de proposer une contrainte de partitionnement de graphes par des arbres dérivée de [2], qui prenne en compte une contrainte sur le nombre de prédécesseurs de chaque sommet (i.e., son degré négatif) du graphe à couvrir. Ainsi, elle rend possible, comme nous le verrons dans la suite, la recherche d'une couverture par des arbres binaires ou même par des chemins.

Par abus de langage nous appellerons *arbre* une anti-arborescence dans cet article (i.e., que l'on chemine des feuilles vers la racine). Par conséquent, dans la suite, un arbre sera défini comme un graphe orienté, connexe, et sans circuits impliquant plus d'un sommet, tel que chaque sommet possède un unique successeur,[1] et tel qu'il existe un chemin élémentaire de chaque sommet vers la racine.

La contrainte d'arbre est donnée sous la forme $tree(\texttt{NTREE}, \texttt{VER})$, où NTREE est une variable entière[2] et VER est la collection des n sommets $\texttt{VER}[1], \ldots, \texttt{VER}[n]$ du graphe donné. À chaque sommet $v_i = \texttt{VER}[i]$ est associée une liste d'attributs définis de la manière suivante :

- idx est un entier compris entre 1 et n. Il représente l'index du sommet v_i dans la collection.
- F est une variable entière définie sur $[1, n]$. Elle représente l'unique successeur (ou père) du sommet v_i dans une couverture.
- D est une variable entière définie sur $[0, n-1]$. Elle représente le nombre de prédécesseurs du sommet v_i.

[1] Le sommet racine de l'arbre porte une boucle sur lui-même.

[2] Une variable entière v est définie sur un domaine de valeurs potentielles $dom(v)$, dont $min(v)$ et $max(v)$ sont les valeurs minimale et maximale.

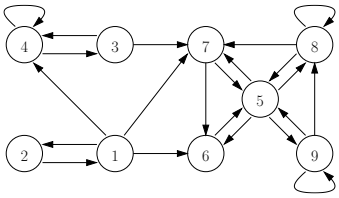


FIG. 1 – Graphe associé $\mathcal{G}$ d'une contrainte d'arbre

Lorsque l'on parle de contraintes globales, il est très souvent plus facile de raisonner sur un graphe (orienté ou pas) modélisant la contrainte, que de raisonner directement sur les variables et leurs domaines. Ainsi, la contrainte d'arbre va être modélisée par un graphe orienté $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ dont les sommets représentent les éléments de VER et les arcs représentent une relation de successeur entre eux. On définit $\mathcal{G}$ de la manière suivante :

Définition 1. *Le* graphe orienté associé $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *d'une contrainte* $tree(\texttt{NTREE}, \texttt{VER})$ *est défini par* $\mathcal{V} = \{v_i \mid i \in [1, n]\}$ *et* $\mathcal{E} = \{(v_i, v_j) \mid j \in dom(\texttt{VER}[i].\texttt{F})\}$. *De plus,* $m = |\mathcal{E}|$ *est le nombre d'arcs du graphe* $\mathcal{G}$.

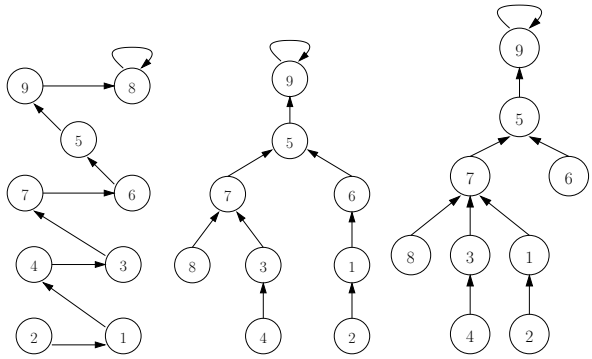
Une contrainte d'arbre spécifie que le graphe orienté $\mathcal{G}$ qui lui est associé doit être une forêt de NTREE arbres respectant la contrainte sur le nombre de prédécesseurs autorisés pour chaque sommet de $\mathcal{G}$. De manière formelle, on peut définir une solution respectant cette contrainte par :

Définition 2. *Un graphe orienté* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ satisfait *une contrainte d'arbre* $tree(\texttt{NTREE}, \texttt{VER})$ *si et seulement si :*

1. *Pour chaque* $i \in [1, n]$ *on a* $\texttt{VER}[i].\texttt{idx} = i$.
2. $\mathcal{G}$ *est composé de* NTREE *composantes connexes.*
3. *Chaque composante connexe de* $\mathcal{G}$ *ne contient pas de circuits, excepté la boucle sur la racine.*
4. *Pour chaque* $i \in [1, n]$, *le nombre de sommets* v_j *précédant* v_i *dans* $\mathcal{G}$ *est égal à* $\texttt{VER}[i].\texttt{D}$.

Exemple 1. *La figure 1 donne une représentation graphique d'une instance de la contrainte d'arbre. La table 1 décrit ce même graphe en termes de domaines de variables successeur, ainsi que pour chaque sommet le domaine de sa variable de degré. La figure 2 fournit trois types de solutions possibles pour le graphe de la figure 1 : la solution donnée en figure 2(a) fournit une couverture par un chemin Hamiltonien, la solution donnée en figure 2(b) donne une couverture par un arbre binaire (voir l'instance décrite par la table 1), et la solution donnée en figure 2(c) fournit une couverture par un arbre non contraint sur le degré de ces sommets.*

Avant de poursuivre, notons que malgré l'existence d'une caractérisation complète polynomialement évaluable pour la contrainte d'arbre pure [2], la prise en compte d'une restriction sur le degré négatif de chaque sommet rend NP-complet le problème de l'existence d'une couverture de $\mathcal{G}$ (voir [4, p.206]). Pour s'en convaincre, il suffit de considérer une contrainte $tree(\texttt{NTREE}, \texttt{VER})$ telle que :



(a) Chemin Hamiltonien (b) Arbre binaire (c) Arbre "quelconque"

FIG. 2 – Différents types de couvertures, dérivées du graphe $\mathcal{G}$ de la figure 1, suivant le type de contraintes sur le degré de chaque sommet

- NTREE $= 1$.
- Pour le sommet d'index s on a $\texttt{VER}[s].\texttt{D} = 0$.
- Pour tout sommet d'index $i \neq s$ on a $\texttt{VER}[i].\texttt{D} = 1$.
- Pour le sommet d'index d on a $dom(\texttt{VER}[d].\texttt{F}) = \{d\}$.

Alors, la contrainte d'arbre impose la recherche d'un chemin élémentaire du sommet v_s au sommet v_d, passant par tous les sommets de $\mathcal{G}$. Elle correspond donc au problème de chemin Hamiltonien [4, p.199] dans $\mathcal{G}$.

La suite de l'article est organisée de la manière suivante. La section 2 rappelle une condition nécessaire et suffisante à l'existence d'une solution pour la contrainte d'arbre pure présentée dans [2], puis introduit un algorithme d'arc-consistance pour cette contrainte. La section 3 présente un algorithme de filtrage permettant de maintenir la contrainte de degré sur chacun des sommets du graphe $\mathcal{G}$. Pour finir, la section 4 présente deux évaluations de la contrainte, la première sur le problème d'existence d'un chemin Hamiltonien dans un graphe orienté, et la seconde sur le problème d'existence d'un arbre binaire couvrant.

2 La contrainte de partitionnement par des arbres

Cette section rappelle la caractérisation complète de la contrainte de partitionnement de graphe par des arbres introduite dans [2]. Tout d'abord, dans la sous-section 2.1 sont définies les bornes minimum et maximum sur le nombre d'arbres[3] possibles pour couvrir le graphe donné en paramètre. La sous-section 2.2 rappelle une condition nécessaire et suffisante à la faisabilité d'une contrainte d'arbre. Finalement, la sous-section 2.3 montre comment utiliser cette condition afin d'assurer un filtrage complet, i.e., atteignant l'arc-consistance.

[3] On accepte qu'un sommet isolé portant une boucle sur lui-même soit considéré comme un arbre.

i	$dom(\texttt{VER}[i].\texttt{F})$	$dom(\texttt{VER}[i].\texttt{D})$		
		Partie (a)	Partie (b)	Partie (c)
1	$\{2,4,6,7\}$	$\{1\}$	$[0,2]$	$[0,8]$
2	$\{1\}$	$\{0\}$	$[0,2]$	$[0,8]$
3	$\{4,7\}$	$\{1\}$	$[0,2]$	$[0,8]$
5	$\{6,7,8,9\}$	$\{1\}$	$[0,2]$	$[0,8]$
4	$\{3,4\}$	$\{1\}$	$[0,2]$	$[0,8]$
6	$\{5\}$	$\{1\}$	$[0,2]$	$[0,8]$
7	$\{5,6\}$	$\{1\}$	$[0,2]$	$[0,8]$
8	$\{5,7,8\}$	$\{1\}$	$[0,2]$	$[0,8]$
9	$\{5,8,9\}$	$\{1\}$	$[0,2]$	$[0,8]$

TAB. 1 – La première colonne donne le nom de chaque variable père, la seconde décrit son domaine (i.e., dans la figure 1 l'ensemble des successeurs du sommet associé à la variable père). Les colonnes 3, 4 et 5 décrivent respectivement le domaine des variables de degré associées à chaque variable père dans la colonne Partie (a) pour une couverture par un chemin Hamiltonien (figure 2(a)), dans la colonne Partie (b) pour une couverture par des arbres binaires (figure 2(b)) et dans la colonne Partie (c) pour une couverture par des arbres quelconques (figure 2(c)).

Nous introduisons tout d'abord certains concepts associés au graphe $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ d'une contrainte d'arbre :

- À chaque instance d'une contrainte $tree(\texttt{NTREE}, \texttt{VER})$ va être associé un *graphe réduit* $\mathcal{G}_r$. Ce graphe est dérivé de $\mathcal{G}$ en associant à chaque composante fortement connexe (cfc) de $\mathcal{G}$ un sommet de $\mathcal{G}_r$ et à un ensemble d'arcs reliant deux cfc de $\mathcal{G}$ un arc unique de $\mathcal{G}_r$.
- Une cfc de $\mathcal{G}$ qui correspond à un sommet puits de $\mathcal{G}_r$ est appelée *composante puits*.
- Un sommet $v \in \mathcal{V}$ tel que $(v, v) \in \mathcal{E}$ est appelé *racine potentielle*. Par extension, un arc $(v, v) \in \mathcal{E}$ est appelé *boucle*. Une cfc de $\mathcal{G}$ contenant au moins une racine potentielle est appelée *composante enracinée*.
- Un sommet $u \in \mathcal{V}$ est appelé *porte* de la cfc le contenant ssi il existe un arc $(u, v) \in \mathcal{E}$ tel que v n'appartient pas à la cfc contenant u. Par extension, un arc $(u, v) \in \mathcal{E}$ tel que u et v n'appartiennent pas à la même cfc est appelé *arc connectant*.
- *Forcer* un arc $(u, v) \in \mathcal{E}$ correspond à supprimer de $\mathcal{E}$ tous les arcs (u, w) tels que $v \neq w$.

2.1 Bornes sur le nombre d'arbres

Nous introduisons maintenant des bornes minimum (MINTREE) et maximum (MAXTREE) sur le nombre d'arbres possibles pour couvrir un graphe $\mathcal{G}$ associé à une contrainte d'arbre. Les preuves sont disponibles dans [2].

Proposition 1. *Le nombre minimum d'arbres (*MINTREE*) pour partitionner un graphe orienté $\mathcal{G}$ associé à une contrainte d'arbre est donné par le nombre de sommets puits dans le graphe réduit $\mathcal{G}_r$.*

Algorithm 1 Faisabilité pour $tree(\texttt{NTREE}, \texttt{VER})$

```
1. Calculer les cfc de G ;
2. Soit C_p les cfc puits de G ;
3. Pour chaque C ∈ C_p faire
4.   Si ∄u ∈ C tel que u est une racine
     potentielle alors Retourner faux ;
5. Si dom(NTREE) ∩ [MINTREE, MAXTREE] = ∅ alors
6.   Retourner faux ;
```

Proposition 2. *Le nombre maximum d'arbres (*MAXTREE*) pour partitionner un graphe orienté $\mathcal{G}$ associé à une contrainte d'arbre est donné par le nombre de racines potentielles dans $\mathcal{G}$.*

2.2 Condition nécessaire et suffisante

Le théorème suivant caractérise les conditions d'existence d'une solution pour une contrainte d'arbre. La première condition assure que l'on peut effectivement créer une forêt couvrante alors que la seconde condition vérifie qu'il existe au moins une valeur compatible k dans le domaine de NTREE permettant de construire k arbres.

Théorème 1. *Il existe une solution pour une contrainte d'arbre $tree(\texttt{NTREE}, \texttt{VER})$ ssi les deux conditions suivantes sont vérifiées :*

1. *Toute composante puits de $\mathcal{G}$ est une composante enracinée de $\mathcal{G}$.*
2. $dom(\texttt{NTREE}) \cap [\texttt{MINTREE}, \texttt{MAXTREE}] \neq \emptyset$.

Démonstration. Voir la preuve dans [2, p.69]. □

Complexité : L'algorithme 1 vérifie que les conditions 1 et 2 du Théorème 1 sont vérifiées, i.e., qu'il existe au moins une couverture de $\mathcal{G}$ par NTREE arbres. Les lignes 1 et 2 de l'algorithme calculent les cfc puits de $\mathcal{G}$ en temps $O(n+m)$ (algorithme de Tarjan [12]). Les lignes 3 et 4 vérifient, en temps $O(n)$, si chacune de ces cfc puits contient au moins une racine potentielle. Pour finir, les lignes 5 et 6 vérifient s'il existe un nombre d'arbres compatible dans le domaine de NTREE ; la complexité de cette étape est dominée par la complexité du calcul de la borne MINTREE. D'où le résultat :

Théorème 2. *L'existence d'une solution est vérifiable en temps $O(n+m)$ pour une contrainte $tree(\texttt{NTREE}, \texttt{VER})$.*

2.3 Filtrage

Avant d'introduire l'algorithme de filtrage, nous rappelons la notion de point fort d'articulation [5, p.175]. En effet, cette notion permettra d'assurer l'arc-consistance des domaines des variables associées au graphe $\mathcal{G}$.

Algorithm 2 Filtrage pour $tree(\texttt{NTREE}, \texttt{VER})$

```
1.  Si tree(NTREE, VER) n'a pas de solution alors
2.    Retourner un échec ;
3.  Calculer les pfa de G et mémoriser dans C_p
    les cfc créés par le retrait de p dans G ;
4.  Pour chaque pfa p ∈ V faire
5.    Pour chaque arc (p, v) ∈ E faire
6.      Soit C_p(v) la cfc contenant v ;
7.      Si C_p(v) ∈ Δ^p_in alors Supprimer (p, v) de G ;
8.  dom(NTREE) ← dom(NTREE) ∩ [MINTREE, MAXTREE] ;
9.  Pour chaque cfc C de G faire
10.   Si C est une cfc puits et il existe
      une unique racine potentielle alors
11.     Forcer la boucle ;
12.   Si il existe un unique sommet porte
      et aucune racine potentielle alors
13.     Interdire tout arc non connectant de C ;
14.   Si dom(NTREE) = {MAXTREE} alors
15.     Forcer toutes les boucles de C ;
16.   Si dom(NTREE) = {MINTREE} et C n'est pas
      une cfc puits de G alors
17.     Interdire toutes les boucles de C ;
```

Définition 3. *Un* point fort d'articulation *(pfa) p d'un graphe orienté $\mathcal{G}$ est un sommet tel que le nombre de cfc de $\mathcal{G} \setminus \{p\}$ est supérieur au nombre de cfc de $\mathcal{G}$.*

On peut ainsi définir, pour chaque cfc $\mathcal{C}$ d'un graphe $\mathcal{G}$ associé à une contrainte d'arbre, une partition entre les cfc créées par le retrait d'un pfa p :

- $\Delta^p_{out} = \{\mathcal{C}^1_{out}, \ldots, \mathcal{C}^k_{out}\}$ est l'ensemble des cfc créées par le retrait de p dans $\mathcal{C}$ depuis lesquelles on peut atteindre, sans passer par p, soit une racine potentielle, soit un sommet porte.
- $\Delta^p_{in} = \{\mathcal{C}^1_{in}, \ldots, \mathcal{C}^k_{in}\}$ est l'ensemble des cfc créées par le retrait de p depuis lesquelles on ne peut pas atteindre une racine potentielle ou un sommet porte sans passer par p.

L'algorithme 2 effectue un filtrage arc-consistant associé à la contrainte d'arbre. Les lignes 1 et 2 vérifient si la contrainte possède au moins une solution. Pour cela, l'algorithme 1 est utilisé. Ensuite, les lignes 3 à 7 suppriment les arcs sortant de pfa qui sont incompatibles avec une couverture arborescente. La ligne 8 met à jour le domaine de NTREE avec les bornes MINTREE et MAXTREE. Puis, pour chaque cfc de $\mathcal{G}$, les lignes 10 à 13 assurent que chaque sommet d'une cfc pourra atteindre une racine potentielle. Pour terminer, les lignes 14 à 17 filtrent les domaines des variables père par rapport aux bornes de NTREE.

Correction et Complétude : Pour prouver que l'algorithme 2 atteint bien l'arc-consistance, nous montrons que :

1. Aucune valeur des domaines des variables F n'est supprimée si elle appartient à au moins une solution à la contrainte $tree(\texttt{NTREE}, \texttt{VER})$ (Lemme 1).
2. Tout arc restant dans $\mathcal{G}$ et toute valeur présente dans $dom(\texttt{NTREE})$ après filtrage participent à au moins une solution à la contrainte $tree(\texttt{NTREE}, \texttt{VER})$ (Lemme 2).

Lemme 1. *L'algorithme 2 ne retire aucun arc du graphe $\mathcal{G}$ pouvant appartenir à une solution satisfaisant la contrainte* $tree(\texttt{NTREE}, \texttt{VER})$.

Démonstration. Supposons qu'il existe une solution $\mathcal{S}$ à la contrainte $tree(\texttt{NTREE}, \texttt{VER})$ contenant l'arc (u, v) telle que ce dernier soit supprimé par l'algorithme 2. Si (u, v) est supprimé par la ligne 7 alors c'est que le sommet u est un pfa dans $\mathcal{G}$ et le sommet v appartient à Δ^u_{in}, i.e., v ne peut pas atteindre une racine potentielle sans passer par u. Mais alors il existe un cycle contenant u et v dans $\mathcal{S}$, ce qui est impossible. Maintenant, supposons que u soit l'unique racine potentielle d'une cfc puits de $\mathcal{G}$. S'il existe $v \neq u$ tel que (u, v) appartient à $\mathcal{S}$ alors il existe un circuit dans cette cfc, ce qui est impossible, donc le filtrage de la ligne 11 est correct. De la même manière, en considérant u comme l'unique porte d'une cfc de $\mathcal{G}$, on déduit que le filtrage de la ligne 13 est correct. Dans le cas où il existe MAXTREE arbres dans $\mathcal{S}$ et u est une racine potentielle de $\mathcal{G}$, il existe une contradiction si $v \neq u$ car MAXTREE ne peut pas être atteint. Pour finir, s'il existe MINTREE arbres dans $\mathcal{S}$ et $u = v$ n'appartient pas à une cfc puits de $\mathcal{G}$ alors il existe une contradiction car MINTREE ne peut pas être atteint. □

Lemme 2. *Après application de l'algorithme 2, toutes les valeurs encore présentes dans les domaines participent à au moins une solution satisfaisant la contrainte* $tree(\texttt{NTREE}, \texttt{VER})$.

Démonstration. On se donne le graphe orienté $\mathcal{G}$ associé à une contrainte $tree(\texttt{NTREE}, \texttt{VER})$ tel que tout arc de $\mathcal{G}$ est compatible avec l'algorithme de filtrage, ainsi qu'un arc $(u, v) \in \mathcal{E}$ et un entier $k \in [\texttt{MINTREE} + 1, \texttt{MAXTREE} - 1]$. Considérons maintenant le graphe $\mathcal{G}_{(u,v)} = \mathcal{G} \setminus \{(u, w) \mid w \neq v\}$ et supposons que la seconde condition du théorème 1 soit violée. Alors, par définition, forcer l'arc (u, v) a créé une cfc puits dans $\mathcal{G}_{(u,v)}$ ne contenant pas de racine potentielle. On distingue trois cas :

1. Le nombre de cfc de $\mathcal{G}_{(u,v)}$ est supérieur au nombre de cfc de $\mathcal{G}$. Alors u était un pfa de $\mathcal{G}$ et (u, v) aurait du être supprimé par la ligne 7 de l'algorithme 2, donc il y a contradiction avec l'hypothèse que $\mathcal{G}$ est filtré.
2. Le nombre de cfc de $\mathcal{G}_{(u,v)}$ est égal au nombre de cfc de $\mathcal{G}$ et la cfc de $\mathcal{G}$ contenant u est une cfc puits. Alors u était l'unique racine potentielle de la cfc et aurait du être supprimé par la ligne 11 de l'algorithme 2, donc il y a contradiction avec l'hypothèse que $\mathcal{G}$ est filtré.
3. Le nombre de cfc de $\mathcal{G}_{(u,v)}$ est égal au nombre de cfc de $\mathcal{G}$ et la cfc de $\mathcal{G}$ contenant u n'est pas une cfc puits. Alors u était l'unique porte de la cfc et il aurait du être supprimé par la ligne 13 de l'algorithme 2, donc il y a contradiction avec l'hypothèse que $\mathcal{G}$ est filtré.

Vérifions maintenant la complétude aux bornes. Si $k =$ MAXTREE, alors si u est l'unique racine potentielle d'une cfc puits de $\mathcal{G}$, alors l'arc (u, v), avec $u \neq v$, ne peut pas être dans $\mathcal{G}$ car le filtrage l'aurait supprimé. D'autre part, si $k =$ MINTREE, alors si u n'appartient pas à une cfc puits de $\mathcal{G}$ (u, v), alors l'arc (u, v) ne peut pas être une boucle car le filtrage l'aurait supprimé. □

Complexité : Deux types de filtrage peuvent être distingués. Tout d'abord, il y a un filtrage purement lié à la *structure* du graphe $\mathcal{G}$ associé à la contrainte d'arbre. Les lignes 3 à 7 filtrent les domaines des variables associées à $\mathcal{G}$ relativement aux pfa de $\mathcal{G}$. Cette détection est faite en temps $O(n \cdot m)$. Ensuite, pour chaque cfc, la ligne 10 vérifie si la cfc courante est une cfc puits de $\mathcal{G}$ et si elle contient une unique racine potentielle (par énumération sur les sommets de la cfc en temps $O(n)$). La ligne 12 filtre les domaines des variables associées à $\mathcal{G}$ relativement aux cfc ne contenant qu'une unique porte et aucune racine potentielle. Ce filtrage est effectué en temps $O(n)$.

D'autre part, le deuxième type de filtrage est lié à la contrainte sur le nombre d'arbres autorisés (NTREE) pour couvrir $\mathcal{G}$. La ligne 8 normalise le domaine de NTREE relativement aux bornes minimum (MINTREE) et maximum (MAXTREE) sur le nombre d'arbres possibles pour couvrir $\mathcal{G}$. La ligne 14 force toutes les boucles de la cfc courante lorsque la borne MAXTREE est atteinte par NTREE (par énumération des boucles de la cfc en temps $O(n)$). Pour finir, la ligne 16 interdit toutes les boucles de la cfc courante s'il ne s'agit pas d'une cfc puits de $\mathcal{G}$ (par énumération sur les boucles en temps $O(n)$). D'où le théorème :

Théorème 3. *L'algorithme 2 filtre totalement la contrainte* $tree(\texttt{NTREE}, \texttt{VER})$ *en temps* $O(n \cdot m)$.

3 La contrainte de degré

La contrainte de degré associée à chaque sommet du graphe $\mathcal{G}$ permet de spécifier le degré négatif de chacun d'eux, i.e., le nombre de prédécesseurs que chaque sommet peut accepter dans une couverture. Ainsi, cette contrainte permet de modéliser une contrainte de partitionnement par des chemins, des arbres binaires, etc, sans avoir besoin de créer une contrainte globale spécifique pour chaque pattern. Cette section va montrer comment représenter une telle contrainte à partir d'une modélisation basée sur la contrainte de cardinalité gcc [8].

Ainsi, à chaque sommet v_i du graphe orienté $\mathcal{G}$ est associée une variable entière $\texttt{VER}[i]$.D, dont le domaine $[d^i_{min}, d^i_{max}]$ représente le nombre de prédécesseurs possibles pour v_i dans une solution. Soulignons que les boucles sur les racines potentielles sont ignorées. Cette contrainte de degré est très similaire à la contrainte gcc, excepté pour le traitement des racines potentielles : une valeur i affectée à v_i est simplement ignorée, i.e., qu'elle n'est pas comptée dans le nombre d'occurrences de la valeur i. La figure 3 montre le graphe de flot associé à cette contrainte gcc modifiée. Observons la présence d'un sommet ℓ supplémentaire correspondant aux racines potentielles du graphe. Formellement, ce graphe de flot est défini de la manière suivante :

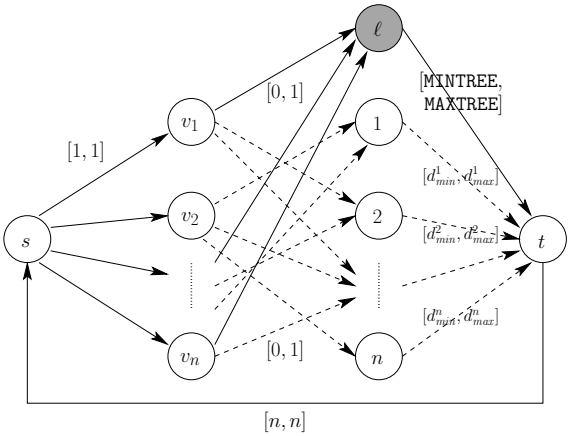


FIG. 3 – Graphe de flot associé à $\mathcal{G}$. Chaque sommet $v_i \in \mathcal{V}_{left}$ représente un sommet de $\mathcal{G}$, et chaque sommet $i \in \mathcal{V}_{right}$, avec $i \neq \ell$, représente un sommet de $\mathcal{G}$. Le sommet ℓ modélise l'ensemble des racines potentielles.

Définition 4. *Le* graphe de flot*, noté* $\mathcal{G}_f = (\mathcal{V}_f, \mathcal{E}_f)$*, d'une contrainte* $tree(\texttt{NTREE}, \texttt{VER})$ *est tel que :*

- $\mathcal{V}_f$ *est l'union des ensembles de sommets suivants :*
  - $\mathcal{V}_{left} = \{\texttt{VER}[i].\texttt{F} \mid i \in [1, n]\}$*, notés* v_i*.*
  - $\mathcal{V}_{right} = \{\texttt{VER}[i].\texttt{idx} \mid i \in [1, n]\}$*, notés* i*.*
  - $\{s, t, \ell\}$ *où* s*,* t*, et* ℓ *sont respectivement appelés les sommets* source*,* puits*, et* boucle*.*
- $\mathcal{E}_f$ *est l'union des arcs valués suivants :*
  - *Il existe un arc de* s *à chaque* $v_i \in \mathcal{V}_{left}$ *de capacité* $[1, 1]$*.*
  - *Un arc* (t, s) *de capacité* $[n, n]$*.*
  - *Il existe un arc de* $v_i \in \mathcal{V}_{left}$ *à* $j \in \mathcal{V}_{right}$ *de capacité* $[0, 1]$ *ssi* $j \in dom(\texttt{VER}[i].\texttt{F})$ *et* $j \neq i$*.*
  - *Il existe un arc de* $v_i \in \mathcal{V}_{left}$ *à* ℓ *de capacité* $[0, 1]$ *ssi* $i \in dom(\texttt{VER}[i].\texttt{F})$*.*
  - *Il existe un arc de* $i \in \mathcal{V}_{right}$ *à* t *de capacité* $\texttt{VER}[i].\texttt{D} = [d^i_{min}, d^i_{max}]$*.*
  - *Un arc* (ℓ, t) *de capacité* $[\texttt{MINTREE}, \texttt{MAXTREE}]$*.*

Remarquons que pour chaque $i \in \mathcal{V}_{right}$, si $i \neq \ell$ alors d^i_{min} et d^i_{max} représentent respectivement les valeurs minimale et maximale du degré négatif du sommet v_i de $\mathcal{G}$. Si $i = \ell$ alors $[d^i_{min}, d^i_{max}]$ est dominé par les nombres minimum (MINTREE) et maximum (MAXTREE) d'arbres possibles pour couvrir le graphe $\mathcal{G}$. En pratique, nous utilisons l'algorithme de filtrage classique basé sur le graphe de flot décrit dans [8] pour supprimer tous les arcs de $\mathcal{G}$ incompa-

Instances	filtrage			Quesada *et al.* [7]	
	réveils	échecs	temps	échecs	temps
SPMN_22full	2	0	$20ms$	0	$1.22s$
SPNM_52full	5	0	$170ms$	3	$45.03s$

TAB. 2 – Comparaison avec les instances de chemin Hamiltonien fournies dans [7]

tibles avec la contrainte sur le degré de chaque sommet.

La complexité de l'algorithme de filtrage dédié à la contrainte de degré est donnée par celle de la gcc, à savoir dans notre cas $O(n^2 \cdot m)$.

4 Expérimentations

Nous présentons maintenant une série d'évaluations de la contrainte d'arbre sur les problèmes de couverture par un chemin Hamiltonien ou par un arbre binaire. Dans le cas de chemins Hamiltoniens, nous comparerons les résultats de notre contrainte face à une contrainte spécifique de chemin [7] récemment publiée. Les tests présentés dans la suite ont été effectués à l'aide du solveur de contraintes *Choco* (qui est une librairie Java disponible à `http://choco.sf.net/`), sur une machine utilisant un Pentium M de 1.6GHz avec 1GO de RAM, dont 64MO alloués à la JVM.

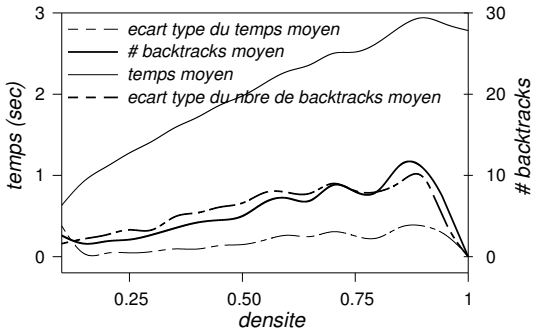
4.1 Existence d'un chemin Hamiltonien

Le problème d'existence d'un chemin Hamiltonien dans un graphe orienté peut s'exprimer de la manière suivante :

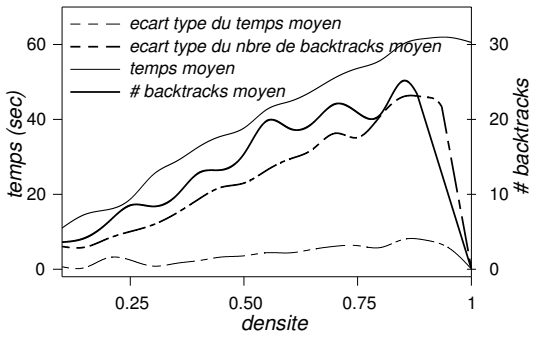
Ham-Path [4, p.199] : *Soit $\mathcal{G} = (\mathcal{V} \cup \{s, d\}, \mathcal{E})$ un graphe orienté, existe-t-il un chemin de s à d dans $\mathcal{G}$ passant exactement une fois par chaque sommet de $\mathcal{V}$?*

Le tableau 2 présente une comparaison entre les résultats de notre contrainte (colonne filtrage) et les résultats obtenus par une contrainte de chemin introduite dans [7]. Il faut cependant préciser que dans notre cas nous traitons exclusivement les couvertures totales du graphe $\mathcal{G}$ (i.e., tous les sommets appartiennent au chemin) alors que cette contrainte permet aussi la recherche d'un chemin ne couvrant qu'un sous-ensemble des sommets de $\mathcal{G}$. De plus, [7] traite également les contraintes de dominance (i.e., il existe un ordre au moins partiel entre les sommets à couvrir). Les deux instances SPNM_22full et SPNM_52full, extraites de [7], proposent des graphes de taille 22 et 52 peu denses.

Notons que les résultats obtenus par les deux contraintes sont très proches en termes du nombre d'échecs (le nombre de réveils n'est pas fourni dans les expérimentations de [7]), mais diffèrent plus significativement dans les temps d'exécution.



(a) Graphes aléatoires de taille $n = 50$

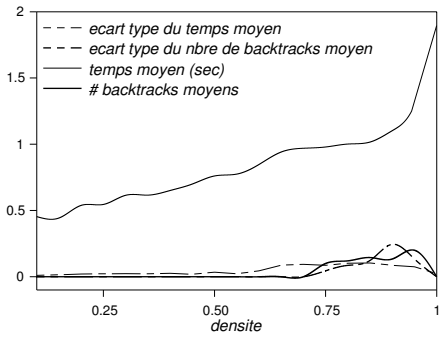


(b) Graphes aléatoires de taille $n = 100$

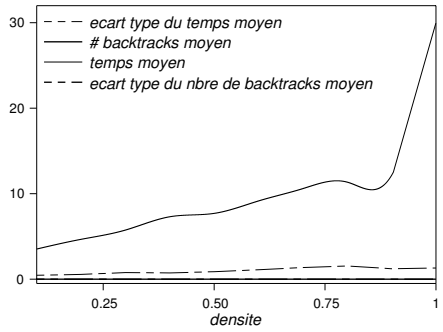
FIG. 4 – Temps de calcul et nombre d'échecs moyen pour vérifier l'existence d'un chemin Hamiltonien dans des graphes connexes générés aléatoirement

La figure 4 rapporte les mesures de performance (temps de calcul, nombre d'échecs et écart type pour les deux quantités) de la contrainte en fonction de la densité[4] du graphe $\mathcal{G}$ connexe généré sur des instances aléatoires. Ainsi, pour chaque densité $k \in [0.1, 1]$ (avec un incrément de 0.05), nous résolvons 50 instances. On remarque ainsi que pour des graphes d'ordre $n = 50$ (voir 4(a)), le nombre d'échecs maximum (11 échecs) est obtenu pour une densité de $\mathcal{G}$ de 85% alors que le temps de calcul maximum varie de 0.63 à 2.9 secondes. Si l'on observe maintenant des graphes d'ordre $n = 100$ (voir figure 4(b)), le nombre d'échecs maximum (25 échecs) est obtenu pour une densité de $\mathcal{G}$ de 85% alors que le temps de calcul maximum varie de 11 à 61 secondes. Il semble donc que la proportion d'échecs ne soit pas fonction de la taille du problème (le nombre d'échecs pour $n = 100$ est proportionnel au nombre d'échecs pour $n = 50$), mais plutôt de la *structure* de chaque instance et donc plus particulièrement de sa *densité*. En ce qui concerne le temps de calcul moyen, il est en accord avec la complexité théorique de la contrainte qui est dominée par la complexité de l'algorithme de filtrage de la contrainte de degré, à savoir $O(n^2 \cdot m)$, où n est le nombre

[4] La *densité* d'un graphe $\mathcal{G}$ d'ordre n est évaluée par le rapport entre le nombre m d'arcs de $\mathcal{G}$ et n^2.



(a) Graphes aléatoires de taille $n = 50$



(b) Graphes aléatoires de taille $n = 100$

FIG. 5 – Temps de calcul et nombre d'échecs moyen pour vérifier l'existence d'un arbre binaire couvrant dans des graphes connexes générés aléatoirement

de sommets de $\mathcal{G}$ et m est le nombre d'arcs dans $\mathcal{G}$.

4.2 Existence d'un arbre binaire couvrant

Le problème d'existence d'un arbre couvrant contraint sur le degré, connu comme NP-complet, est formulé de la manière suivante :

Arbre Couvrant sous Contrainte de Degré [4, p.206] : *Soit $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ un graphe orienté et $K < |\mathcal{V}|$, existe-t-il un arbre recouvrant dans $\mathcal{G}$ tel que tout sommet soit de degré inférieur ou égal à K ?*

La figure 5 rapporte les mesures de performance (temps de calcul, nombre d'échecs et écart types) de la contrainte en fonction de la densité du graphe $\mathcal{G}$. Le protocole utilisé est le même que pour le cas de chemin Hamiltonien. On remarque ainsi que pour des graphes d'ordre $n = 50$ (voir figure 5(a)), le nombre d'échecs maximum (0.2 échecs) est obtenu pour une densité de $\mathcal{G}$ de 93%, alors que le temps de calcul maximum varie de 0.3 à 1.9 secondes. Si l'on observe maintenant des graphes d'ordre $n = 100$ (voir figure 5(b)), le nombre d'échecs reste nul quelque soit la densité de $\mathcal{G}$, et le temps de calcul maximum varie de 3.5 à 30.01 secondes. Le temps de calcul moyen est toujours très fortement lié à la contrainte de degré sur chaque sommet.

5 Conclusion

Nous avons présenté un algorithme de filtrage pour une contrainte de partitionnement de graphes par des arbres sous contraintes de degré. La complexité d'une telle contrainte peut être découpée en deux parties. D'une part, la complexité de la contrainte d'arbre "pure", pour laquelle un filtrage arc-consistant est effectué en temps $O(n \cdot m)$, alors que l'existence d'une solution est vérifiée en temps $O(n + m)$. D'autre part, la complexité de maintenir une contrainte sur le nombre de prédécesseurs de chaque sommet du graphe des variables père, qui consiste à effectuer le filtrage d'une contrainte globale de cardinalité, ce qui est effectué en temps $O(n^2 \cdot m)$.

Si les expérimentations sur les problèmes NP-complets de couverture par un chemin Hamiltonien ou un arbre binaire se sont montrées très satisfaisantes quant au nombre d'échecs lors de la recherche, les performances en temps sur les graphes de forte densité sont beaucoup plus anecdotiques, ce qui laisse penser que le traitement de graphes de forte taille (à plus de 1000 sommets) s'avère difficile voire impossible. Cependant, il semble évident que la relaxation de certains algorithmes (par exemple, le calcul des points forts d'articulation), voire même l'optimisation de l'algorithme de filtrage dérivé de la contrainte de cardinalité gcc pour éviter des réveils inutiles, devraient permettre d'améliorer sensiblement les performances et autoriser le traitement d'instances de plus grande taille.

Les travaux futurs ont pour objectif d'ajouter de nouvelles contraintes pouvant enrichir significativement la contrainte de couverture par des arbres. Nous pensons, par exemple, à des contraintes introduisant un ordre entre les sommets à couvrir (contraintes dites de *précédence* ou de *dominance*), et à des contraintes d'*incomparabilité* s'assurant que deux sommets ne soient pas sur un même chemin dans une couverture.

Références

[1] Nicolas Beldiceanu and Évelyne Contejean. Introducing global constraints in CHIP. *Mathematical and Computer Modeling*, 20(12) :97–123, 1994.

[2] Nicolas Beldiceanu, Pierre Flener, and Xavier Lorca. The *tree* constraint. In Roman Barták and Michela Milano, editors, *Proceedings of the Second International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'05)*, volume 3524 of *LNCS*, pages 64–78. Springer-Verlag, 2005.

[3] Hadrien Cambazard and Eric Bourreau. Conception d'une contrainte globale de chemin. In *Proceedings of the Dixièmes Journées Nationales sur la Résolution Pratique de Problèmes NP-Complets (JNPC'04)*, pages 107–120, 2004. In French.

[4] M. R. Garey and D. S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. Freeman, New York, 1978.

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

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

[7] Luis Quesada, Peter Van Roy, Yves Deville, and Raphaël Collet. Using dominators for solving constrained path problems. In Pascal Van Hentenryck, editor, *Proceedings of PADL'06*, volume 3819 of *LNCS*, pages 73–87. Springer-Verlag, 2006.

[8] Jean-Charles Régin. Generalized arc consistency for global cardinality constraint. In *Proceedings of AAAI'96*, pages 209–215, 1996.

[9] Abhik Roychoudhury and Susmita Sur-Kolay. Efficient algorithms for vertex arboricity of planar graphs. In *Proceedings of the 15th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of *LNCS*, pages 37–51. Springer-Verlag, 1995.

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

[11] Meinolf Sellmann. Cost-based filtering for shortest path constraints. In *Proceedings of the 9th International Conference on the Principles and Practice of Constraint Programming (CP'03)*, volume 2833 of *LNCS*, pages 694–708. Springer-Verlag, 2003.

[12] R.E. Tarjan. Depth-first search and linear graph algorithms. In *SIAM Journal on Computing*, volume 1, pages 146–160, 1972.