RenPar'20 / SympA'14 / CFSE 8
Saint-Malo, France, du 10 au 13 mai 2011

Algorithme auto-stabilisant construisant un ensemble k-dominant minimal borné

Ajoy K. Datta[2], Stéphane Devismes[1], Karel Heurtefeux[1], Lawrence L. Larmore[2] et Yvan Rivierre[1]

[1]Université de Grenoble, VERIMAG – prénom.nom@imag.fr
[2]Université du Nevada Las Vegas – nom@cs.unlv.edu

Résumé
Nous proposons un algorithme distribué, asynchrone, silencieux et auto-stabilisant calculant un ensemble k-dominant minimal d'un réseau identifié quelconque de n processus. La taille de l'ensemble k-dominant calculé est d'au plus $\lceil \frac{n}{k+1} \rceil$ processus. Notre solution stabilise en $O(n)$ rondes et nécessite $O(\log n + k \log \frac{n}{k})$ bits de mémoire par processus.

Mots-clés : auto-stabilisation, ensemble k-dominant, systèmes distribués

1 Introduction

Un algorithme distribué est dit *auto-stabilisant* [10] si, à partir d'une configuration quelconque du système, toute exécution de l'algorithme atteint en un temps fini (et sans intervention extérieure) une configuration dite *légitime* à partir de laquelle tous les suffixes d'exécution possibles sont corrects. Cette configuration quelconque pouvant être la résultante d'une période finie où des fautes transitoires ont perturbé le système, un algorithme auto-stabilisant tolère naturellement ce type de fautes.
Soit $G = (V, E)$ un graphe simple, non orienté et connexe. Pour toute paire de nœuds p,q de V, la *distance* entre p et q dans G, notée $\|p, q\|$, est la longueur du plus court chemin de G liant p à q. Un sous-ensemble D de V est k-*dominant* dans G si tous les nœuds de $V \setminus D$ sont au plus à distance k d'un nœud de D.
Dans les réseaux, disposer d'un ensemble k-dominant D permet de hiérarchiser le réseau en k-grappes (k-cluster) : chaque processus de D est la tête d'une k-grappe (clusterhead) et les autres processus appartiennent à la k-grappe dont la tête est la plus proche d'eux.
Idéalement, il serait souhaitable de disposer d'un ensemble k-dominant *minimum*, c'est-à-dire un ensemble k-dominant le plus petit possible. Malheureusement, le calcul d'un tel ensemble est $\mathcal{NP}$-difficile [11]. À défaut, nous pouvons considérer le problème consistant à calculer un ensemble k-dominant *minimal*, c'est-à-dire un ensemble k-dominant dont aucun des sous-ensembles propres n'est k-dominant. Cependant le caractère minimal ne garantit pas que l'ensemble k-dominant soit de petite taille. Par exemple dans la figure 1, le singleton $\{v_0\}$ est un ensemble 1-dominant minimal, mais également l'ensemble complémentaire constitué des nœuds gris.

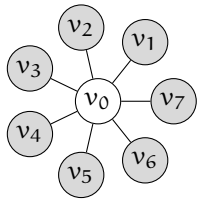


FIG. 1 : Ensemble 1-dominant minimal

Contributions. Ce travail a été initié à partir d'une borne supérieure sur la taille d'un ensemble k-dominant minimum donnée dans [13]. Nous montrons que la preuve de cette borne contient une erreur. Nous proposons ensuite un correctif ne modifiant pas la borne. À partir de cette nouvelle preuve, nous construisons un algorithme distribué, asynchrone, silencieux et auto-stabilisant calculant un ensemble k-dominant minimal d'un réseau identifié quelconque contenant au plus $\lceil \frac{n}{k+1} \rceil$ processus. Notre algorithme est prouvé en supposant un ordonnanceur faiblement équitable, stabilise en $O(n)$ rondes, et nécessite $O(\log n + k \log \frac{n}{k})$ bits par processus, où n est la taille du réseau. Enfin nous analysons par simulation notre algorithme. Les résultats expérimentaux montrent que la taille des ensembles k-dominants obtenus est, en général, beaucoup plus petite que la borne supérieure théorique.

État de l'art. Il existe de nombreux algorithmes distribués, asynchrones et auto-stabilisants calculant un ensemble k-dominant d'un réseau [8, 6, 2]. La solution de [8] stabilise en $O(k)$ rondes en utilisant $O(k \log n)$ bits par processus. Celle de [6] stabilise en $O(n)$ rondes en utilisant $O(\log n)$ bits par processus. La solution de [2] stabilise en $O(k.n)$ rondes en utilisant $O(k \log n)$ bits par processus. Seul l'algorithme de [6] calcule un ensemble k-dominant qui soit minimal. En outre, aucune de ces solutions ne permet d'obtenir des ensembles k-dominants de petite taille. Enfin, il existe de nombreux algorithmes distribués *non auto-stabilisants* pour calculer un ensemble k-dominant d'un réseau [13, 14].

Plan. La section 2 est consacrée au modèle. Dans la section 3, nous donnons un contre-exemple de la preuve de borne donnée dans [13] et nous proposons une correction. Dans la section 4, nous présentons une méthode de composition simplifiant la construction de notre algorithme ; celui-ci est décrit dans la section 5. La section 6 est dédiée aux simulations. Dans la section 7, nous exposons des perspectives. [1]

2 Modèle de calcul

Nous représentons un réseau par un graphe simple, non orienté et connexe $G = (V, E)$ où V est un ensemble de n processus et E est un ensemble de liens bidirectionnels. Les processus sont identifiés de manière unique. Dans la suite, l'identité du processus p sera simplement notée p.

Modèle à états. Nos algorithmes sont écrits dans *le modèle à états* [10]. Dans ce modèle, chaque processus peut lire ses propres variables et celles de ses voisins. En revanche, il ne peut écrire que dans ses propres variables. L'ensemble des voisins du processus p est noté $\mathcal{N}_p$.
Chaque processus exécute un *programme*. Nous appelons *algorithme* (distribué) $\mathcal{A}$ l'ensemble des programmes exécutés par les processus. Le *programme* d'un processus consiste en un ensemble fini d'actions. Chaque action est de la forme suivante : $\langle nom \rangle :: \langle garde \rangle \longrightarrow \langle instruction \rangle$. Le *nom* d'une action est un identifiant. La *garde* d'une action est un prédicat impliquant les variables de p et de ses voisins. L'*instruction* d'une action de p met à jour une ou plusieurs variables de p. Une action ne peut être exécutée que si elle est *activable*, c'est-à-dire si sa garde est vraie. Un processus est dit *activable* si au moins une de ses actions est activable. L'*état* d'un processus pour l'algorithme $\mathcal{A}$ est défini par les valeurs de ses variables de $\mathcal{A}$. Une configuration du réseau pour l'algorithme $\mathcal{A}$ est une instance des états de tous les processus pour $\mathcal{A}$. Nous notons $\gamma(p)$ l'état du processus p dans la configuration γ.
Notons $\mapsto$ la relation binaire entre les configurations du réseau pour $\mathcal{A}$ telle que $\gamma \mapsto \gamma'$ si et seulement si il est possible que le réseau passe de la configuration γ à la configuration γ' en un pas de calcul de $\mathcal{A}$. Une *exécution* de $\mathcal{A}$ est une séquence maximale de configurations $e = \gamma_0\gamma_1 \ldots \gamma_i \ldots$ telle que $\gamma_{i-1} \mapsto \gamma_i$ pour tout $i > 0$. Le terme « maximal » signifie que l'exécution est infinie ou s'arrête dans une configuration dite *terminale* dans laquelle aucune action de $\mathcal{A}$ n'est activable. Lors de chaque pas de calcul $\gamma_i \mapsto \gamma_{i+1}$, un à plusieurs processus peuvent exécuter, de façon atomique, une de leurs actions activables. Ceux-ci sont choisis par un *ordonnanceur*, lequel est caractérisé par son *équité*. Un ordonnanceur est *faiblement équitable* s'il permet à tout processus *continument* activable d'exécuter une action en un temps fini.
Un processus p subit une *neutralisation* en un pas de calcul $\gamma_i \mapsto \gamma_{i+1}$ si p est activable dans γ_i mais plus dans γ_{i+1}, alors qu'il n'exécute aucune action entre ces deux configurations. La neutralisation d'un processus représente la situation où au moins un voisin de p change d'état entre γ_i et γ_{i+1}, rendant ainsi fausses les gardes de toutes les actions de p. Nous utilisons la notion de *ronde*. La première *ronde* d'une exécution ρ, notée ρ', correspond au plus petit préfixe de ρ dans lequel chaque processus activable dans la configuration initiale exécute une action ou est neutralisé. Soit ρ'' un suffixe de ρ commençant par la dernière configuration de ρ'. La deuxième ronde de ρ est la première ronde de ρ'', etc.

Auto-stabilisation et silence. Une configuration est *conforme* à un prédicat si ce prédicat est satisfait dans cette configuration, dans le cas contraire la configuration *viole* ce prédicat. Par définition, toute configuration est conforme au prédicat vrai et aucune n'est conforme au prédicat faux. Soient R et S deux prédicats sur les configurations. Le prédicat R est *clos* pour les actions de l'algorithme si toutes les configurations de toutes les exécutions de l'algorithme commençant par une configuration conforme à R, sont aussi conformes à R. Le prédicat R *converge* vers S, si R et S sont clos et toutes les exécutions

[1] Pour les preuves omises de cet article, voir [5], `http://www-verimag.imag.fr/TR/TR-2011-6.pdf`.

commençant par une configuration conforme à R contiennent une configuration conforme à S. Un algorithme distribué est *auto-stabilisant* pour le prédicat R si vrai converge vers R. Pour tout algorithme qui stabilise à R, toute configuration qui est conforme à R est dite *légitime* et toute configuration qui viole R est dite *illégitime*. Un algorithme est dit *silencieux* si chacune de ses exécutions est finie.

3 Borne

Dans cette section, nous présentons une borne supérieure sur la taille d'un ensemble k-dominant minimum dans un réseau connexe quelconque. La preuve de cette borne est constructive : notre algorithme $\mathcal{DS}(k)$ présenté en section 5 s'en inspire. Cette borne est apparue dans [13]. Cependant, la preuve proposée dans [13] contient une erreur. Cette erreur est également présente dans des travaux ultérieurs, par exemple [14]. Ci-dessous, nous présentons un contre-exemple de cette preuve, puis nous proposons une correction, ne modifiant pas la borne.

Un arbre couvrant de $G = (V, E)$ enraciné à un processus r est un graphe connexe $T = (V_T, E_T)$ tel que $V_T = V$, $E_T \subseteq E$ et $|E_T| = |V_T| - 1$ où un processus r est distingué. Dans T, la *hauteur* d'un processus p, notée $h(p)$, représente sa distance à la racine r. La *hauteur* de T, notée $h(T)$, est égale à $\max_{p \in V_T} h(p)$. La hauteur du sous-arbre $T(p)$ enraciné à p est notée $h(T(p))$.

Soit T un arbre couvrant de $G = (V, E)$ enraciné en un processus r et notons h sa hauteur. La preuve originale consiste à diviser les processus en niveaux $T_0, \ldots, T_h$, en assignant au niveau T_i tous les processus de hauteur i. Ces niveaux sont ensuite fusionnés en $k+1$ ensembles $D_0, \ldots, D_k$ tels que $D_i = \bigcup_{j \geq 0} T_{i+j(k+1)}$.

Dans le cas où $k < h$, la preuve de [13] affirme que (1) le plus petit ensemble D_i contient au plus $\lceil \frac{n}{k+1} \rceil$ processus et que (2) chaque D_i ($i \in [0..k]$) est un ensemble k-dominant. La borne est alors obtenue en considérant le plus petit des ensembles D_i.

En fait, cet ensemble n'est pas toujours k-dominant. Par exemple, considérons le cas $k = 2$ dans le réseau présenté figure 2. D_2 est le plus petit ensemble, mais il n'est pas 2-dominant. En effet, u n'est 2-dominé par aucun processus de D_2 car le plus proche élément de D_2 (w) est à distance 3.

Cette erreur peut être corrigée sans changer la borne. Le problème en question n'apparaît que lorsque le plus petit ensemble D_i ($i \in [0..k]$), appelons le D_j, n'est pas D_0. Dans ce cas, il est possible qu'un processus feuille dont la hauteur est strictement inférieure à j ne soit k-dominé par aucun processus de D_j, comme le montre l'exemple précédent. Pour corriger cette erreur, nous procédons comme suit. Dans le cas où $k \geq h$ (alors $|D_0| = 1$) ou quand tous les D_i ($i \in [0..k]$) ont la même taille ($\lceil \frac{n}{k+1} \rceil$), nous choisissons D_0. Dans tous les autres cas, la taille du plus petit D_i ($i \in [0..k]$), disons D_j, est strictement inférieure à $\lceil \frac{n}{k+1} \rceil$ et $D_j \cup \{r\}$ est un ensemble k-dominant du réseau.

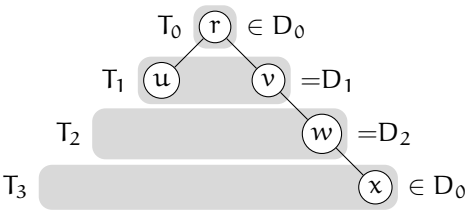


FIG. 2 : Contre-exemple

Théorème 1 *Soient* $G = (V, E)$ *un réseau connexe de* n *processus et* $k \geq 1$. *Il existe un ensemble* k*-dominant* D *tel que* $|D| \leq \lceil \frac{n}{k+1} \rceil$.

Preuve. Si $n = 0$, alors $\lceil \frac{n}{k+1} \rceil = 0 = |\emptyset|$ et $\emptyset$ est un ensemble k-dominant.

Supposons maintenant que $n > 0$. Soit T un arbre couvrant de G quelconque, enraciné en r et notons h sa hauteur. Divisons les processus de V en niveaux $T_0, \ldots, T_h$, en assignant au niveau T_i tous les processus de hauteur i. Puis fusionnons ces niveaux en $k+1$ ensembles $D_0, \ldots, D_k$ tels que $D_i = \bigcup_{j \geq 0} T_{i+j(k+1)}$.

- Supposons $k \geq h$. Alors D_0 ne contient que la racine, car tous les autres processus sont au plus à distance k de celle-ci. Ainsi D_0 est un ensemble k-dominant de taille $1 \leq \lceil \frac{n}{k+1} \rceil$.
- Supposons $k < h$. Alors, $\forall i \in [0..k]$, $|D_i| > 0$.
  - Supposons que $\forall i \in [0..k-1]$, $|D_i| = |D_{i+1}|$. Alors, $\forall i \in [0..k]$, $|D_i| = \lceil \frac{n}{k+1} \rceil$. Considérons maintenant un quelconque processus $v \notin D_0$. La hauteur de v, $h(v)$, satisfait $h(v) \bmod (k+1) \neq 0$. Soit u l'ancêtre de v tel que $h(u) = h(v) - (h(v) \bmod (k+1))$ (un tel processus existe car $h(v) \geq (h(v) \bmod (k+1))$). Comme $h(v) \bmod (k+1) \leq k$, alors u est à distance au plus k de v.

Or $h(v) = \lfloor \frac{h(v)}{k+1} \rfloor \times (k+1) + (h(v) \bmod (k+1))$, donc $h(u) = \lfloor \frac{h(v)}{k+1} \rfloor \times (k+1)$ et $h(u) \mod (k+1) = 0$, c'est-à-dire, $u \in D_0$.
Par conséquent, D_0 est un ensemble k-dominant tel que $|D_0| = \lceil \frac{n}{k+1} \rceil$.

– Supposons qu'il existe $i \in [0..k-1]$ tel que $|D_i| \neq |D_{i+1}|$. Soit $j \in [0..k]$ tel que $\forall i \in [0..k], |D_j| \leq |D_i|$. Alors $|D_j| < \lceil \frac{n}{k+1} \rceil$. Soit $D = D_j \cup \{r\}$ où r est la racine de T. Alors, $|D| \leq \lceil \frac{n}{k+1} \rceil$.
Considérons un quelconque processus $v \notin D$.
  – Si $h(v) \leq k$, alors v est à distance au plus k de r et $r \in D$.
  – Si $h(v) > k$, alors la hauteur de v, $h(v)$, satisfait $h(v) \bmod (k+1) \neq j$. Soit u l'ancêtre de v tel que $h(u) = h(v) - ((h(v) - j) \bmod (k+1))$. Comme $h(v) > k$ et $(h(v) - j) \bmod (k+1) < k+1$, alors $h(u) \geq 1$. Donc u existe et comme $h(v) - h(u) = (h(v) - j) \bmod (k+1) \leq k$, alors $\|u, v\| \leq k$.
  De plus, $h(u) \bmod (k+1) = (h(v) - ((h(v) - j) \bmod (k+1))) \bmod (k+1) = ((h(v) \bmod (k+1)) - ((h(v) - j) \bmod (k+1))) \bmod (k+1) = ((h(v) - (h(v) - j)) \bmod (k+1)) \bmod (k+1) = j \bmod (k+1)$.
  Comme $j \leq k$, $h(u) \bmod (k+1) = j$, alors $u \in D_j$, c'est-à-dire, $u \in D$.

Ainsi, D est un ensemble k-dominant tel que $|D| \leq \lceil \frac{n}{k+1} \rceil$. □

4 Composition collatérale hiérarchique

Pour simplifier notre algorithme, nous utilisons une variante de la *composition collatérale*.Lorsque nous composons collatéralement deux algorithmes $\mathcal{A}$ et $\mathcal{B}$, ceux-ci sont exécutés concurremment et $\mathcal{B}$ utilise la sortie de $\mathcal{A}$ dans ses calculs. Dans notre variante, nous modifions le code de $\mathcal{B}$ de sorte qu'un processus ne puisse exécuter une action de $\mathcal{B}$ que si aucune action de $\mathcal{A}$ n'est activable.

Définition 1 *Soient $\mathcal{A}$ et $\mathcal{B}$ deux algorithmes tels qu'aucune variable écrite par $\mathcal{B}$ n'apparaisse dans $\mathcal{A}$. La* composition collatérale hiérarchique *de $\mathcal{A}$ et $\mathcal{B}$, notée $\mathcal{B} \circ \mathcal{A}$, est l'algorithme défini comme suit :*
– *$\mathcal{B} \circ \mathcal{A}$ contient toutes les variables de $\mathcal{A}$ et $\mathcal{B}$;*
– *$\mathcal{B} \circ \mathcal{A}$ contient toutes les actions de $\mathcal{A}$;*
– *pour toute action $G_i \rightarrow S_i$ de $\mathcal{B}$, $\mathcal{B} \circ \mathcal{A}$ contient l'action $\neg C \wedge G_i \rightarrow S_i$ où C est la disjonction de toutes les gardes des actions de $\mathcal{A}$.*

Propriété 1 *$\mathcal{B} \circ \mathcal{A}$ stabilise à SP en supposant un ordonnanceur faiblement équitable si :*
– *$\mathcal{A}$ est un algorithme silencieux auto-stabilisant en supposant un ordonnanceur faiblement équitable ;*
– *$\mathcal{B}$ stabilise à SP à partir de toute configuration où aucune action de $\mathcal{A}$ n'est activable.* [2]

5 Algorithme $\mathcal{SMDS}(k)$

Dans cette section, nous présentons un algorithme auto-stabilisant silencieux, appelé $\mathcal{SMDS}(k)$ pour *Small Minimal k-Dominating Set*, qui calcule un ensemble k-dominant minimal d'au plus $\lceil \frac{n}{k+1} \rceil$ processus dans un réseau identifié quelconque en supposant un ordonnanceur faiblement équitable. Cet algorithme est une composition collatérale hiérarchique de quatre algorithmes auto-stabilisants silencieux, $\mathcal{SMDS}(k) = \mathcal{MIN}(k) \circ \mathcal{DS}(k) \circ \mathcal{ST} \circ \mathcal{LE}$ où :
– $\mathcal{LE}$ est un algorithme d'élection de leader.
– $\mathcal{ST}$ est un algorithme qui construit un arbre couvrant enraciné au processus élu par $\mathcal{LE}$.
– $\mathcal{DS}(k)$ calcule un ensemble k-dominant d'au plus $\lceil \frac{n}{k+1} \rceil$ nœuds utilisant l'arbre couvrant construit par $\mathcal{ST}$.
– $\mathcal{MIN}(k)$ rend minimal l'ensemble k-dominant calculé par $\mathcal{DS}(k)$.

Module $\mathcal{LE}$. $\mathcal{LE}$ est un algorithme auto-stabilisant silencieux d'élection de leader pour des réseaux identifiés quelconques, en supposant un ordonnanceur faiblement équitable. Un tel algorithme peut être trouvé dans [7]. Dans la suite de cet article, nous supposerons l'existence d'un prédicat de sortie $\texttt{EstLeader}_p$, défini pour tout processus p, tel que $\texttt{EstLeader}_p$ est vrai si et seulement si p croit être le leader. Ainsi, $\mathcal{LE}$ converge vers le prédicat $SP_{\mathcal{LE}}$ défini comme suit : $SP_{\mathcal{LE}}$ est vrai si et seulement si il existe un unique processus p tel que $\texttt{EstLeader}_p$.

[2] Pour rappel, la spécification de $\mathcal{A}$ est satisfaite dans une telle configuration.

Module $\mathcal{ST}$. $\mathcal{ST}$ est un algorithme auto-stabilisant silencieux construisant un arbre couvrant dans un réseau enraciné quelconque, en supposant un ordonnanceur faiblement équitable. De nombreuses variantes de cet algorithme existent pour construire un arbre couvrant quelconque [3], en largeur d'abord [12] ou en profondeur d'abord [4], par exemple. $\mathcal{ST}$ utilise la sortie de $\mathcal{LE}$ pour choisir la racine de l'arbre couvrant. En d'autres termes, $\mathcal{ST}$ construit un arbre couvrant enraciné au leader élu par $\mathcal{LE}$. Par la suite, nous supposerons que la sortie de $\mathcal{ST}$ est une macro appelée $\texttt{Parent}_p$ définie pour tout processus p. $\texttt{Parent}_p$ retourne $\perp$ si p croit être la racine de l'arbre couvrant, sinon $\texttt{Parent}_p$ désigne un voisin q en tant que parent de p dans l'arbre couvrant. Ainsi, $\mathcal{ST} \circ \mathcal{LE}$ converge vers le prédicat $SP_{\mathcal{ST}}$ défini comme suit : $SP_{\mathcal{ST}}$ est vrai si et seulement si (1) il existe un unique processus r tel que $\texttt{Parent}_r = \perp$ et (2) le graphe $T = (V, E_T)$ où $E_T = \{\{p, \texttt{Parent}_p\}, \forall p \in V \setminus \{r\}\}$ est un arbre couvrant. De la propriété 1, nous déduisons : $\mathcal{ST} \circ \mathcal{LE}$ est un algorithme silencieux qui stabilise à $SP_{\mathcal{ST}}$ en supposant un ordonnanceur faiblement équitable.

Module $\mathcal{DS}(k)$. $\mathcal{DS}(k)$ (voir algorithme 1 pour une description formelle) utilise l'arbre couvrant T construit par $\mathcal{ST}$ pour calculer un ensemble k-dominant d'au plus $\lceil \frac{n}{k+1} \rceil$ processus. Il est basé sur la construction proposée dans la preuve du théorème 1. Informellement, $\mathcal{DS}(k)$ utilise trois variables à chaque processus p :

- $p.couleur \in [0..k]$. Dans cette variable, p calcule $h(p)$ mod $(k+1)$ (c'est-à-dire sa hauteur dans T modulo $k+1$) de façon descendante en utilisant l'action Colorer. Une fois que $\mathcal{DS}(k)$ a stabilisé, chaque ensemble D_i défini en section 3 correspond donc à l'ensemble $\{p \in V \mid p.couleur = i\}$.
- $p.pop$ est un tableau de $k+1$ entiers naturels. Dans chaque case $p.pop[i]$, quel que soit $i \in [0..k]$, p calcule le nombre de processus de couleur i dans le sous-arbre $T(p)$, c'est-à-dire les processus q tels que $q.couleur = i$. Ce calcul est effectué de façon ascendante en utilisant l'action Compter. Ainsi, après que $\mathcal{DS}(k)$ a stabilisé, r connaît la taille de chaque ensemble D_i.
- $p.min \in [0..k]$. Dans cette variable, p calcule le plus petit indice i d'un plus petit ensemble D_i non vide, c'est-à-dire la valeur la moins utilisée pour colorer des processus du réseau. Cette valeur est évaluée de façon descendante en utilisant l'action Choisir à partir des valeurs calculées dans le tableau $r.pop$. En effet, une fois que les valeurs de $r.pop$ sont exactes, r peut évaluer dans $r.min$ la couleur la moins utilisée (en cas d'égalité, nous choisissons le plus petit indice). Ensuite, la valeur de $r.min$ est propagée dans l'arbre vers les processus feuilles.

D'après le théorème 1, une fois que $\mathcal{DS}(k)$ a stabilisé, l'ensemble des processus p tels que $p = r$ ou $p.couleur = p.min$, c'est-à-dire l'ensemble $\{p \in V \mid \texttt{EstDominant}_p\}$, est un ensemble k-dominant d'au plus $\lceil \frac{n}{k+1} \rceil$ processus. Ainsi, $\mathcal{DS}(k) \circ \mathcal{ST} \circ \mathcal{LE}$ converge vers le prédicat $SP_{\mathcal{DS}(k)}$ défini comme suit : $SP_{\mathcal{DS}(k)}$ est vrai si et seulement si l'ensemble $\{p \in V \mid \texttt{EstDominant}_p\}$ est un ensemble k-dominant d'au plus $\lceil \frac{n}{k+1} \rceil$ processus. Chacune de ses variables se propageant en $O(n)$ rondes (voir [5] pour la preuve complète), on en déduit que $\mathcal{DS}(k) \circ \mathcal{ST} \circ \mathcal{LE}$ stabilise à $SP_{\mathcal{DS}(k)}$ en $O(n)$ rondes en supposant un ordonnanceur faiblement équitable.

La figure 3 montre l'exemple d'un ensemble 2-dominant calculé par $\mathcal{DS}(2) \circ \mathcal{ST} \circ \mathcal{LE}$. Dans cette figure, les traits épais représentent les arêtes de l'arbre couvrant et les traits en pointillés les autres arêtes. Dans cet exemple, après que $\mathcal{DS}(2) \circ \mathcal{ST} \circ \mathcal{LE}$ a stabilisé, $r.pop[0] = 5$, $r.pop[1] = 5$ et $r.pop[2] = 3$, donc $r.min = 2$, ce qui signifie que la couleur la moins fréquente est 2 : $D_2 = \{p_4, p_9, p_{10}\}$ et $|D_2| = 3$. Dans ce cas, l'ensemble 2-dominant en sortie de $\mathcal{DS}(2) \circ \mathcal{ST} \circ \mathcal{LE}$ est $SD = \{r\} \cup D_2$, c'est-à-dire $\{r, p_4, p_9, p_{10}\}$. Cet ensemble 2-dominant suit la borne donnée par le théorème 1. En effet, la taille de SD est 4, ce qui est inférieur à $\lceil \frac{13}{2+1} \rceil = 5$. Néanmoins, SD n'est pas minimal. Par exemple, $\{r, p_{10}\}$ est un sous-ensemble de SD qui 2-domine le réseau. Ce dernier ensemble 2-dominant est minimal, c'est-à-dire qu'aucun de ses sous-ensembles ne sont 2-dominants.

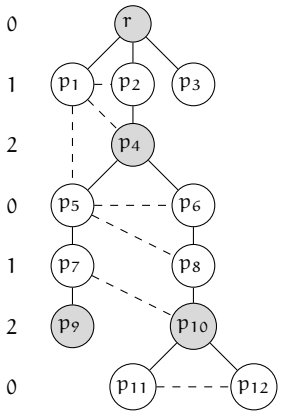


FIG. 3 : Ensemble calculé par $\mathcal{DS}(2)$

Algorithme 1 $\mathcal{DS}(k)$, programme pour tout processus p

Entrée : $\texttt{Parent}_p \in \mathcal{N}_p \cup \{\perp\}$

Variables : $p.couleur \in [0..k]$; $p.pop[i] \in \mathbb{N}, \forall i \in [0..k]$; $p.min \in [0..k]$

Macros :

$\texttt{EvaluerCouleur}_p = 0$ **si** $(\texttt{Parent}_p = \perp)$ **sinon** $(\texttt{Parent}_p.couleur + 1) \bmod (k+1)$
$\texttt{Fils}_p = \{q \in \mathcal{N}_p \mid \texttt{Parent}_q = p\}$
$\texttt{EvaluerPop}_p(i) = (1$ **si** $(p.couleur = i)$ **sinon** $0) + \sum_{q \in \texttt{Fils}_p} q.pop[i]$
$\texttt{PopMin}_p = \min_{i \in [0..k]} \{p.pop[i] \mid p.pop[i] > 0\}$
$\texttt{CouleurMin}_p = \min_{i \in [0..k]} \{i \mid p.pop[i] = \texttt{PopMin}_p\}$
$\texttt{EvaluerMin}_p = \texttt{CouleurMin}_p$ **si** $(\texttt{Parent}_p = \perp)$ **sinon** $\texttt{Parent}_p.min$

Prédicats :

$\texttt{CouleurAssortie}_p \equiv p.couleur = \texttt{EvaluerCouleur}_p$
$\texttt{ComptesRonds}_p \equiv \forall i \in [0..k], p.pop[i] = \texttt{EvaluerPop}_p(i)$
$\texttt{MinChoisi}_p \equiv p.min = \texttt{EvaluerMin}_p$
$\texttt{EstDominant}_p \equiv \texttt{Parent}_p = \perp \vee p.couleur = p.min$

Actions :

Colorer :: $\neg \texttt{CouleurAssortie}_p \longrightarrow p.couleur \leftarrow \texttt{EvaluerCouleur}_p$
Compter :: $\texttt{CouleurAssortie}_p \wedge \neg \texttt{ComptesRonds}_p \longrightarrow \forall i \in [0..k], p.pop[i] \leftarrow \texttt{EvaluerPop}_p(i)$
Choisir :: $\texttt{CouleurAssortie}_p \wedge \texttt{ComptesRonds}_p \wedge \neg \texttt{MinChoisi}_p \longrightarrow p.min \leftarrow \texttt{EvaluerMin}_p$

Module $\mathcal{MIN}(k)$. $\mathcal{MIN}(k)$ calcule un ensemble k-dominant minimal qui est un sous-ensemble de l'ensemble k-dominant calculé par $\mathcal{DS}(k)$. En section 6, nous verrons que cette minimisation permet un gain non négligeable.

Ce dernier module de notre algorithme peut être réalisé en utilisant l'algorithme auto-stabilisant silencieux $\mathcal{MIN}(k)$ de [6]. Cet algorithme prend un ensemble k-dominant I en entrée, et construit un sous-ensemble k-dominant de I qui est minimal. La connaissance de I est répartie, c'est-à-dire que chaque processus p n'utilise que son entrée $\texttt{EstDominant}_p$ pour savoir s'il est dans l'ensemble k-dominant ou non. À partir de cette entrée, $\mathcal{MIN}(k)$ positionne la variable booléenne de sortie $p.dansD$ de chaque processus p, de sorte qu'en $O(n)$ rondes, $\{p \in V \mid p.dansD\}$ est un ensemble k-dominant minimal.

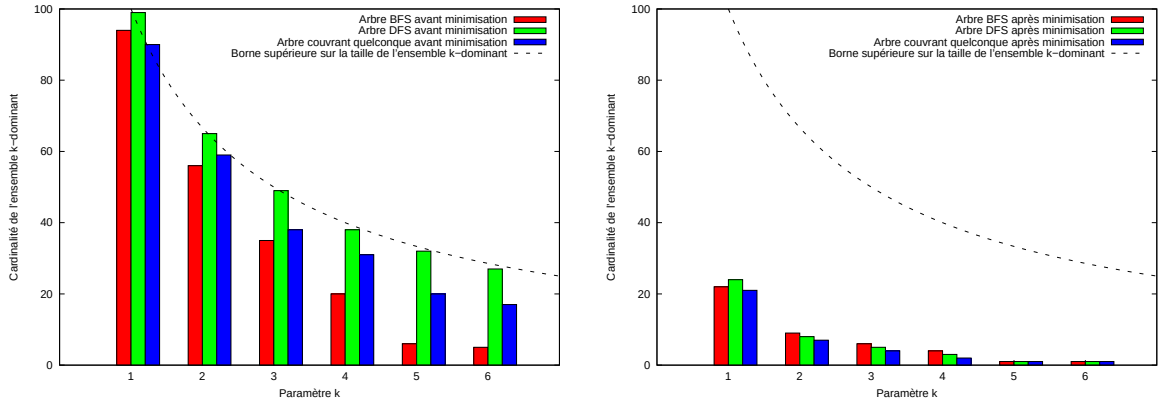
En utilisant la sortie de l'algorithme $\mathcal{DS}(k) \circ \mathcal{ST} \circ \mathcal{LE}$ comme entrée pour l'algorithme $\mathcal{MIN}(k)$, la taille de l'ensemble k-dominant minimal ainsi obtenu est toujours bornée par $\lceil \frac{n}{k+1} \rceil$. Ainsi, $\mathcal{SMDS}(k) = \mathcal{MIN}(k) \circ \mathcal{DS}(k) \circ \mathcal{ST} \circ \mathcal{LE}$ stabilise au prédicat $SP_{\mathcal{SMDS}(k)}$ défini comme suit : $SP_{\mathcal{SMDS}(k)}$ est vrai si et seulement si l'ensemble $\{p \in V \mid p.dansD\}$ est k-dominant minimal de taille au plus $\lceil \frac{n}{k+1} \rceil$.

Analyse de complexité. Considérons d'abord la complexité en rondes de $\mathcal{SMDS}(k)$. En utilisant l'algorithme de [7], le module $\mathcal{LE}$ stabilise en $O(n)$ rondes. Après que celui-ci a stabilisé, le module $\mathcal{ST}$ stabilise en $O(n)$ rondes si nous utilisons l'algorithme de [12]. Une fois que l'arbre couvrant est disponible, $\mathcal{DS}(k)$ stabilise en $O(n)$ rondes, comme nous l'avons montré précédemment. Pour finir, l'ensemble k-dominant calculé est rendu minimal par $\mathcal{MIN}(k)$ en $O(n)$ rondes, d'après [6]. Donc : $\mathcal{SMDS}(k)$ stabilise à $SP_{\mathcal{SMDS}(k)}$ en $O(n)$ rondes.

Considérons la complexité en espace de $\mathcal{SMDS}(k)$. $\mathcal{LE}$, $\mathcal{ST}$ et $\mathcal{MIN}(k)$ peuvent être implémentés avec $O(\log n)$ bits par processus [7, 12, 6]. Pour chaque processus, $\mathcal{DS}(k)$ utilise deux variables d'un domaine de $k+1$ éléments, et un tableau de $k+1$ entiers. Cependant, dans une configuration terminale, la plus petite valeur non nulle d'une case est au plus $\lceil \frac{n}{k+1} \rceil$. Donc $\mathcal{DS}(k)$ fonctionne toujours si nous remplaçons toute affectation d'une valeur λ à une case par $\min(\lambda, \lceil \frac{N}{k+1} \rceil + 1)$ où N est un majorant de n. Dans ce cas, chaque tableau peut être implémenté en n'utilisant que $O(k \log \frac{n}{k})$ bits. Il faut bien noter que cette complexité ne peut être obtenue qu'en faisant l'hypothèse que chaque processus a connaissance du majorant. Ainsi, $\mathcal{SMDS}(k)$ peut être implémenté en utilisant $O(\log n + k \log \frac{n}{k})$ bits par processus.

6 Simulations

Tous les résultats fournis dans cette section proviennent de WSNet [1], un simulateur événementiel pour les réseaux sans fil. Nous avons adapté, d'après [9], notre algorithme au modèle à passage de messages. Dans nos simulations, nous avons répartis n processus (n variant de 50 et 400 nœuds) de manière aléatoire uniforme dans un plan carré de $100m$ de côté. Les processus sont immobiles et équipés de radio. Deux processus u et v peuvent communiquer si et seulement si la distance euclidienne qui les sépare est au



(a) (b)

FIG. 4 : Taille moyenne de l'ensemble k-dominant en fonction de k : (a) avant minimisation ($n = 200$ et degré moyen $\in]10, 20[$) ; (b) après minimisation ($n = 200$ et degré moyen $\in]10, 20[$)

plus rad, où rad est la portée de transmission. Ainsi, la topologie du réseau est isomorphe à un graphe de disques uniformes (UDG). Par souci de simplicité, nous considérons les couches physiques et MAC comme idéales : il n'y a ni interférence ni collision. Il faut noter que notre algorithme fonctionne même si les communications sont non fiables. En outre, les processus sont concurrents et asynchrones.

En faisant varier la portée de transmission entre $10m$ et $50m$, nous pouvons contrôler le degré moyen du réseau qui varie entre 10 et 50. Enfin, nous avons effectué nos expériences pour k variant de 1 à 6.

Afin d'étudier l'influence du type d'arbre sur les performances de $\mathcal{SMDS}(k)$, nous simulons notre protocole avec trois constructions d'arbre couvrant différentes : en profondeur d'abord (DFS) [4], en largeur d'abord (BFS) [12] et quelconque [3].

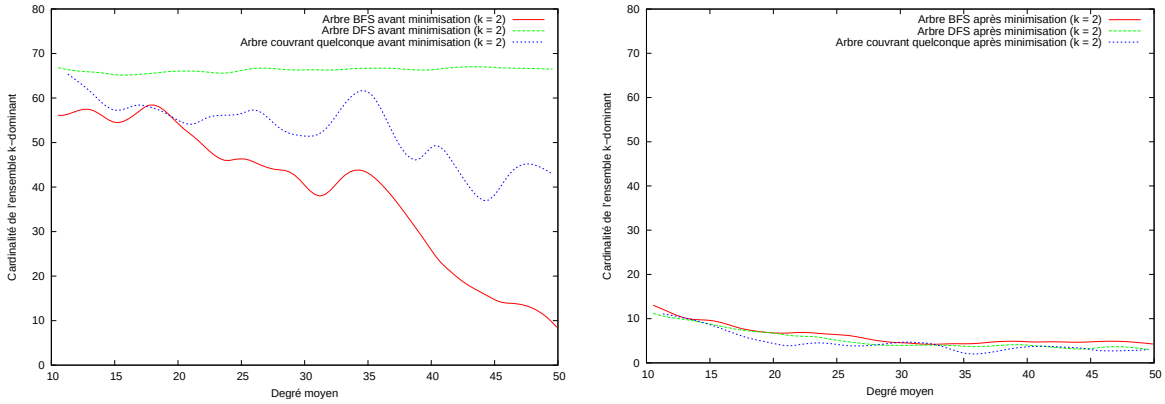
Dans le contexte des réseaux ad hoc et des réseaux de capteurs, il est intéressant d'étudier les performances moyennes des algorithmes $\mathcal{DS}(k)$ et $\mathcal{MIN}(k)$ dans des topologies aléatoires, au delà du pire cas. En particulier, le choix d'un arbre couvrant spécifique influe-t-il de manière significative sur la taille de l'ensemble k-dominant construit par $\mathcal{DS}(k)$ ou $\mathcal{DS}(k) \circ \mathcal{MIN}(k)$? Quel est le gain dû à $\mathcal{MIN}(k)$? Ce gain est-il le même avec tous les arbres couvrants ? Comment varie la taille de l'ensemble k-dominant de sortie par rapport à k, à n et au degré moyen du réseau ? Nous mettons ici en évidence les performances, en termes de taille de l'ensemble k-dominant construit par $\mathcal{DS}(k)$ et $\mathcal{DS}(k) \circ \mathcal{MIN}(k)$ dans des topologies aléatoires, par rapport à k, à n, à l'arbre couvrant choisi et au degré moyen du réseau.

La figure 4a montre l'évolution de la taille de l'ensemble k-dominant en fonction de k, après stabilisation de l'algorithme $\mathcal{DS}(k)$. Nous observons une forte différence entre les ensembles k-dominants calculés selon le type d'arbre couvrant. L'arbre DFS, par construction, induit un grand nombre de processus k-dominants. Nous remarquons que la taille moyenne obtenue par simulation est proche de la borne supérieure théorique dans ce cas. D'autre part, construire des ensembles k-dominants sur des arbres quelconques ou BFS permet de meilleures performances. Dans ces résultats, la hauteur de l'arbre a également un impact majeur sur la taille de l'ensemble k-dominant.

L'influence du degré moyen peut être observée dans la figure 5a. La taille de l'ensemble k-dominant construit sur un arbre DFS est constante et élevée, alors que la taille d'un tel ensemble construit sur un autre arbre est décroissante. En effet, quand le degré moyen augmente, le diamètre du réseau diminue. Dans le cas des arbres quelconques et BFS, cela conduit à une diminution de la hauteur et donc à une baisse de la taille de l'ensemble k-dominant.

Quel que soit l'arbre couvrant, la taille de l'ensemble k-dominant construit par $\mathcal{DS}(k)$ dans un UDG est proche du pire cas (Fig. 4a et Fig. 5a), sauf pour le BFS. Dans ce contexte, il est intéressant d'étudier si $\mathcal{MIN}(k)$ est capable de réduire significativement la taille de l'ensemble k-dominant calculée par $\mathcal{DS}(k)$.

La figure 4b illustre à la fois le gain obtenu par le quatrième module en termes de taille de l'ensemble k-dominant et les différences entre les ensembles k-dominants minimisés par $\mathcal{MIN}(k)$ selon l'arbre à partir duquel ils ont été calculés. Pour les trois types d'arbres couvrants et pour $1 \leq k \leq 4$, la diminution moyenne obtenue est supérieure à 75%. Pour des valeurs plus élevées de k, les bonnes performances de



(a) (b)

FIG. 5 : Taille moyenne de l'ensemble k-dominant en fonction du degré moyen : (a) avant minimisation ($k = 2$ et $n = 200$) ; (b) après minimisation ($k = 2$ et $n = 200$)

$\mathcal{DS}(k)$ sur l'arbre BFS limitent les possibilités d'obtenir des gains importants en appliquant $\mathcal{MIN}(k)$. Ici, la taille de l'ensemble k-dominant obtenue par $\mathcal{MIN}(k)$ est assez similaire quel que soit le type d'arbre considéré, malgré un léger avantage pour les arbres quelconques. Le quatrième module fourni par $\mathcal{MIN}(k)$ permet d'homogénéiser la taille des ensembles k-dominants quel que soit le degré moyen. La figure 5b illustre cette propriété pour $k = 2$. Les simulations montrent également qu'il est efficace sur tous les types d'arbres considérés : pour $k = 2$, $n = 200$ et un degré moyen dans $]10, 20[$, la minimisation permet de réduire la taille de l'ensemble k-dominant de 85% en moyenne.
Pour résumer, nos simulations soulignent le fait que la taille de l'ensemble k-dominant calculé est significativement plus petite que la borne théorique dans la majorité des cas.

7 Perspectives

Dans de futurs travaux, nous souhaiterions trouver un algorithme distribué auto-stabilisant pour calculer un ensemble k-dominant minimal dont la taille est une approximation constante de la taille du minimum, c'est-à-dire un algorithme qui calcule un ensemble k-dominant minimal de taille s telle que $\frac{s}{s_{opt}} \leq c$ où c est une constante et s_{opt} est la taille d'un ensemble k-dominant minimum du réseau.

Bibliographie

1. Elyes Ben Hamida, Guillaume Chelius, and Jean-Marie Gorce. Scalable versus accurate physical layer modeling in wireless network simulations. *pads*, 0 :127–134, 2008.
2. Eddy Caron, Ajoy Kumar Datta, Benjamin Depardon, and Lawrence L. Larmore. A self-stabilizing k-clustering algorithm for weighted graphs. *JPDC*, 70(11) :1159–1173, 2010.
3. Nian-Shing Chen, Hwey-Pyng Yu, and Shing-Tsaan Huang. A self-stabilizing algorithm for constructing spanning trees. *Inf. Process. Lett.*, 39 :147–151, 1991.
4. Zeev Collin and Shlomi Dolev. Self-stabilizing depth-first search. *Inf. Process. Lett.*, 49 :297–301, 1994.
5. Ajoy K. Datta, Stéphane Devismes, Karel Heurtefeux, Lawrence L. Larmore, and Yvan Rivierre. Self-stabilizing small k-dominating sets. Technical report, VERIMAG, 2011. `http://www-verimag.imag.fr/TR/TR-2011-6.pdf`.
6. Ajoy K. Datta, Stéphane Devismes, and Lawrence L. Larmore. A self-stabilizing o(n)-round k-clustering algorithm. In *SRDS*, pages 147–155, 2009.
7. Ajoy K. Datta, L. L. Larmore, and P. Vemula. Self-stabilizing leader election in optimal space. In *SSS*, pages 109–123, 2008.
8. Ajoy K. Datta, Lawrence L. Larmore, and Priyanka Vemula. A Self-Stabilizing O(k)-Time k-Clustering Algorithm. *The Computer Journal*, page bxn071, 2009.
9. Sylvie Delaët, Bertrand Ducourthial, and Sébastien Tixeuil. Self-stabilization with r-operators revisited. In Ted Herman and Sébastien Tixeuil, editors, *Self-Stabilizing Systems*, volume 3764 of *Lecture Notes in Computer Science*, pages 68–80. Springer, 2005.
10. Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17 :643–644, 1974.
11. M. R. Garey and David S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
12. Shing-Tsaan Huang and Nian-Shing Chen. A self-stabilizing algorithm for constructing breadth-first trees. *Inf. Process. Lett.*, 41 :109–117, 1992.
13. David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *JACM*, 36(3) :510–530, 1989.
14. Lucia Draque Penso and Valmir C. Barbosa. A distributed algorithm to find k-dominating sets. *Discrete Appl. Math.*, 141(1-3) :243–253, 2004.