Algorithme autostabilisant construisant un petit ensemble k-dominant

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

1. *School of Computer Science*
   University of Nevada, Las Vegas, USA
   prénom.nom@cs.unlv.edu

2. *Laboratoire Verimag*
   Université de Grenoble, France
   prénom.nom@imag.fr

RÉSUMÉ. *Nous proposons un algorithme distribué, asynchrone, silencieux et autostabilisant 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. À partir d'un transformateur présenté également dans cet article, nous obtenons une solution fonctionnant sous un ordonnanceur inéquitable (le plus général des ordonnanceurs). Notre solution stabilise en $O(n)$ rondes et $O(\mathcal{D}n^3)$ pas de calcul, où $\mathcal{D}$ est le diamètre du réseau. Enfin, elle nécessite $O(\log k + \log n + k \log \frac{N}{k})$ bits de mémoire par processus où N est un majorant de n.*

ABSTRACT. *We propose a distributed asynchronous silent self-stabilizing algorithm for finding a minimal k-dominating set in an arbitrary identified network of n processes. The size of the computed k-dominating set is at most $\lceil \frac{n}{k+1} \rceil$ processes. Using a transformer also proposed in this paper, we make our algorithm working under an unfair daemon (the weakest scheduling assumption). The complexity of our solution is in $O(n)$ rounds and $O(\mathcal{D}n^3)$ steps using $O(\log k + \log n + k \log \frac{N}{k})$ bits per process where $\mathcal{D}$ is the diameter of the network and N is an upper bound on n.*

MOTS-CLÉS : *autostabilisation, ensemble k-dominant, systèmes distribués.*

KEYWORDS : *self-stabilization, k-dominating set, distributed systems.*

DOI : 10.3166/TSI.31.1273-1299

1. Introduction

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 reliant 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 k-partitionner le réseau, c'est-à-dire de le hiérarchiser 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.

Une application majeure du k-partitionnement est de rendre les communications entre les nœuds d'un réseau plus efficaces. Généralement, la règle suivante est adoptée : les nœuds qui ne sont pas des têtes de k-grappe ne communiquent qu'avec des membres de leur k-grappe et les têtes de k-grappe communiquent entre elles *via* des « super arêtes » consistant en des chemins dans le réseau.

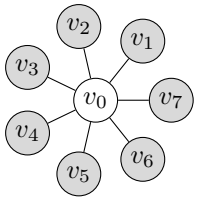


Figure 1. Exemple d'ensemble 1*-dominant minimal de grande taille*

Idéalement, il serait souhaitable de disposer d'un ensemble k-dominant *minimum*, c'est-à-dire un ensemble k-dominant de plus petite taille (la taille étant le nombre de nœuds de l'ensemble). Malheureusement, le calcul d'un tel ensemble est $\mathcal{NP}$-difficile (Garey, Johnson, 1979). 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 l'ensemble complémentaire constitué des nœuds gris en est également un et sa taille est $|V| - 1$.

1.1. État de l'art

Un algorithme distribué est dit *autostabilisant* (Dijkstra, 1974) 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 autostabilisant tolère naturellement ce type de fautes.

Il existe de nombreux algorithmes distribués, asynchrones et autostabilisants calculant un ensemble k-dominant d'un réseau, par exemple ceux de (Datta *et al.*, 2010 ; 2009 ; Caron *et al.*, 2010). La correction de tous ces algorithmes est prouvée en supposant un ordonnanceur inéquitable. La solution de (Datta *et al.*, 2010) stabilise en $O(k)$ rondes en utilisant $O(k \log n)$ bits par processus. Celle de (Datta *et al.*, 2009) stabilise

en $O(n)$ rondes en utilisant $O(\log n)$ bits par processus. L'algorithme proposé dans (Caron *et al.*, 2010) stabilise en $O(k.n)$ rondes en utilisant $O(k \log n)$ bits par processus. Il faut noter que seul l'algorithme de (Datta *et al.*, 2009) calcule un ensemble k-dominant qui soit minimal. En outre, aucune de ces solutions ne garantit d'obtenir des ensembles k-dominants de petite taille. Enfin il existe de nombreuses solutions autostabilisantes pour calculer un ensemble 1-dominant minimal, par exemple (Shukla *et al.*, 1995 ; Ikeda *et al.*, 2002). Cependant, ces solutions sont difficiles à généraliser, en particulier leur généralisation ne donne pas de borne intéressante sur la taille de l'ensemble k-dominant calculé.

Il existe de nombreux algorithmes distribués *non autostabilisants* pour calculer un ensemble k-dominant d'un réseau (Peleg, Upfal, 1989 ; Kutten, Peleg, 1995 ; Amis *et al.*, 2000 ; Fernandess, Malkhi, 2002 ; Ravelomanana, 2005). Les solutions déterministes proposées dans (Amis *et al.*, 2000 ; Fernandess, Malkhi, 2002) sont conçues pour des réseaux *ad hoc mobiles asynchrones*. Les complexités en temps et en espace de la solution de (Amis *et al.*, 2000) sont $O(k)$ et $O(k \log n)$, respectivement. La solution proposée dans (Fernandess, Malkhi, 2002) est un algorithme d'approximation avec un ratio $O(k)$ par rapport à l'optimal. Les complexités en espace et en temps de l'algorithme distribué de (Fernandess, Malkhi, 2002) ne sont pas indiquées. (Peleg, Upfal, 1989) considèrent le problème du calcul déterministe d'un ensemble k-dominant de taille au plus $\lceil \frac{n}{k+1} \rceil$ processus. Leur solution fait l'hypothèse d'un système synchrone et a une complexité en temps de $O(k \log^* n)$ pas de calcul. Cependant, comme nous le verrons dans cet article, les auteurs ont oublié un cas particulier dans leur preuve, la rendant erronée pour certains réseaux. Cette erreur minime est également présente dans plusieurs articles ultérieurs, tels que ceux de (Kutten, Peleg, 1995 ; Penso, Barbosa, 2004). (Ravelomanana, 2005) a proposé un algorithme probabiliste, conçu pour des réseaux UDG synchrones dont la complexité en temps est $O(\mathcal{D})$ rondes, où $\mathcal{D}$ est le diamètre du réseau.

Toutes les solutions non autostabilisantes précédentes peuvent être transformées en solutions autostabilisantes en utilisant un des *transformateurs* existants tels que ceux proposés par (Katz, Perry, 1993 ; Cournier *et al.*, 2003). Cependant, les solutions autostabilisantes ainsi produites sont inefficaces, à la fois en espace et en temps, dû aux mécanismes employés (par exemple, une succession de calcul d'état global du réseau) par ces transformateurs.

1.2. Contributions

Dans cet article, nous proposons un algorithme déterministe, distribué, asynchrone, silencieux et autostabilisant calculant un ensemble k-dominant minimal contenant au plus $\lceil \frac{n}{k+1} \rceil$ processus dans un réseau identifié quelconque.

Nous proposons une correction de la preuve de borne supérieure sur la taille d'un ensemble k-dominant minimum donnée dans (Peleg, Upfal, 1989).

Puis nous proposons un algorithme asynchrone, silencieux et autostabilisant, appelé $\mathcal{SMDS}(k)$ (pour *Small Minimal k-Dominating Set*), calculant un ensemble k-dominant minimal de petite taille, basé sur notre preuve de la borne. Pour simplifier la conception de notre algorithme, nous l'avons écrit comme une composition de trois modules. Les deux premiers modules composés ensemble calculent un ensemble k-dominant D d'au plus $\lceil \frac{n}{k+1} \rceil$ processus. Comme l'ensemble D calculé n'est pas nécessairement minimal, nous appliquons l'algorithme de (Datta *et al.*, 2009) en troisième module, afin de retirer des processus de D jusqu'à obtenir un ensemble k-dominant minimal. La correction de notre méthode est prouvée en supposant un ordonnanceur faiblement équitable. Cette solution stabilise en $O(n)$ rondes et nécessite $O(\log k + \log n + k \log \frac{N}{k})$ bits par processus, où n est la taille du réseau et N est un majorant de n.

Nous proposons ensuite une méthode générale pour transformer *efficacement* tout algorithme autostabilisant sous un ordonnanceur faiblement équitable en algorithme autostabilisant malgré un ordonnanceur inéquitable (la plus faible des hypothèses d'ordonnancement). Le transformateur proposé offre plusieurs avantages par rapport aux solutions précédentes. (1) Il préserve la propriété de silence. (2) Il ne dégrade ni la complexité en rondes ni l'occupation mémoire de l'algorithme en entrée. (3) Il produit un algorithme efficace en termes de pas de calcul ($O(\mathcal{D}n(R + n^2))$, où R est le temps de stabilisation de l'algorithme d'entrée en rondes). Par exemple, en utilisant cette méthode, la version transformée $\mathcal{SMDS}(k)$ stabilise en $O(\mathcal{D}n^3)$ pas de calcul.

Enfin nous avons testé notre algorithme sur une plateforme pour la simulation de réseaux de capteurs sans fil. Les résultats expérimentaux que nous présentons montrent que la taille des ensembles k-dominants obtenus est, en général, beaucoup plus petite que la borne supérieure théorique. En particulier, un gain important en taille est observé après la minimisation effectuée par le troisième et dernier module.

Cet article est organisé comme suit. La section 2 est consacrée aux définitions et au modèle de calcul utilisés dans la suite de cet article. Dans la section 3, nous donnons un contre-exemple de la preuve de borne donnée dans (Peleg, Upfal, 1989) et nous proposons une correction. Nous présentons dans la section 4 une méthode de composition. Cette méthode est utilisée pour construire notre algorithme autostabilisant $\mathcal{SMDS}(k)$ décrit dans la section 5. Dans la section 6, nous montrons comment transformer notre algorithme $\mathcal{SMDS}(k)$ pour obtenir une solution qui fonctionne sous un ordonnanceur inéquitable. La section 7 est dédiée aux résultats de simulation. Enfin nous concluons par des perspectives en section 8.

2. Préliminaires

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 est simplement notée p.

Si b bits sont utilisés pour stocker chaque identité, alors la complexité en mémoire de notre algorithme sera en $\Omega(b)$ par nœud. Dans cet article, tout comme dans une large partie de la littérature, nous supposons que $b = O(\log n)$.

2.1. Modèle de calcul

Nos algorithmes sont écrits dans *le modèle à états* introduit par (Dijkstra, 1974). 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 son *programme* (local). Nous appelons *algorithme* (distribué) $\mathcal{A}$ l'ensemble des programmes exécutés par les processus. Nous notons $\mathcal{A}(p)$ le programme (local) du processus p dans l'algorithme (distribué) $\mathcal{A}$. 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 utilisé uniquement dans les raisonnements. La *garde* d'une action est un prédicat sur 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 dans $\mathcal{A}$. Une configuration du réseau pour l'algorithme $\mathcal{A}$ est une instance des états de tous les processus pour $\mathcal{A}$. L'évaluation des gardes et l'exécution éventuelle des actions sont supposées atomiques (on parle d'atomicité en lecture et écriture (Dolev, 2000)).

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 plus aucun processus n'est activable. Lors de chaque pas de calcul $\gamma_i \mapsto \gamma_{i+1}$, un à plusieurs processus peuvent exécuter une de leurs actions activables.

Nous supposons que chaque pas de calcul est dirigé par un *ordonnanceur*. Si un ou plusieurs processus sont activables, l'ordonnanceur sélectionne au moins un de ces processus pour exécuter une de ses actions activables. Un ordonnanceur est caractérisé par son *équité*. Un ordonnanceur est *faiblement équitable* s'il permet à tout processus *continûment* activable d'exécuter une action en un temps fini. L'hypothèse d'équité la plus faible est celle d'*inéquité* : un ordonnanceur inéquitable peut empêcher un processus d'exécuter une action activable tant que ce processus n'est pas le seul activable.

Un processus p subit une *neutralisation* lors d'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.

Pour évaluer la complexité en temps, nous utilisons la notion de *ronde*. La première ronde d'une exécution e, notée e', correspond au plus petit préfixe de e dans lequel chaque processus activable dans la configuration initiale exécute une action ou est neutralisé. Soit e'' un suffixe de e commençant par la dernière configuration de e'. La deuxième ronde de e est la première ronde de e'', la troisième ronde de e est la deuxième ronde de e'', etc.

2.2. Autostabilisation et silence

Soit $\mathcal{A}$ un algorithme distribué et P un prédicat sur les configurations de $\mathcal{A}$. $\mathcal{A}$ *stabilise à* P s'il existe un sous-ensemble non vide $\mathcal{S}$ de configurations de $\mathcal{A}$ tel que :

Correction. $\forall \gamma \in \mathcal{S}, P(\gamma)$;

Fermeture. Pour tout pas de calcul $\gamma \mapsto \gamma'$ de $\mathcal{A}$, si $\gamma \in \mathcal{S}$ alors $\gamma' \in \mathcal{S}$;

Convergence. Toute exécution de $\mathcal{A}$ contient une configuration de $\mathcal{S}$. [1]

Les configurations de $\mathcal{S}$ sont alors dites *légitimes*. *A contrario,* les autres sont dites *illégitimes*.

Un algorithme est dit *silencieux* (Dolev *et al.*, 1996) si chacune de ses exécutions est finie. En d'autres termes, à partir d'une configuration quelconque, le réseau atteindra en temps fini une configuration terminale.

Nous nous intéressons en particulier aux algorithmes autostabilisants silencieux. Pour prouver la stabilisation vers un prédicat P de ce type d'algorithmes, il suffit de démontrer que (1) toutes les exécutions de l'algorithme sont finies et que (2) P est vérifié dans toute configuration terminale (la propriété de fermeture étant trivialement vérifiée dans une configuration terminale).

3. Borne

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 (Peleg, Upfal, 1989). Cependant, la preuve qui y est proposée contient une erreur minime. Celle-ci est également présente dans des travaux ultérieurs, par exemple (Penso, Barbosa, 2004). Ci-après, 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 en 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 r est

1. Nous rappelons que toute configuration initiale d'une exécution est quelconque.

un processus distingué appelé *racine*. 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 $G = (V, E)$ de hauteur h enraciné en r. 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 $D_i = \bigcup_{j \geq 0} T_{i+j(k+1)}$. Notez que les ensembles D_i sont disjoints. De plus, chaque D_i contient l'ensemble des nœuds dont la hauteur modulo $k+1$ est égale à i.

Dans le cas où $k < h$, la preuve de (Peleg, Upfal, 1989) 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. Le problème en question n'apparaît que lorsque le plus petit des ensembles D_i ($i \in [0..k]$), appelons le D_ℓ, n'est pas D_0. Dans ce cas, il est possible qu'un processus feuille dont la hauteur est strictement inférieure à ℓ ne soit k-dominé par aucun processus de D_ℓ, comme le montre l'exemple précédent. Pour corriger ce problème, 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 ($\frac{n}{k+1}$), nous posons $D = D_0$. Dans tous les autres cas, la taille du plus petit des D_i ($i \in [0..k]$), disons $D_{\min}$, est strictement inférieure à $\lceil \frac{n}{k+1} \rceil$ et nous posons $D = D_{\min} \cup \{r\}$. Dans les deux cas, D est un ensemble k-dominant du réseau comme le montre la preuve suivante.

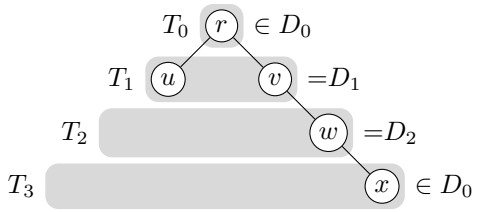


Figure 2. Contre-exemple de la preuve originale

THÉORÈME 1. — *Soit $G = (V, E)$ 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. Considérons les ensembles $D_0, \ldots, D_k$ définis précédemment.

Si $k \geq h$, 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 maintenant que $k < h$. Alors, $\forall i \in [0..k]$, $|D_i| > 0$.

1. Supposons que $\forall i \in [0..k-1]$, $|D_i| = |D_{i+1}|$. Alors, $\forall i \in [0..k]$, $|D_i| = \frac{n}{k+1}$. Considérons maintenant un processus $v \notin D_0$. La hauteur de v, $h(v)$, satisfait $h(v) = x(k+1) + y$ avec $x \geq 0$ et $0 < y \leq k$. Soit u l'ancêtre de v tel que $h(u) = h(v) - y$ (un tel processus existe car $y \leq h(v)$). Par définition, $u \in D_0$ et $\|u, v\| \leq k$. Par conséquent, D_0 est un ensemble k-dominant tel que $|D_0| = \lceil \frac{n}{k+1} \rceil$.

2. 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 processus $v \notin D$.

a) Si $h(v) \leq k$, alors v est à distance au plus k de r et $r \in D$.

b) Si $h(v) > k$, alors $h(v) = x(k+1) + y$ avec $x > 0$, $0 \leq y \leq k$ et $y \neq j$. Si $y > j$, alors soit u l'ancêtre de v tel que $h(u) = x(k+1) + j$. Si $y < j$, soit u l'ancêtre de v tel que $h(u) = (x-1)(k+1) + j$. Par définition, $u \in D$ (plus précisément $u \in D_j$) et $\|u, v\| \leq k$.

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* définie dans (Herman, 1992 ; Tel, 2001). Lorsque nous composons collatéralement deux algorithmes (distribués) $\mathcal{A}$ et $\mathcal{B}$, chaque processus p exécute ses programmes (locaux) $\mathcal{A}(p)$ et $\mathcal{B}(p)$ concurremment et $\mathcal{B}(p)$ utilise la sortie de $\mathcal{A}(p)$ dans ses calculs. Dans notre variante, nous modifions le code de $\mathcal{B}(p)$ de sorte qu'un processus ne puisse exécuter une action de $\mathcal{B}(p)$ que si aucune action de $\mathcal{A}(p)$ n'est activable.

DÉFINITION 2. — *Soit $\mathcal{A}$ et $\mathcal{B}$ deux algorithmes tels qu'aucune variable écrite par $\mathcal{B}$ n'apparaît dans $\mathcal{A}$. Dans la* composition collatérale hiérarchique *de $\mathcal{A}$ et $\mathcal{B}$, notée $\mathcal{B} \circ \mathcal{A}$, le programme local de chaque processus p, $\mathcal{B}(p) \circ \mathcal{A}(p)$, est défini comme suit :*

– *$\mathcal{B}(p) \circ \mathcal{A}(p)$ contient toutes les variables de $\mathcal{A}(p)$ et $\mathcal{B}(p)$;*

– *$\mathcal{B}(p) \circ \mathcal{A}(p)$ contient toutes les actions de $\mathcal{A}(p)$;*

– *pour toute action $G_i \rightarrow S_i$ de $\mathcal{B}(p)$, $\mathcal{B}(p) \circ \mathcal{A}(p)$ contient l'action $\neg C_p \wedge G_i \rightarrow S_i$ où C_p est la disjonction de toutes les gardes des actions de $\mathcal{A}(p)$.*

Ci-après, nous présentons deux propriétés de la *composition collatérale hiérarchique* : le théorème 5 et le corollaire 6. Ces propriétés énoncent une condition suffisante pour montrer la correction de notre algorithme présenté en section 5. Afin de

prouver ces propriétés, nous définissons au préalable les notions de *sous-séquence utile minimale* (SUM) et de *projection*.

DÉFINITION 3 (SUM). — *Soit s une séquence de configurations. La* sous-séquence utile minimale *de s, notée $\mathcal{SUM}(s)$, est la sous-séquence maximale de s où il n'y a pas deux configurations consécutives identiques.*

DÉFINITION 4 (Projection). — *Soit γ une configuration et $\mathcal{A}$ un algorithme. La* projection $\gamma_{|\mathcal{A}}$ *est la configuration obtenue en supprimant de γ les valeurs de toutes les variables qui n'existent pas dans $\mathcal{A}$. Soit $e = \gamma_0 \dots \gamma_i$ une séquence de configurations, la* projection $e_{|\mathcal{A}}$ *est la séquence $\gamma_{0|\mathcal{A}} \dots \gamma_{i|\mathcal{A}}$.*

THÉORÈME 5. — *Soit $\mathcal{A}$ un algorithme silencieux qui stabilise à $SP_{\mathcal{A}}$ en supposant un ordonnanceur faiblement équitable. Soit $\mathcal{B}$ un algorithme tel qu'aucune variable écrite par $\mathcal{B}$ n'apparaît dans $\mathcal{A}$. $\mathcal{B} \circ \mathcal{A}$ satisfait les deux affirmations suivantes :*

– $\mathcal{B} \circ \mathcal{A}$ stabilise à $SP_{\mathcal{A}}$ en supposant un ordonnanceur faiblement équitable ;

– $\mathcal{B} \circ \mathcal{A}$ atteint en temps fini une configuration à partir de laquelle les valeurs de toutes les variables écrites par $\mathcal{A}$ sont figées.

PREUVE. — Soit une exécution e de $\mathcal{B} \circ \mathcal{A}$ en supposant un ordonnanceur faiblement équitable. Soit $e' = \mathcal{SUM}(e_{|\mathcal{A}})$. Dans les configurations de e', aucune variable n'est écrite par $\mathcal{B}$. Toutes les configurations de e' sont des configurations possibles de $\mathcal{A}$.

Considérons tout processus p continûment activable pour l'algorithme $\mathcal{A}$ dans une configuration γ de e'. Notons que p est continûment activable dès la première configuration de e qui génère γ. Donc, en temps fini, p exécutera une action de $\mathcal{A}$ dans e et ainsi dans e'. Donc e' est une exécution possible de $\mathcal{A}$ en supposant un ordonnanceur faiblement équitable. Par conséquent e' stabilise à $SP_{\mathcal{A}}$ et compte un nombre fini de configurations. Enfin e stabilise à $SP_{\mathcal{A}}$ et atteint en temps fini une configuration à partir de laquelle toutes les variables écrites par $\mathcal{A}$ sont figées. ■

Du théorème précédent, nous déduisons directement le corollaire suivant :

COROLLAIRE 6. — *$\mathcal{B} \circ \mathcal{A}$ stabilise à SP en supposant un ordonnanceur faiblement équitable si :*

– $\mathcal{A}$ est un algorithme (autostabilisant) silencieux 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 en supposant un ordonnanceur faiblement équitable. [2]

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

Dans cette section, nous présentons un algorithme autostabilisant silencieux, appelé $\mathcal{SMDS}(k)$ pour *Small Minimal k-Dominating Set*, qui calcule un ensemble

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

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 trois algorithmes autostabilisants silencieux, $\mathcal{SMDS}(k) = \mathcal{MIN}(k) \circ \mathcal{DS}(k) \circ \mathcal{ST}$, où :

– $\mathcal{ST}$ est un algorithme qui construit un arbre couvrant enraciné ;

– $\mathcal{DS}(k)$ calcule un ensemble k-dominant d'au plus $\lceil \frac{n}{k+1} \rceil$ processus utilisant l'arbre couvrant construit par $\mathcal{ST}$;

– $\mathcal{MIN}(k)$ rend minimal l'ensemble k-dominant calculé par $\mathcal{DS}(k)$.

Nous détaillons chacun de ces trois modules dans les sous-sections 5.1 à 5.3. Après avoir prouvé la correction globale de $\mathcal{SMDS}(k)$, nous analysons sa complexité dans la section 5.4.

5.1. Module $\mathcal{ST}$

$\mathcal{ST}$ est un algorithme autostabilisant silencieux construisant un arbre couvrant dans un réseau identifié quelconque. Cet arbre est enraciné, c'est-à-dire qu'un des nœuds de l'arbre, noté r, est distingué comme racine et que tous les autres distinguent comme « père » leur voisin le plus proche de r dans l'arbre. Dans le reste de cet article, nous supposons 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, $\mathcal{ST}$ stabilise au prédicat $SP_{\mathcal{ST}}$ défini comme suit : $SP_{\mathcal{ST}}$ est vrai si et seulement si (1) la configuration est terminale, (2) il existe un unique processus r tel que $\texttt{Parent}_r = \perp$ et (3) le graphe $T = (V, E_T)$ où $E_T = \{\{p, \texttt{Parent}_p\}, \forall p \in V \setminus \{r\}\}$ est un arbre couvrant, c'est-à-dire un sous-graphe induit par V connexe et sans cycle.

L'algorithme autostabilisant silencieux pour des réseaux identifiés de (Datta *et al.*, 2011) peut être utilisé comme algorithme $\mathcal{ST}$. Bien que ce soit un algorithme d'élection, cet algorithme, comme de nombreux autres algorithmes autostabilisant d'élection, construit aussi un arbre couvrant enraciné. Notez que cet algorithme fonctionne sous un ordonnanceur inéquitable, stabilise en $O(n)$ rondes, nécessite $O(\log n)$ bits par processus et ne requiert aucune connaissance *a priori* d'un majorant de n ou de $\mathcal{D}$. D'après (Datta *et al.*, 2011), nous avons :

LEMME 7. — *$\mathcal{ST}$ est un algorithme silencieux qui stabilise à $SP_{\mathcal{ST}}$ en supposant un ordonnanceur faiblement équitable.*

5.2. Module $\mathcal{DS}(k)$

$\mathcal{DS}(k)$ (voir l'algorithme 1 pour une description formelle) est aussi silencieux et 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, chaque processus p maintient les trois variables suivantes dans $\mathcal{DS}(k)$:

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

Entrées :

$\texttt{Parent}_p \in \mathcal{N}_p \cup \{\perp\}$	Processus parent de p dans l'arbre couvrant, ou $\perp$ si p est la racine.

Variables :

$p.couleur \in [0..k]$	Couleur de p.
$p.pop[i] \in \mathbb{N}, \forall i \in [0..k]$	Population de couleur i, dans le sous-arbre enraciné à p.
$p.min \in [0..k]$	Couleur de plus petite population, dans V.

Macros :

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

Prédicats :

$CouleurAssortie_p$	$\equiv$	$p.couleur = \texttt{EvaluerCouleur}_p$
$ComptesRonds_p$	$\equiv$	$\forall i \in [0..k], p.pop[i] = \texttt{EvaluerPop}_p(i)$
$MinChoisi_p$	$\equiv$	$p.min = \texttt{EvaluerMin}_p$
$EstDominant_p$	$\equiv$	$\texttt{Parent}_p = \perp \vee p.couleur = p.min$

Actions :

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

– $p.couleur \in [0..k]$. Dans cette variable, p calcule $h(p) \bmod (k+1)$ (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 tel que 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 $p.pop[i]$, avec $i \in [0..k]$, 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 tel que $p = r$ ou $p.couleur = p.min$, *i.e.* l'ensemble $\{p \in V \mid EstDominant_p\}$, est un ensemble k-dominant d'au plus $\lceil \frac{n}{k+1} \rceil$ processus. Ainsi, $\mathcal{DS}(k) \circ \mathcal{ST}$ stabilise au prédicat $SP_{\mathcal{DS}(k)}$ défini comme suit : $SP_{\mathcal{DS}(k)}$ est vrai si et seulement si la configuration est terminale et l'ensemble $\{p \in V \mid EstDominant_p\}$ est un ensemble k-dominant d'au plus $\lceil \frac{n}{k+1} \rceil$ processus.

Nous prouvons maintenant la correction de $\mathcal{DS}(k)$. Dans les preuves suivantes, nous étudions le comportement de notre algorithme à partir d'une configuration où aucune action de $\mathcal{ST}$ n'est activable. Comme $\mathcal{DS}(k)$ n'écrit pas dans les variables de $\mathcal{ST}$, toutes les variables de $\mathcal{ST}$ sont figées pour toujours à partir d'une telle configuration. De plus, un arbre couvrant est défini (en utilisant l'entrée $\texttt{Parent}_p$ pour chaque processus p), d'après le lemme 7. Notons T cet arbre couvrant et r sa racine.

LEMME 8. — *À partir de toute configuration où aucune action de $\mathcal{ST}$ n'est activable, la variable $p.couleur$ de tout processus p est égale pour toujours à $h(p) \bmod (k+1)$ après au plus n rondes.*

PREUVE. — Remarquons d'abord que :

(a) Pour tout processus p, l'action `Colorer`, dont la garde est $\neg CouleurAssortie_p$, est la seule action de p qui modifie $p.couleur$.

Nous prouvons ce lemme par récurrence sur la hauteur des processus dans T.

Soit γ une configuration où aucune action de $\mathcal{ST}$ n'est activable.

Cas de base. Considérons la racine r (le seul processus de hauteur 0).

(b) Le prédicat $CouleurAssortie_r$ ne dépend que de la variable $r.couleur$ et de l'entrée $\texttt{Parent}_r$ qui vaut $\perp$ pour toujours depuis la configuration γ.

Supposons que $CouleurAssortie_r$ soit vrai dans γ. Alors, $r.couleur = 0$. De plus, par (a) et (b), $CouleurAssortie_r$ est vrai pour toujours et, par conséquent, $r.couleur = 0$ est vrai pour toujours.

Supposons au contraire que $CouleurAssortie_r$ ne soit pas vrai dans γ. Alors, par (a) et (b), l'action `Colorer` est continûment activable pour r. Comme l'ordonnanceur est faiblement équitable, l'action `Colorer` est exécutée par r en au plus 1 ronde. Donc, après au plus 1 ronde, $CouleurAssortie_r$ devient vrai et nous pouvons alors nous ramener au cas précédent.

Hypothèse. Soit $j \in \mathbb{N}^*$. Supposons que, pour tout processus p tel que $h(p) < j$, la variable $p.couleur$ est égale pour toujours à $h(p) \bmod (k+1)$ après au plus $h(p)+1$ rondes depuis γ.

Récurrence. Considérons tout processus p tel que $h(p) = j$.

(c) Le prédicat $CouleurAssortie_p$ ne dépend que de la variable $p.couleur$, de l'entrée $\texttt{Parent}_p$ qui est égale pour toujours à une valeur dans $\mathcal{N}_p$ à partir de

γ, et de $\texttt{Parent}_p.couleur$ qui est égale pour toujours à $h(\texttt{Parent}_p) \bmod (k+1)$ après au plus $h(p)$ rondes depuis γ par hypothèse de récurrence.

Supposons que $CouleurAssortie_p$ soit vrai après $h(p)$ rondes depuis γ. Alors, $p.couleur = (\texttt{Parent}_p.couleur + 1) \bmod (k+1) = (h(\texttt{Parent}_p) \bmod (k+1) + 1) \bmod (k+1) = h(p) \bmod (k+1)$. De plus, par (a) et (c), $CouleurAssortie_p$ est vrai pour toujours et, par conséquent, $p.couleur = h(p) \bmod (k+1)$ est vrai pour toujours.

Supposons au contraire que $CouleurAssortie_p$ ne soit pas vrai après $h(p)$ rondes depuis γ. Alors, par (a) et (c), l'action `Colorer` est continûment activable pour p depuis γ. Comme l'ordonnanceur est faiblement équitable, l'action `Colorer` est exécutée par p en au plus 1 ronde supplémentaire. Donc, en au plus $h(p)+1$ rondes depuis γ, $CouleurAssortie_p$ devient vrai et nous pouvons alors nous ramener au cas précédent.

Comme la hauteur de T est bornée par $n-1$, le lemme est vérifié. ■

LEMME 9. — *À partir de toute configuration où :*

– *aucune action de $\mathcal{ST}$ n'est activable, et*

– *la variable $q.couleur$ de tout processus q est égale pour toujours à $h(q) \bmod (k+1)$,*

pour tout processus p et tout indice $i \in [0..k]$, la variable $p.pop[i]$ est égale pour toujours à $|\{q \in T(p) \mid q.couleur = i\}|$ en au plus n rondes.

PREUVE. — Remarquons d'abord que :

(a) Pour tout processus p, l'action `Compter`, dont la garde est $CouleurAssortie_p \wedge \neg ComptesRonds_p$, est la seule action de p qui modifie $p.pop$.

Soit γ une configuration où :

– aucune action de $\mathcal{ST}$ n'est activable, et

– la variable $q.couleur$ de tout processus q vaut $h(q) \bmod (k+1)$ pour toujours.

Remarquons que :

(b) À partir de γ, pour tout processus p, $CouleurAssortie_p$ est vrai pour toujours et, par conséquent, l'action `Compter` est activable pour p si et seulement si $\neg ComptesRonds_p$ est vrai.

Nous prouvons ce lemme par récurrence sur $h(T(p))$ pour tout processus p.

Cas de base. Considérons tout processus p tel que $h(T(p)) = 0$ (p est un processus feuille).

(c) Le prédicat $ComptesRonds_p$ ne dépend que des variables $p.pop$ et $p.couleur$, cette dernière étant égale pour toujours à $h(p) \bmod (k+1)$.

Supposons que $ComptesRonds_p$ soit vrai dans γ. Alors, $\forall i \in [0..k], p.pop[i] = \texttt{SelfPop}_p(i) = |\{q \in T(p) \mid q.couleur = i\}|$. De plus, par (a) et (c),

$ComptesRonds_p$ est vrai pour toujours et, par conséquent, $\forall i \in [0..k]$, $p.pop[i] = |\{q \in T(p) \mid q.couleur = i\}|$ est vrai pour toujours.

Supposons au contraire que $ComptesRonds_p$ soit faux dans γ. Alors par (a), (b) et (c), l'action `Compter` est continûment activable. Comme l'ordonnanceur est faiblement équitable, l'action `Compter` est exécutée par p en au plus 1 ronde depuis γ. Alors, $ComptesRonds_p$ devient vrai et nous pouvons nous ramener au cas précédent.

Hypothèse. Soit $j \in \mathbb{N}^*$. Supposons que pour tout processus p tel que $h(T(p)) < j$ et pour tout indice $i \in [0..k]$, la variable $p.pop[i]$ est égale à $|\{q \in T(p) \mid q.couleur = i\}|$ après au plus $h(T(p)) + 1$ rondes depuis γ.

Récurrence. Considérons tout processus p tel que $h(T(p)) = j$.

(d) Le prédicat $ComptesRonds_p$ ne dépend que des variables $p.pop$, $p.couleur$ (qui est figée par hypothèse), et $q.pop$ de chaque fils q de p dans T. Ces dernières variables sont figées après $h(T(p))$ rondes depuis γ par hypothèse de récurrence.

Supposons que $ComptesRonds_p$ est vrai après $h(T(p))$ rondes depuis γ. Alors, $\forall i \in [0..k], \texttt{SelfPop}_p(i) = true$, c'est-à-dire, $p.pop[i] = \texttt{SelfPop}_p(i) + \sum_{q \in \texttt{Fils}_p} |\{q' \in T(q) \mid q'.couleur = i\}| = |\{q \in T(p) \mid q.couleur = i\}|$, par hypothèse de récurrence. De plus, par (a), (b) et (d), $ComptesRonds_p$ est vrai pour toujours et, par conséquent, $\forall i \in [0..k], p.pop[i] = |\{q \in T(p) \mid q.couleur = i\}|$ est vrai pour toujours.

Supposons au contraire que $ComptesRonds_p$ est faux après $h(T(p))$ rondes depuis γ. Alors, par (a), (b) et (d), l'action `Compter` est continûment activable pour p. Comme l'ordonnanceur est faiblement équitable, l'action `Compter` est exécutée par p en au plus 1 ronde. Donc, en au plus $h(T(p)) + 1$ rondes, $ComptesRonds_p$ devient vrai et nous pouvons nous ramener au cas précédent.

Comme la hauteur de T est bornée par $n - 1$, le lemme est vérifié. ■

La preuve du lemme suivant suit le même schéma que la preuve du lemme 8.

LEMME 10. — *À partir de toute configuration où :*

– *aucune action de $\mathcal{ST}$ n'est activable,*

– *la variable $q.couleur$ de tout processus q est égale pour toujours à $h(q) \bmod (k+1)$, et*

– *pour tout processus p et tout indice $i \in [0..k]$, la variable $p.pop[i]$ est égale pour toujours à $|\{q \in T(p) \mid q.couleur = i\}|$,*

en au plus n rondes, la variable $p.min$ de tout processus p est égale pour toujours au plus petit indice $i_{min} \in [0..k]$ qui satisfait $|C_{i_{min}}| = \min_{j \in [0..k] \mid C_j \neq \emptyset} |C_j|$ où pour tout $j \in [0..k]$, $C_j = \{q \in T \mid q.couleur = j\}$.

D'après les lemmes 8 à 10, nous déduisons le théorème suivant :

THÉORÈME 11. — *À partir de toute configuration où aucune action de $\mathcal{ST}$ n'est activable, $\mathcal{DS}(k) \circ \mathcal{ST}$ atteint en au plus $3n$ rondes une configuration terminale où*

pour tout processus p :

(a) $p.couleur = h(p) \bmod (k+1)$, *et*

(b) pour tout indice $i \in [0..k]$, *la variable* $p.pop[i]$ *est égale pour toujours à* $|\{q \in T(p) \mid q.couleur = i\}|$,

(c) $p.min = i_{min}$ *où* i_{min} *est le plus petit indice dans* $[0..k]$ *qui satisfait* $|C_{i_{min}}| = \min_{j\in[0..k] \mid C_j \neq \emptyset} |C_j|$ *où pour tout* $j \in [0..k]$, $C_j = \{q \in T \mid q.couleur = j\}$.

Nous considérons désormais toute configuration terminale γ_t de $\mathcal{DS}(k) \circ \mathcal{ST}$ (une telle configuration existe d'après le corollaire 6, le lemme 7 et le théorème 11). Soit c_t l'unique valeur des variables min dans γ_t (c_t est bien définie d'après le théorème 11). Dans γ_t, la sortie de $\mathcal{DS}(k) \circ \mathcal{ST}$ est l'ensemble $DS^{out} = \{p \in V \mid EstDominant_p\}$.

D'après le théorème 11 et la définition du prédicat $EstDominant_p$, nous pouvons déduire le lemme suivant :

LEMME 12. — *Dans* γ_t, $DS^{out}=\{r\} \cup DS^{c_t}$ *où* $DS^{c_t}=\{p \in V \mid h(p) \bmod (k+1)=c_t\}$.

Nous montrons maintenant que, dans tous les cas, DS^{out} est le même ensemble que celui obtenu en appliquant la construction donnée dans la preuve du théorème 1. Pour cela, nous réutilisons les ensembles $D_0, \ldots, D_k$ définis dans la section 3.

REMARQUE 13. — $DS^{c_t} = D_{c_t}$. □

THÉORÈME 14. — *Dans* γ_t, DS^{out} *est un ensemble* k*-dominant de* G *tel que* $|DS^{out}| \leq \lceil \frac{n}{k+1} \rceil$.

PREUVE. — Considérons les trois cas suivants :

– $k \geq h(T)$. Dans ce cas, la preuve du théorème 1 énonce que D_0 est un ensemble k-dominant de taille au plus $\lceil \frac{n}{k+1} \rceil$. Par le théorème 11.(c), c_t est le plus petit indice dans $[0..k]$ qui satisfait $|C_{c_t}| = \min_{j\in[0..k] \mid C_j \neq \emptyset} |C_j|$ où pour tout $j \in [0..k]$, $C_j = \{q \in T \mid q.couleur = j\}$. De plus, d'après le théorème 11.(a), pour tout $j \in [0..k]$, $C_j = D_j$. Donc, c_t est le plus petit indice dans $[0..k]$ qui satisfait $|D_{c_t}| = \min_{j\in[0..k] \mid D_j \neq \emptyset} |D_j|$. Par définition, $\min_{j\in[0..k] \mid D_j \neq \emptyset} |D_j| \geq 1$. Comme $k \geq h(T)$, $D_0 = \{r\}$, *i.e.*, $|D_0| = 1$ et $c_t = 0$. Donc, $DS^{c_t} = D_0$ d'après la remarque 13 et $DS^{out} = \{r\} \cup D_0 = D_0$, et le théorème est vérifié dans ce cas.

– $k < h(T)$ *et pour tout* $i \in [0..k-1]$, $|D_i| = |D_{i+1}|$. Par une preuve similaire au cas précédent nous prouvons que le théorème est aussi vérifié dans ce cas.

– $k < h(T)$ *et il existe* $i \in [0..k-1]$ *tel que* $|D_i| \neq |D_{i+1}|$. Soit i_{min} le plus petit indice tel que $|D_{i_{min}}| = \min_{j\in[0..k] \mid D_j \neq \emptyset} |D_j|$. Dans ce cas, la preuve du théorème 1 énonce que $\{r\} \cup D_{i_{min}}$ est un ensemble k-dominant d'au plus $\lceil \frac{n}{k+1} \rceil$ processus. D'après le théorème 11.(c), c_t est le plus petit indice dans $[0..k]$ qui satisfait $|C_{c_t}| = \min_{j\in[0..k] \mid C_j \neq \emptyset} |C_j|$ où pour tout $j \in [0..k]$, $C_j = \{q \in T \mid q.couleur = j\}$. De plus, par le théorème 11.(a), pour tout $j \in [0..k]$, $C_j = D_j$. Donc, c_t est le plus petit indice dans $[0..k]$ qui satisfait $|D_{c_t}| = \min_{j\in[0..k] \mid D_j \neq \emptyset} |D_j|$. Par conséquent, $c_t = i_{min}$, $DS^{c_t} = D_{i_{min}}$ d'après la remarque 13, $DS^{out} = \{r\} \cup D_{i_{min}}$, et le théorème est vérifié dans ce cas.

■

D'après les théorèmes 11 et 14, nous pouvons déduire le théorème suivant :

THÉORÈME 15. — *À partir de toute configuration où aucune action de $\mathcal{ST}$ n'est activable, l'algorithme $\mathcal{DS}(k)$ atteint en au plus $3n$ rondes une configuration (terminale) satisfaisant $SP_{\mathcal{DS}(k)}$.*

D'après le corollaire 6, le lemme 7 et le théorème 15, nous pouvons déduire le théorème suivant :

THÉORÈME 16. — *$\mathcal{DS}(k) \circ \mathcal{ST}$ 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 $\mathcal{DS}(2) \circ \mathcal{ST}$. 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 la stabilisation de $\mathcal{DS}(2) \circ \mathcal{ST}$, $r.pop[0] = 5$, $r.pop[1] = 5$ et $r.pop[2] = 3$. Ainsi, $r.min = 2$, ce qui signifie que 2 est la couleur utilisée le moins souvent : $|D_2| = |\{p_4, p_9, p_{10}\}| = 3$. Dans ce cas, l'ensemble 2-dominant en sortie de $\mathcal{DS}(2) \circ \mathcal{ST}$ est $SD = \{r\} \cup D_2$, c'est-à-dire $\{r, p_4, p_9, p_{10}\}$. Cet ensemble 2-dominant vérifie 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, car aucun de ses sous-ensembles n'est 2-dominant.

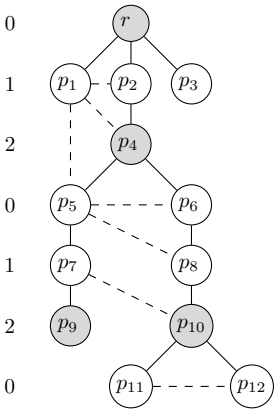


Figure 3. Ensemble calculé par $\mathcal{DS}(2)$ (nœuds gris)

5.3. Module $\mathcal{MIN}(k)$

$\mathcal{MIN}(k)$ est aussi silencieux et calcule un ensemble k-dominant minimal qui est un sous-ensemble de l'ensemble k-dominant calculé par $\mathcal{DS}(k)$. En section 7, nous verrons que cette minimisation apporte généralement un gain non négligeable.

Ce dernier module de notre algorithme peut être réalisé en utilisant l'algorithme autostabilisant silencieux $\mathcal{MIN}(k)$ de (Datta *et al.*, 2009). Cet algorithme prend un ensemble k-dominant I en entrée, et construit un sous-ensemble de I qui est k-dominant minimal. La connaissance de I est répartie, c'est-à-dire que chaque processus p n'utilise que son entrée $EstDominant_p$ 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 temps fini, $\{p \in V \mid p.dansD\}$ est un ensemble k-dominant minimal.

En utilisant la sortie de l'algorithme $\mathcal{DS}(k) \circ \mathcal{ST}$ comme entrée pour l'algorithme $\mathcal{MIN}(k)$, la taille de l'ensemble k-dominant minimal ainsi obtenu est tou-

jours bornée par $\lceil \frac{n}{k+1} \rceil$, car $\mathcal{MIN}(k)$ ne fait que retirer des membres de l'ensemble k-dominant calculé par $\mathcal{DS}(k)$. Ainsi, $\mathcal{SMDS}(k) = \mathcal{MIN}(k) \circ \mathcal{DS}(k) \circ \mathcal{ST}$ stabilise au prédicat $SP_{\mathcal{SMDS}(k)}$ défini comme suit : $SP_{\mathcal{SMDS}(k)}$ est vrai si et seulement si la configuration est terminale et l'ensemble $\{p \in V \mid p.dansD\}$ est k-dominant minimal de taille au plus $\lceil \frac{n}{k+1} \rceil$.

Comme $\mathcal{SMDS}(k) = \mathcal{MIN}(k) \circ \mathcal{DS}(k) \circ \mathcal{ST}$, d'après le corollaire 6 et le théorème 16, nous pouvons déduire le résultat suivant :

THÉORÈME 17 (Correction globale). — *$\mathcal{SMDS}(k)$ stabilise à $SP_{\mathcal{SMDS}(k)}$ en supposant un ordonnanceur faiblement équitable.*

5.4. Complexité de $\mathcal{SMDS}(k)$

Considérons d'abord la complexité en rondes de $\mathcal{SMDS}(k)$. En utilisant l'algorithme de (Datta *et al.*, 2011), le module $\mathcal{ST}$ stabilise en $O(n)$ rondes. Une fois que l'arbre couvrant est construit, $\mathcal{DS}(k)$ stabilise en $O(n)$ rondes, comme nous l'avons montré précédemment, par le théorème 16. Pour finir, l'ensemble k-dominant calculé par les trois premiers modules est rendu minimal par $\mathcal{MIN}(k)$ en $O(n)$ rondes, d'après (Datta *et al.*, 2009). D'où le théorème suivant.

THÉORÈME 18. — *$\mathcal{SMDS}(k)$ stabilise à $SP_{\mathcal{SMDS}(k)}$ en $O(n)$ rondes.*

Considérons la complexité en espace de $\mathcal{SMDS}(k)$. $\mathcal{ST}$ et $\mathcal{MIN}(k)$ peuvent être implémentés avec $O(\log n)$ bits par processus (Datta *et al.*, 2011 ; Huang, Chen, 1992 ; Datta *et al.*, 2009). 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 d'un majorant. Cependant, n peut être calculé dynamiquement à partir de l'arbre couvrant.

THÉORÈME 19. — *$\mathcal{SMDS}(k)$ peut être implémenté en utilisant $O(\log k + \log n + k \log \frac{N}{k})$ bits par processus où N est un majorant de n.*

6. Transformateur

Dans la section précédente nous avons montré que $\mathcal{SMDS}(k)$ stabilise au prédicat $SP_{\mathcal{SMDS}(k)}$ en supposant un ordonnanceur faiblement équitable. Nous proposons maintenant une méthode automatique pour transformer tout algorithme autostabilisant supposant un ordonnanceur faiblement équitable en algorithme autostabilisant supposant un ordonnanceur inéquitable (pour la même spécification).

Parmi les méthodes similaires préexistantes, la *composition croisée* de (Beauquier *et al.*, 2001) ne préserve pas le silence de l'algorithme en entrée et la complexité de l'algorithme transformé n'a pas, à notre connaissance, été analysée. (Kosowski, Kuszner, 2006) proposent un transformateur qui préserve le silence, mais pour lequel l'algorithme transformé stabilise en $O(n^4 \times R)$ pas de calcul, où R est le temps de stabilisation de l'algorithme en entrée en rondes. Enfin, il faut remarquer que la complexité en rondes de l'algorithme transformé est beaucoup plus grande que celle de l'algorithme en entrée (du même ordre que la complexité en pas de calcul).

À la différence des approches précédentes, notre méthode, à la fois, préserve le silence (éventuel) de l'algorithme en entrée, et ne dégrade pas la complexité en rondes de l'algorithme en entrée. De plus, la complexité en pas de calcul de l'algorithme transformé est $O(\mathcal{D}n(R + n^2))$ où R est le temps de stabilisation de l'algorithme en entrée en rondes.

Soit $\mathcal{A}$ un algorithme distribué qui stabilise à $SP_{\mathcal{A}}$ en supposant un ordonnanceur faiblement équitable [3]. Soit p un processus. Nous rappelons que nous notons $\mathcal{A}(p)$ le programme local de p dans $\mathcal{A}$. $\mathcal{A}(p)$ a x actions, indicées de 0 jusqu'à $x - 1$. Ces actions sont de la forme suivante : $A_i :: G_i \longrightarrow S_i$. Nous notons $\mathcal{A}^t$ la version transformée de $\mathcal{A}$. Concrètement, $\mathcal{A}^t$ est obtenu en composant $\mathcal{A}$ avec une horloge de phase autostabilisante. Cette dernière est considérée comme une boîte noire, appelée $\mathcal{U}$ ($\mathcal{U}(p)$ dénote le programme local du processus p dans $\mathcal{U}$), ayant les propriétés suivantes :

– chaque processus p a une variable compteur $p.temps$, membre d'un groupe cyclique $\mathbb{Z}_\alpha$ où α est un entier positif ;

– l'horloge de phase est autostabilisante en supposant un ordonnanceur inéquitable, c'est-à-dire qu'une fois stabilisée, il existe une fonction entière f, pour chaque processus, telle que :

- $f(p) \bmod \alpha = p.temps$;

- pour tout processus p et q, $|f(p) - f(q)| \leq \|p, q\|$;

- pour tout processus p, $f(p)$ est incrémenté de 1 infiniment souvent par l'instruction $Incr_p$;

– dans chaque programme local $\mathcal{U}(p)$, il existe une action $I :: Can\_Incr_p \rightarrow Incr_p$ telle que, après stabilisation de $\mathcal{U}$, I est la seule action que p peut exécuter. Celle-ci incrémente l'horloge locale de p. De plus, l'exécution d'actions I n'est pas nécessaire à la stabilisation de $\mathcal{U}$.

Un algorithme qui réunit ces conditions peut être trouvé dans (Boulinier *et al.*, 2004).

Voici les détails de la composition ($\mathcal{A}^t$) entre $\mathcal{A}$ et l'horloge de phase autostabilisante $\mathcal{U}$. Pour tout processus p :

– $\mathcal{A}^t(p)$ contient toutes les variables de $\mathcal{A}(p)$ et $\mathcal{U}(p)$;

3. En particulier, si $\mathcal{A}$ est silencieux, alors toute configuration satisfaisant $SP_{\mathcal{A}}$ est terminale.

– $\mathcal{A}^t(p)$ contient toutes les actions de $\mathcal{U}(p)$ sauf I qui est remplacée par :
  - $A'_i :: Can\_Incr_p \wedge G_i \rightarrow Incr_p, S_i$ pour tout $i \in [0..x-1]$;
  - $L :: Can\_Incr_p \wedge Stable_p \wedge Late_p \rightarrow Incr_p$ où

$Stable_p \equiv (\forall i \in [0..x-1] \mid \neg G_i)$ et $Late_p \equiv \neg(\forall q \in \mathcal{N}_p \mid q.temps = p.temps)$.

Notre transformateur impose l'équité entre les processus qui sont activables dans $\mathcal{A}$. En effet, ils ne peuvent exécuter qu'un seul pas de $\mathcal{A}$ par tic d'horloge. Après stabilisation de $\mathcal{A}$, si $\mathcal{A}$ est silencieux, alors chaque processus p satisfait $Stable_p$ et, une fois que toutes les horloges ont la même valeur, plus aucune action n'est plus activable, ainsi la propriété de silence de $\mathcal{A}$ est préservée.

Les deux théorèmes suivants établissent la correction de notre transformateur.

THÉORÈME 20. — *$\mathcal{A}^t$ stabilise à $SP_{\mathcal{A}}$ en supposant un ordonnanceur inéquitable.*

PREUVE. — Par construction, toute exécution de $\mathcal{A}^t$ atteint en un nombre fini de pas de calcul une configuration γ qui est légitime pour $\mathcal{U}$. Considérons maintenant toute configuration γ' atteignable depuis γ. Supposons que le prédicat $\bigvee_{i\in[0..x-1]} G_i$ soit continûment vrai au processus p à partir de γ' mais que p n'exécute plus aucune action A'_i. Dans ce cas, $Stable_p$ est faux pour toujours et ainsi $p.temps$ n'est plus jamais incrémentée. Comme $\mathcal{U}$ fonctionne malgré un ordonnanceur inéquitable, tout processus $q \neq p$ finit par ne plus être activable. Dès lors, $f(p)$ est un minimum du système et, en particulier, Can_Incr_p est vrai. Donc p est activable pour exécuter l'une des ses actions A'_i. Comme p est le seul processus activable, il exécutera une de ses actions A'_i activables lors du prochain pas de calcul. Par conséquent, si le prédicat $\bigvee_{i\in[0..x-1]} G_i$ soit continûment vrai au processus p à partir de γ', alors p exécute une de ses actions A'_i activables après un nombre fini de pas de calcul à partir de γ'. Comme $\mathcal{A}$ stabilise en supposant un ordonnanceur faiblement équitable, $\mathcal{A}^t$ stabilise à la même spécification en supposant un ordonnanceur inéquitable. ■

THÉORÈME 21. — *Si $\mathcal{A}$ est silencieux, alors $\mathcal{A}^t$ est silencieux.*

PREUVE. — D'abord, par le théorème 20 et sa preuve, $\mathcal{A}^t$ converge vers une configuration γ depuis laquelle la spécification de l'algorithme $\mathcal{U}$ et le prédicat $Stable_p$ (pour tout nœud p) sont vrais pour toujours. Donc, à partir de γ, seule l'action L peut être exécutée par les nœuds. Soit $M = \max_{p\in V} f(p)$, et $m = \min_{p\in V} f(p)$. Tant que $M \neq m$, seuls des nœuds q tels que $f(q) \neq M$ peuvent être activables pour exécuter l'action L. De plus, lorsqu'ils exécutent l'action L, ils incrémentent $f(q)$ de 1. Par conséquent, $M = m$ en un temps fini, après quoi plus aucune action n'est plus activable dans le système. ■

Ci-après, nous présentons la complexité de l'algorithme transformé. Ces résultats supposent que $\mathcal{U}$ est l'algorithme de (Boulinier *et al.*, 2004). D'après (Boulinier *et al.*, 2004), $2n-1$ états par nœuds (en fait, la portée de l'horloge de phase) sont suffisants pour que l'algorithme $\mathcal{U}$ fonctionne dans n'importe quelle topologie (le pire cas étant une topologie de cycle). De plus, en utilisant $2n-1$ états, le temps de stabilisation de $\mathcal{U}$ est en $O(n)$ rondes (Boulinier, 2007) et $O(\mathcal{D}n^3)$ pas de calcul (Devismes, Petit, s. d.). Ainsi, nous avons le théorème suivant :

THÉORÈME 22. — *L'occupation mémoire de $\mathcal{A}^t$ est $O(\log n) + MEM$ bits par processus où MEM est l'occupation mémoire de $\mathcal{A}$.*

Ci-après, nous montrons une propriété supplémentaire de l'algorithme $\mathcal{U}$:

LEMME 23. — *Une fois que $\mathcal{U}$ a stabilisé, tout nœud avance son horloge de phase de $\mathcal{D}$ tics au plus tous les $2\mathcal{D}$ rondes.*

PREUVE. — Soit $f_\gamma^{\min} = \min_{p \in V} f(p)$ dans une configuration γ située après la stabilisation de $\mathcal{U}$. Soit un processus q et f_γ^q la valeur de $f(q)$ dans γ. On a : $f_\gamma^{\min} \leq f_\gamma^q \leq f_\gamma^{\min} + \mathcal{D}$. $2\mathcal{D}$ rondes après γ, on a : $f(q) \geq f_\gamma^{\min} + 2\mathcal{D}$. Donc, $f(q) - f_\gamma^q \geq f_\gamma^{\min} + 2\mathcal{D} - (f_\gamma^{\min} + \mathcal{D})$, c'est-à-dire, $f(q) - f_\gamma^q \geq \mathcal{D}$. D'où, q avance son horloge d'au moins $\mathcal{D}$ tics pendant cette période. ■

THÉORÈME 24. — *$\mathcal{A}^t$ stabilise à $SP_{\mathcal{A}}$ en $O(n + \lceil \frac{R}{\mathcal{D}} \rceil \times 2\mathcal{D})$ rondes où R est le temps de stabilisation de $\mathcal{A}$ en rondes ; et si $\mathcal{A}$ est silencieux, alors $\mathcal{A}^t$ atteint une configuration terminale en un nombre de ronde du même ordre.*

PREUVE. — Dans un premier temps, $\mathcal{A}^t$ stabilise à la spécification de $\mathcal{U}$ en $O(n)$ rondes. Puis, $\mathcal{A}^t$ doit émuler au plus R rondes de $\mathcal{A}$ pour stabiliser à $SP_{\mathcal{A}}$. D'après le lemme 23, cela nécessite au plus $\lceil \frac{R}{\mathcal{D}} \rceil \times 2\mathcal{D}$ rondes.

Supposons maintenant que $\mathcal{A}$ est silencieux. Ensuite, considérons la première configuration γ de $\mathcal{A}^t$ qui est légitime vis-à-vis de $SP_{\mathcal{A}}$ et de la spécification de $\mathcal{U}$. Soit les valeurs $M = \max_{p \in V} f(p)$, et $m = \min_{p \in V} f(p)$ dans γ. $M - m \leq \mathcal{D}$. D'où, d'après le lemme 23, après au plus $2\mathcal{D}$ rondes supplémentaires, $\mathcal{A}^t$ atteint une configuration terminale. ■

Le lemme suivant donne une borne sur le nombre de pas de calcul nécessaire pour émuler une ronde de $\mathcal{A}$, après que $\mathcal{U}$ soit stabilisé.

LEMME 25. — *Une fois que $\mathcal{U}$ a stabilisé, tout nœud continûment activable dans $\mathcal{A}^t$ exécute une action après au plus $2\mathcal{D}(n-1)$ pas de calcul.*

PREUVE. — Considérons une configuration γ où $\mathcal{U}$ a stabilisé et un nœud p qui soit continûment activable à partir de γ.

Tout nœud $q \neq p$ satisfait $f(p) - \|p,q\| \leq f(q) \leq f(p) + \|p,q\|$. Donc tout nœud $q \neq p$ peut incrémenter $q.temps$ au plus $2\|p,q\|$ fois avant que $p.temps$ soit incrémenté. Donc au plus $\sum_{q \in V \setminus \{p\}} 2\|p,q\|$ pas de calcul peuvent être effectués avant que p exécute une action. Or $\sum_{q \in V \setminus \{p\}} 2\|p,q\| \leq (n-1) \times 2\mathcal{D}$. ■

THÉORÈME 26. — *$\mathcal{A}^t$ stabilise à $SP_{\mathcal{A}}$ en $O(\mathcal{D}n(R+n^2))$ pas de calcul où R est le temps de stabilisation de $\mathcal{A}$ en rondes ; et si $\mathcal{A}$ est silencieux, alors $\mathcal{A}^t$ atteint une configuration terminale en un nombre de pas du même ordre.*

PREUVE. — Tout d'abord, $\mathcal{A}^t$ stabilise à la spécification de $\mathcal{U}$ en $O(\mathcal{D}n^3)$ pas de calcul. Après quoi R rondes de $\mathcal{A}$ sont émulées par $\mathcal{A}^t$ en $O(\mathcal{D}nR)$ pas de calcul d'après le lemme 25.

Supposons maintenant que $\mathcal{A}$ est silencieux. Ensuite, considérons la première configuration γ de $\mathcal{A}^t$ qui est légitime vis-à-vis de $SP_{\mathcal{A}}$ et de la spécification de $\mathcal{U}$. Soit les valeurs $M = \max_{p \in V} f(p)$, et $m = \min_{p \in V} f(p)$ dans γ. $M - m \leq \mathcal{D}$. D'où, après $O(\mathcal{D}n)$ pas supplémentaires, $\mathcal{A}^t$ atteint une configuration terminale. ■

À titre d'exemple, $\mathcal{SMDS}(k)^t$ stabilise à $SP_{\mathcal{SMDS}(k)}$ en $O(n)$ rondes et $O(\mathcal{D}n^3)$ pas avec $O(\log k + \log n + k \log \frac{N}{k})$ bits par processus, d'après les théorèmes 18-19 et 22-26. Ceci illustre que notre transformateur ne dégrade pas la complexité en rondes ni l'occupation mémoire tout en offrant une complexité satisfaisante en pas de calcul.

7. Simulations

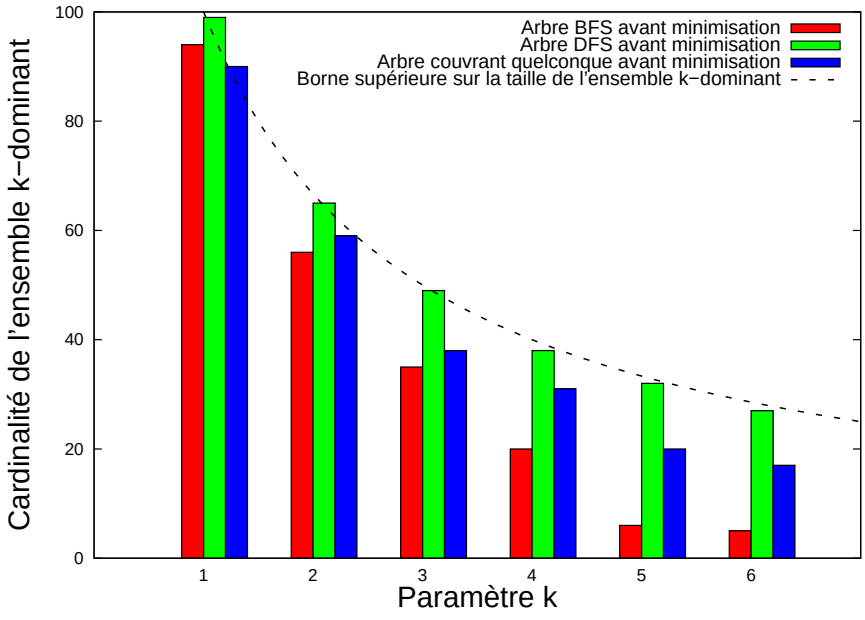
7.1. Modèle et hypothèses

Tous les résultats fournis dans cette section proviennent de WSNet (Hamida *et al.*, 2008), un simulateur événementiel pour les réseaux sans fil. Nous avons adapté notre algorithme au modèle à passage de messages, en utilisant des méthodes similaires à celles proposées dans (Delaët *et al.*, 2005).

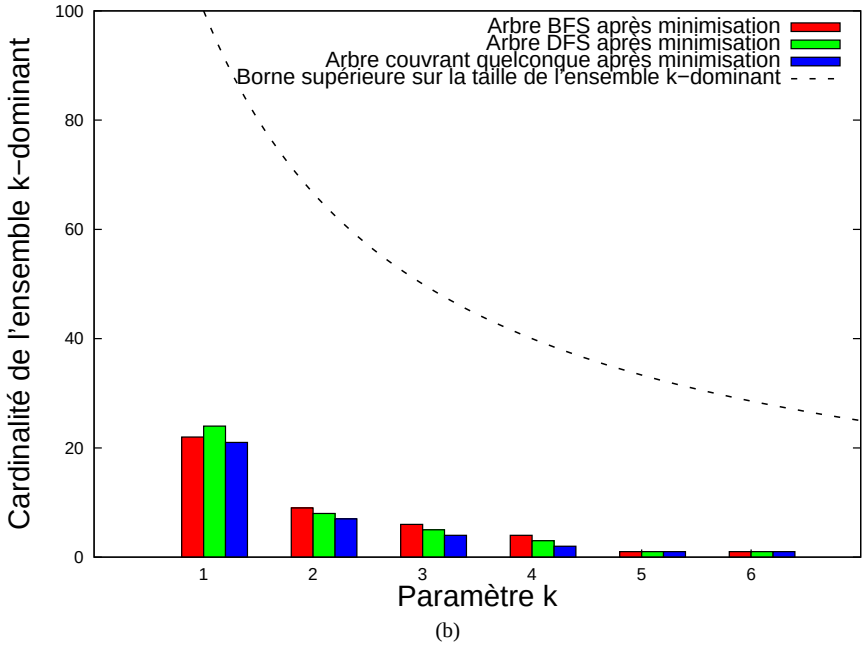
Dans WSNet, les processus sont déployés aléatoirement sur un plan carré. Les processus sont immobiles et équipés d'une interface radio. Chaque processus envoie périodiquement un paquet `hello`. Ces paquets servent à découvrir le voisinage, construire et maintenir une structure logique à travers le réseau. Deux processus u et v peuvent communiquer si et seulement si la distance euclidienne entre eux est au plus rad, où rad est la portée de transmission d'une radio. Ainsi, la topologie du réseau est isomorphe à un graphe de disques unitaires (UDG pour *Unit Disk Graph*). Par souci de simplicité, nous considérons les couches physiques et MAC comme idéales : il n'y a ni interférence ni collision. Néanmoins, comme le montrent (Delaët *et al.*, 2005), notre algorithme fonctionne même si les communications sont non fiables. En outre, les processus sont concurrents et asynchrones.

Dans nos simulations, nous considérons des réseaux UDG connexes de taille n variant entre 50 et 400 nœuds. Ceux-ci sont distribués aléatoirement dans un plan de $100\,m$ de côté selon une loi uniforme. Nous faisons varier la puissance de transmission selon le nombre de nœuds, afin d'obtenir un degré moyen entre 10 et 50. Par exemple, en fixant n à 200 et en réglant la portée de transmission entre $14\,m$ et $26\,m$, le degré moyen du réseau $\overline{d}$ est compris entre 10 et 50. Enfin, nous avons fait varier k de 1 à 6.

Les performances de $\mathcal{SMDS}(k)$ pouvant *a priori* dépendre du type d'arbre couvrant utilisé en second module, nous avons testé notre protocole avec trois types d'arbre couvrant différents : un arbre couvrant en profondeur (arbre DFS), en largeur (arbre BFS) et un arbre couvrant quelconque.



(a)



(b)

Figure 4. Taille moyenne de l'ensemble k-dominant calculé en fonction de k (avec $n = 200$ et $\overline{d} \in]10, 20[$) : (a) avant ; (b) après minimisation

7.2. Motivations

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, est-ce que le choix d'un arbre couvrant spécifique influe de manière significative sur la taille de l'ensemble k-dominant construit par $\mathcal{DS}(k)$ ou $\mathcal{MIN}(k) \circ \mathcal{DS}(k)$? 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 ?

7.3. Résultats

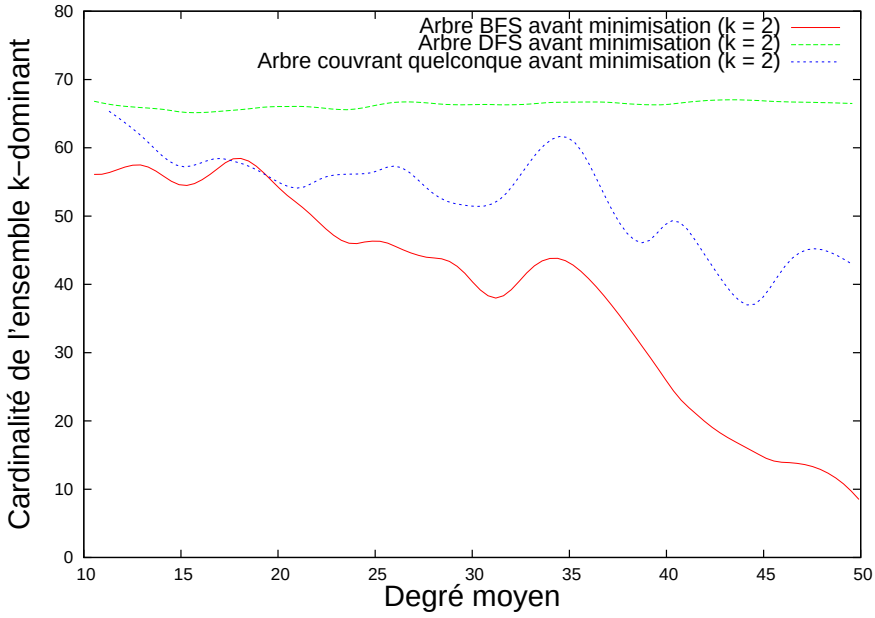
Nous mettons ici en évidence les performances de notre algorithme, en termes de taille de l'ensemble k-dominant construit par $\mathcal{DS}(k)$ et $\mathcal{MIN}(k) \circ \mathcal{DS}(k)$ dans des topologies aléatoires, par rapport à k, n, $\overline{d}$ et à l'arbre couvrant choisi. Notez que pour chaque paramétrage, nous avons effectué 50 expériences.

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. Dans ce cas, la taille moyenne obtenue par simulation est ainsi proche de la borne supérieure théorique. En revanche, de meilleures performances sont obtenues sur des arbres quelconques ou BFS. Dans ces résultats, la hauteur de l'arbre a également un impact majeur sur la taille de l'ensemble k-dominant.

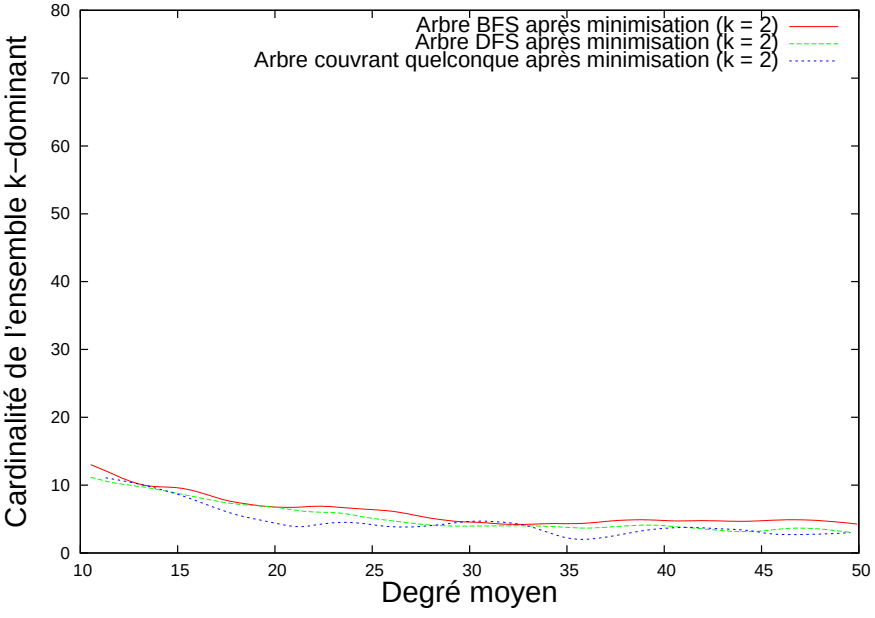
L'influence du degré moyen $\overline{d}$ peut être observée dans la figure 5a. La taille de l'ensemble k-dominant construit sur un arbre DFS est constante et élevée, alors que la taille d'un tel ensemble construit sur un autre arbre est décroissante. En effet, quand le degré moyen augmente, le diamètre 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 (figure 4a et figure. 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 troisième 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 $\mathcal{DS}(k)$ 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 $\mathcal{MIN}(k)$



(a)



(b)

Figure 5. Taille moyenne de l'ensemble k-dominant en fonction du degré moyen $\overline{d}$ (avec $k = 2$ et $n = 200$) : (a) avant minimisation (b) après minimisation

est assez similaire quel que soit le type d'arbre considéré, malgré un léger avantage pour les arbres quelconques.

Le troisième 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 montrent que la taille de l'ensemble k-dominant calculé par $\mathcal{DS}(k)$ est influencée par le type d'arbre couvrant utilisé. Elles soulignent surtout le fait que la taille de l'ensemble k-dominant minimisé par $\mathcal{MIN}(k)$ est significativement plus petite que la borne théorique dans la grande majorité des cas. Par exemple, le tableau 1 montre le gain moyen obtenu par la minimisation des ensembles k-dominants calculés par $\mathcal{DS}(k)$ pour $k = 2$, $n = 200$ et $\overline{d} \in]10, 20[$.

Enfin, sur l'ensemble des simulations que nous avons faites, nous observons que notre algorithme en trois modules, $\mathcal{SMDS}(k)$, calcule des ensembles k-dominants minimaux dont la taille moyenne est radicalement plus petite que la borne théorique, comme le montre par exemple la figure 4b. Plus précisément, pour $n = 200$, $1 \leq k \leq 6$ et $\overline{d} \in]10, 20[$, la taille des ensembles k-dominants que nous obtenons est en moyenne $89\ \%$ plus petite que la borne théorique.

Tableau 1. Gain moyen obtenu grâce à la minimisation

Arbre	Avant minimisation	Après minimisation	Gain moyen
BFS	56.93	9.93	83 %
DFS	65.87	8.93	86 %
Quelconque	59.17	7.83	87 %

8. Conclusion

Dans cet article, nous avons proposé un algorithme déterministe, distribué, asynchrone, autostabilisant et silencieux pour calculer un ensemble k-dominant minimal de taille au plus $\lceil \frac{n}{k+1} \rceil$ dans un réseau quelconque. Nous avons montré sa correction en considérant un ordonnanceur faiblement équitable. Nous avons ensuite proposé une méthode pour transformer un algorithme fonctionnant sous l'hypothèse faiblement équitable en un algorithme fonctionnant sous l'hypothèse inéquitable. En appliquant notre transformateur sur notre première solution, nous obtenons un algorithme qui reste silencieux, qui stabilise en $0(n)$ rondes et $O(\mathcal{D}n^3)$ pas de calcul, et qui requiert $O(\log k + \log n + k \log \frac{N}{k})$ bits par nœuds, où $\mathcal{D}$ est le diamètre du réseau et N est un majorant de n. Nos résultats de simulation montrent que la taille de l'ensemble k-dominant obtenu par notre solution est généralement beaucoup plus petite que $\lceil \frac{n}{k+1} \rceil$.

Une extension envisageable de ce travail est d'étudier s'il est possible d'améliorer le temps de stabilisation en $O(k)$ rondes (l'optimal). Dans de futurs travaux, nous souhaiterions trouver un algorithme distribué autostabilisant 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 tel 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

Amis A. D., Prakash R., Huynh D., Vuong T. (2000). Max-min d-cluster formation in wireless ad hoc networks. In *International conference on computer communications*, p. 32-41.

Beauquier J., Gradinariu M., Johnen C. (2001). Cross-over composition - enforcement of fairness under unfair adversary. In *Workshop on self-stabilizing systems*, p. 19–34.

Boulinier C. (2007). *L'unisson*. Thèse de doctorat non publiée, Université de Picardie Jules Verne, Amiens. (Dissertation in french)

Boulinier C., Petit F., Villain V. (2004). When graph theory helps self-stabilization. In *Acm symposium on principles of distributed computing*, p. 150–159.

Caron E., Datta A. K., Depardon B., Larmore L. L. (2010). A self-stabilizing k-clustering algorithm for weighted graphs. *Journal of Parallel and Distributed Computing*, vol. 70, n° 11, p. 1159-1173.

Cournier A., Datta A. K., Petit F., Villain V. (2003). Enabling snap-stabilization. In *International conference on distributed computing systems*, p. 12-19.

Datta A. K., Devismes S., Larmore L. L. (2009). A self-stabilizing o(n)-round k-clustering algorithm. In *International symposium on reliable distributed systems*, p. 147–155.

Datta A. K., Larmore L. L., Vemula P. (2010). A Self-Stabilizing O(k)-Time k-Clustering Algorithm. *Comp. J.*, vol. 53, n° 3, p. 342-350.

Datta A. K., Larmore L. L., Vemula P. (2011). Self-stabilizing leader election in optimal space under an arbitrary scheduler. *Theor. Comput. Sci.*, vol. 412, n° 40, p. 5541-5561.

Delaët S., Ducourthial B., Tixeuil S. (2005). Self-stabilization with r-operators revisited. In *Self-stabilizing systems*, p. 68-80.

Devismes S., Petit F. (s. d.). On efficiency of unison. In *Workshop on theoretical aspects of dynamic distributed systems*, p. To appear. (2012)

Dijkstra E. W. (1974). Self-stabilizing systems in spite of distributed control. *Commun. ACM*, vol. 17, p. 643–644.

Dolev S. (2000). *Self-stabilization*. MIT Press.

Dolev S., Gouda M. G., Schneider M. (1996). Memory requirements for silent stabilization. In *Acm symposium on principles of distributed computing*, p. 27–34.

Fernandess Y., Malkhi D. (2002). K-clustering in wireless ad hoc networks. In *Workshop on principles of mobile computing*, p. 31-37.

Garey M. R., Johnson D. S. (1979). *Computers and intractability : A guide to the theory of np-completeness*. W. H. Freeman.

Hamida E. B., Chelius G., Gorce J.-M. (2008). Scalable versus accurate physical layer modeling in wireless network simulations. In *Workshop on principles of advanced and distributed simulation*, p. 127-134.

Herman T. R. (1992). *Adaptivity through distributed convergence*. Thèse de doctorat non publiée, Austin, TX, USA. (UMI Order No. GAX92-12547)

Huang S.-T., Chen N.-S. (1992). A self-stabilizing algorithm for constructing breadth-first trees. *Inf. Process. Lett.*, vol. 41, p. 109–117.

Ikeda M., Kamei S., Kakugawa H. (2002). A space-optimal self-stabilizing algorithm for the maximal independent set problem. In *Conference on parallel and distributed computing, applications and technologies*, p. 70-74.

Katz S., Perry K. J. (1993). Self-stabilizing extensions for message-passing systems. *Distributed Computing*, vol. 7, n° 1, p. 17-26.

Kosowski A., Kuszner L. (2006). Energy optimisation in resilient self-stabilizing processes. In *Symposium on parallel computing in electrical engineering*, p. 105–110.

Kutten S., Peleg D. (1995). Fast distributed construction of k-dominating sets and applications. In *Acm symposium on principles of distributed computing*, p. 238–251.

Peleg D., Upfal E. (1989). A trade-off between space and efficiency for routing tables. *Journal of the ACM*, vol. 36, n° 3, p. 510–530.

Penso L. D., Barbosa V. C. (2004). A distributed algorithm to find k-dominating sets. *Discrete Appl. Math.*, vol. 141, n° 1-3, p. 243–253.

Ravelomanana V. (2005). Distributed k-clustering algorithms for random wireless multihop networks. In *International conference on networking*, p. 109-116.

Shukla S. K., Rosenkrantz D. J., Ravi S. S. (1995). Observations on self-stabilizing graph algorithms on anonymous networks. In *Workshop on self-stabilizing systems*, p. 7.1–7.15.

Tel G. (2001). *Introduction to distributed algorithms* (2nd éd.). Cambridge University Press.

Reçu le 7 septembre 2011
Accepté après révision le 21 juin 2012

Ajoy K. Datta. *Professeur à l'université de Las Vegas Nevada. Ses recherches portent sur l'algorithmique distribuée et l'autostabilisation.*

Stéphane Devismes. *Maître de conférences à l'université Joseph Fourier de Grenoble (Grenoble I) et membre de l'équipe Synchrone du laboratoire VERIMAG de Grenoble. Ses recherches portent sur l'algorithmique distribuée, la tolérance aux fautes, et en particulier l'autostabilisation.*

Karel Heurtefeux. *Post-doctorant au CNRS et membre de l'équipe Synchrone du laboratoire VERIMAG. Ses recherches portent sur les protocoles de communication dans les réseaux sans fils. Il s'intéresse en particulier au routage, à l'accès au médium radio et à la localisation.*

Lawrence L. Larmore. *Professeur à l'université de Las Vegas Nevada. Ses recherches portent sur l'algorithmique distribuée, l'autostabilisation et les algorithmes en ligne.*

Yvan Rivierre. *Doctorant à l'université Joseph Fourier de Grenoble (Grenoble I) et membre de l'équipe Synchrone du laboratoire VERIMAG. Ses recherches portent sur l'autostabilisation, notamment dans les réseaux de capteurs sans fils.*