Construction auto-stabilisante d'un arbre couvrant maximisant le nombre de feuilles

Stephane Rovedakis

Construction auto-stabilisante d'un arbre couvrant maximisant le nombre de feuilles

Stephane Rovedakis[1]

[1]*Laboratoire IBISC, Université d'Evry, 91000 Evry, France (supporté par le projet* ANR SHAMAN*).*

Pour router les messages dans des réseaux auto-organisés, comme les réseaux sans fil ad-hoc ou de senseurs, un ensemble dominant connexe (CDS) est souvent utilisé. La construction d'un arbre couvrant maximisant le nombre de feuilles (MLST) permet d'obtenir un CDS de petite taille pour router les messages dans ce type de réseaux tout en économisant l'énergie. Le paradigme de l'auto-stabilisation est une technique générale et flexible permettant de traiter les fautes transitoires qui peuvent survenir sur les éléments constituant un réseau auto-organisé.

Le problème de la construction d'un MLST est un problème NP-difficile à résoudre, ainsi une solution approchée est recherchée. Nous proposons un algorithme distribué et auto-stabilisant pour la construction d'un MLST considérant une topologie arbitraire, sans aucune connaissance sur le réseau et ayant un meilleur rapport d'approximation que les travaux existants. Cet algorithme permet de construire une $1/2$-approximation par rapport à une solution optimale, c'est-à-dire que l'arbre couvrant construit possède un nombre de feuilles au moins égal à la moitié des feuilles d'une solution optimale.

Keywords: Tolérance aux pannes, auto-stabilisation, algorithme distribué, construction d'arbre couvrant.

1 Introduction

Les réseaux sans fil sont employés dans un nombre grandissant d'applications (ad-hoc, de senseurs, de robots ...). Ce type de réseaux fonctionne d'une façon auto-organisée, et ne possède aucune structure fixe, ni aucune gestion centralisée. Ce mode de fonctionnement impose à ses composants de construire et de maintenir collaborativement une structure de communication utilisée par les applications haut niveau. Un *ensemble dominant connexe* (CDS) est un sous-ensemble connexe de noeuds tel que tout noeud hors de cet ensemble, appelé noeud *dominé*, est voisin d'au moins un noeud du CDS. Un CDS est souvent employé pour le routage dans ce type de réseaux, à cause de sa simplicité et de l'existence d'*au moins* un chemin entre toute paire de noeuds. Afin d'optimiser l'énergie, on cherche à minimiser la taille du CDS car les noeuds dans le CDS consomment plus d'énergie pour gérer le trafic. Le problème de la construction d'un *arbre couvrant maximisant le nombre de feuilles* (MLST) consiste à construire un arbre couvrant le réseau ayant le plus grand nombre de feuilles possible. Ce problème est intéressant d'un point de vu théorique, mais aussi d'un point de vu pratique pour la communication dans les réseaux auto-organisés. En effet, la construction d'un MLST permet d'obtenir un CDS (noeuds internes du MLST) dont la taille est minimisée du fait de la maximisation du nombre de noeuds dominés (feuilles du MLST).

Les réseaux sans fil sont sujets aux changements topologiques et aux fautes. Pour traiter ces évènements, nous utilisons le paradigme de l'*auto-stabilisation* [Dol00]. L'auto-stabilisation est une technique flexible permettant de tolérer les fautes dans les réseaux et les systèmes distribués. Un algorithme distribué est dit *auto-stabilisant* si à la suite de fautes ou d'attaques plaçant le système dans un état global arbitraire, le système retrouve en un temps fini un comportement (état global) correct sans intervention extérieure. Comme aucune hypothèse n'est faite sur la nature des fautes, cette technique tolère aussi les changements dynamiques sur la topologie du réseau car ces modifications sont vues comme des fautes par le système.

Travaux précédents. Le problème de la construction d'un MLST est NP-difficile à résoudre de façon optimale. Solis-Oba a proposé dans [SO98] un algorithme séquentiel permettant de construire une $1/2$-approximation d'un MLST optimal dans un graphe quelconque. L'algorithme possède deux phases. La

première phase construit une forêt F d'arbres ayant un grand nombre de feuilles. La racine de chaque arbre de F est un noeud de degré au moins trois n'appartenant à aucun arbre. Une règle de croissance est appliquée sur un noeud feuille v d'un arbre. Il y a deux types de règles qui ajoutent de nouveaux noeuds n'appartenant à aucun arbre : (1) ajoute au moins deux voisins à v, (2) ajoute un voisin u à v ayant lui-même au moins deux voisins. Ces règles sont appliquées sur un arbre tant que cela est possible (en privilégiant le premier type), sinon un nouvel arbre est construit. Lorsqu'il n'est plus possible de construire un nouvel arbre, F contient soit des arbres ayant au moins quatre noeuds, soit un unique noeud. La seconde phase connecte les arbres de F de manière arbitraire pour obtenir un arbre couvrant.

À notre connaissance, il n'existe qu'un seul travail proposant un algorithme distribué auto-stabilisant pour le problème du MLST [DKKTre]. Une $1/3$-approximation d'un MLST optimal est calculée pour un graphe quelconque. L'algorithme est une composition de quatre couches auto-stabilisantes : (1) chaque noeud calcule le plus grand couple (degré et identifiant d'un noeud) du réseau, (2) une forêt d'arbres est construite s'apparentant à celle construite dans [SO98] mais contenant le minimum d'arbres possible, (3) à chaque arête e du graphe est attribué un poids égale à 0 si e appartient à un arbre de la forêt, à ∞ si e connecte deux noeuds du même arbre ou à 1 sinon, (4) pour obtenir un arbre couvrant les arbres de la forêt sont fusionnés à l'aide du calcul d'un arbre couvrant de poids minimum utilisant les poids attribués en (3).

Nos résultats. Nous proposons un algorithme distribué auto-stabilisant pour le problème du MLST, capable de tolérer les fautes transitoires pouvant survenir dans un réseau. L'algorithme proposé apporte plusieurs améliorations comparé à [DKKTre]. Tout d'abord, notre algorithme construit une $1/2$-approximation d'un MLST optimal. Ensuite, la topologie du réseau considéré est arbitraire mais aucune connaissance sur le réseau n'est nécessaire, contrairement à [DKKTre] qui nécessite la connaissance du nombre de noeuds du réseau. Enfin, l'algorithme retourne un MLST en au plus $O(n^2)$ rounds, contrairement à [DKKTre] qui a besoin dans le pire des cas d'au moins $O(mn)$ rounds (meilleure complexité connue) pour le calcul d'un arbre couvrant de poids minimum, où n est le nombre de noeuds et m le nombre d'arêtes du réseau.

2 Modèle et notations

Le réseau est modélisé par un graphe non-orienté $G = (V,E)$, où V est l'ensemble des noeuds ($n = |V|$) et E est l'ensemble des arêtes ($m = |E|$). Les noeuds voisins du noeud v dans G sont notés $N(v)$. Un arbre couvrant T de G est un sous-graphe acyclique et connexe de G, tel que $T = (V_T, E_T), V_T = V$ et $E_T \subseteq E$. Le *degré d'un noeud* dans le sous-graphe S est le nombre de noeuds voisins de v dans S, noté $deg_S(v)$. Une *feuille* dans un arbre T est un noeud v dont le degré dans T est égal à un (i.e., $deg_T(v) = 1$). $l(T)$ retourne l'ensemble des noeuds feuilles dans T (i.e., $l(T) = \{v : v \in V_T, deg_T(v) = 1\}$).

On considère un système distribué asynchrone dans lequel chaque noeud $v \in V$ possède un identifiant unique, noté ID_v. Le modèle à états [Dol00] est utilisé pour la communication des noeuds, où tout noeud $v \in V$ peut lire la valeur des variables de ses voisins $u \in N(v)$ et réciproquement.

L'*état local* d'un noeud est défini par la valeur de ses variables locales et de son compteur de programme. Une *configuration* du système est l'union des états locaux de tous les noeuds du système. La transition d'une configuration à une autre est réalisée par l'exécution d'une action atomique sur un noeud. Une *action atomique* sur un noeud v est un calcul interne basé sur l'état de v et de l'état de ses voisins $u \in N(v)$. Une *exécution* du système est une séquence infinie de configurations, $e = (c_0, c_1, \ldots c_i, \ldots)$, où chaque configuration c_{i+1} est obtenue de c_i en exécutant au moins une action atomique. Dans la suite, nous considérons que le système peut partir d'une configuration quelconque. En d'autres termes, l'état local d'un noeud peut être corrompu. Aucune borne sur le nombre de noeuds corrompus n'est supposée. Dans le pire des cas, le système peut partir d'une configuration où l'état de tous les noeuds sont corrompus. Afin de traiter ces fautes, nous utilisons le paradigme de l'*auto-stabilisation*.

Définition 1 (algorithme auto-stabilisant) *Étant donné un* prédicat de légitimité[†] *(non-vide)* $\mathcal{L}_{\mathcal{A}}$*, un algorithme* $\mathcal{A}$ *est dit* auto-stabilisant *ssi les deux conditions suivantes sont respectées : (i) Toute exécution de* $\mathcal{A}$ *partant d'une configuration satisfaisant* $\mathcal{L}_{\mathcal{A}}$ *préserve* $\mathcal{L}_{\mathcal{A}}$ *(*clôture*). (ii) Toute exécution de* $\mathcal{A}$ *partant d'une configuration arbitraire contient une configuration qui satisfait* $\mathcal{L}_{\mathcal{A}}$ *(*convergence*).*

[†] un prédicat de légitimité est défini sur les configurations du système et indique si une configuration est correcte ou non.

Une *configuration légitime* pour le problème considéré est une configuration où un unique arbre couvrant T est construit. De plus, nous désirons avoir une $1/2$-approximation pour le problème du MLST, c'est-à-dire $\frac{|l(T)|}{|l(T^*)|} \geq \frac{1}{2}$ où T^* est une solution optimale.

3 Notre algorithme auto-stabilisant pour le problème du MLST

L'algorithme possède deux couches auto-stabilisantes exécutées simultanément. La première couche reprend la phase 1 de [SO98] et construit une forêt F d'arbres *feuillus* constituée d'*étoiles* d'*au moins trois branches* ou de noeuds *isolés* (Fig. 1(a)). La seconde couche termine la phase 1 et réalise la phase 2 de [SO98] en fusionnant tous les arbres de la forêt F (Fig. 1(b)) pour obtenir un arbre couvrant V (Fig. 1(c)). La première couche contraint la forêt F à ne contenir que des noeuds isolés ou des étoiles ayant au moins trois branches permettant la construction d'un arbre couvrant contenant un plus grand nombre de feuilles.

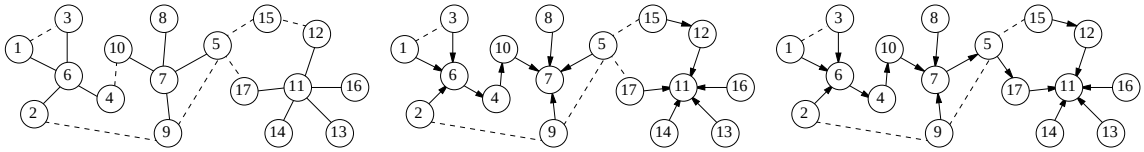

FIG. 1: Exemple de construction d'un MLST par l'algorithme.

3.1 Construction d'une forêt d'arbres feuillus

Pour tout noeud $v \in V$, on définit Max_v comme le couple degré et identifiant de v (i.e., $Max_v \equiv (deg_G(v), ID_v)$), et on note $Max_v[0] = deg_G(v)$ (resp. $Max_v[1] = ID_v$). Les noeuds v ayant localement le plus grand Max_v deviennent racines de leur étoile (ou noeuds isolés), sinon v devient feuille de l'étoile de u ayant le plus grand Max_u dans le voisinage de v. Ainsi, Max_r identifie l'étoile du noeud racine r.

Pour construire ces étoiles, trois variables sont maintenues sur chaque noeud v : max_l_v indique le plus grand Max_u dans le voisinage de v (avec $u \in N(v) \cup \{v\}$), nbf_v stocke le nombre de noeuds voisins u tel que $max\_l_u = Max_v$ et $root\_l_v$ est égal à l'identifiant de l'étoile Max_u contenant v (avec $u \in N(v) \cup \{v\}$).

Tout d'abord, chaque noeud v corrige sa variable max_l_v de sorte à ce qu'elle soit égale au plus grand Max_u local, c'est-à-dire si $max\_l_v \neq ValMax(v)$ alors $max\_l_v := ValMax(v)$ avec $ValMax(v) \equiv \max\{Max_u : u \in N(v) \cup \{v\}\}$. Ensuite, en se basant sur la valeur des variables max_l_u de ses voisins u chaque noeud v compte le nombre de fils qu'il pourrait avoir dans son étoile. Ainsi, si $nbf_v \neq Nbfils(v)$ alors nbf_v prend la valeur retournée par $Nbfils(v)$, avec $Nbfils(v) \equiv |\{u : u \in N(v), max\_l_u = Max_v\}|$. Enfin, lorsque les variables max_l_v et nbf_v sont corrigées (c'est-à-dire $max\_l_v = ValMax(v)$ et $nbf_v = Nbfils(v)$), alors le noeud v peut déterminer l'identifiant de son étoile. Il y a deux cas à considérer soit v est la racine de son étoile, soit v possède un voisin u avec un plus grand Max_u et u est la racine de son étoile. Donc, si $nbf_v \geq 3$ et $root\_l_v \neq Max_v$ alors v se déclare racine de son étoile en corrigeant sa variable $root\_l_v$ à Max_v. Dans le cas où $nbf_v < 3$ et $root\_l_v \neq max\_l_v$, le noeud v met à jour sa variable $root\_l_v$ égal à max_l_v. Dans ce dernier cas, soit v possède un voisin u tel que $Max_v < Max_u$ alors v choisit le voisin u ayant le plus grand Max_u, sinon v est un noeud isolé et devient la racine de son étoile.

3.2 Construction de l'arbre couvrant

La seconde couche considère une forêt F d'arbres qui doivent fusionner jusqu'à obtenir un unique arbre couvrant. Les étoiles construites par la première couche constituent les arbres de base de la forêt F. L'arbre couvrant final est enraciné au noeud u de plus grand Max_u et contient toutes les arêtes des étoiles. Une étoile e_1 est rattachée à une étoile e_2 si l'identifiant de e_2 est supérieur (i.e., $Max_{e_1} < Max_{e_2}$). Pour cela, une élection est faite dans chaque étoile. Les noeuds d'une étoile e_1 sélectionnent le plus grand identifiant parmi les étoiles voisines, et la racine élit le plus grand identifiant. Si une étoile voisine e_2 possède un identifiant localement supérieur à e_1, alors le noeud de e_1 voisin d'un noeud de e_2 le prend comme parent pour fusionner avec e_2 (dans le cas d'une feuille il y a une inversion de parenté dans e_1).

Cette construction nécessite cinq variables sur chaque noeud v : p_v stocke l'identifiant du noeud parent de v dans l'arbre, $root_v$ est égal à Max_r avec r le noeud racine de l'arbre contenant v, d_v est la distance (en nombre de sauts) entre v et r, $root\_cand_v$ indique le plus grand $root_u$ dans le voisinage de v et $root\_cand\_neig_v$ indique le voisin u tel que $root\_cand_v = root_u$.

Chaque arbre T_i dans F a un identifiant stocké dans $root_r$, égal à la variable $root\_l_r$ du noeud racine r de T_i. L'arbre T_j d'identifiant plus petit que l'arbre T_i se rattache à T_i. Pour cela nous adaptons l'approche utilisée dans [AKY91], en considérant des étoiles qui rejoignent un arbre au lieu de simple noeuds. Un noeud v est *cohérent* si (1) v est racine de son arbre et $p_v = ID_v, root_v = root\_l_v$ et $d_v = 0$, ou (2) v n'est pas racine et $p_v \in N(v), root_v = root_{p_v}, root_v \geq root\_l_v$ et $d_v = d_{p_v} + 1$.

La seconde couche se découpe en trois parties : la première se charge de vérifier et corriger l'état des noeuds, la deuxième identifie les paires d'arbres qui doivent fusionner et la dernière réalise les fusions.

Correction de l'état d'un noeud. Il y a deux classes de noeuds : les noeuds racines d'une étoile et les noeuds feuilles d'une étoile. Dans le premier cas, si le noeud v n'est pas cohérent alors v réinitialise son état et devient racine de son arbre, c'est-à-dire $p_v = ID_v, root_v = root\_l_v$ et $d_v = 0$. Dans le second cas, si le noeud v n'est pas cohérent alors v réinitialise son état et prend comme parent la racine de son étoile calculée par la première couche, c'est-à-dire $p_v = root\_l_v[1], root_v = root_{p_v}$ et $d_v = d_{p_v} + 1$.

Calcul de l'identifiant maximum. Chaque noeud cohérent v calcule dans $root\_cand_v$ le plus grand identifiant de l'arbre avec lequel réaliser la prochaine fusion. Cet identifiant est égal au plus grand $root_u$ parmi les voisins u hors de son arbre si v est feuille d'une étoile, ou le maximum parmi le plus grand $root_u$ (u hors de l'étoile de v) et le plus grand $root\_cand_u$ (avec u dans la même étoile que v) si v est racine d'une étoile. De plus, la racine v d'une étoile stocke dans la variable $root\_cand\_neig_v$ l'identifiant du noeud dans son étoile voisin du noeud ayant le plus grand identifiant d'arbre calculé. Cela permettra de pouvoir réaliser une fusion à travers une feuille de l'étoile. Ainsi, si sur un noeud cohérent v les variables $root\_cand_v$ et $root\_cand\_neig_v$ ne contiennent pas l'identifiant d'arbre localement maximum et l'identifiant du noeud voisin de cet arbre, alors v corrige ces deux variables en conséquence.

Fusion. Étant donné un noeud cohérent v dont les variables $root\_cand_v$ et $root\_cand\_neig_v$ sont corrigées. Il faut considérer deux cas : soit v est le noeud d'une étoile adjacent au noeud de l'arbre d'identifiant maximum, soit v est racine d'une étoile et n'est pas adjacent au noeud de l'arbre d'identifiant maximum. Dans le premier cas, soit v est racine d'une étoile ou la racine de l'étoile de v a sélectionné comme identifiant d'arbre maximum la valeur calculée par v (c'est-à-dire $root\_cand_{root\_l_v[1]} = root\_cand_v$ et $root\_cand\_neig_{root\_l_v[1]} = ID_v$). Alors v doit fusionner avec l'arbre de plus grand identifiant en choisissant comme nouveau parent le voisin u tel que u a le plus grand identifiant $root_u$. Il faut noter qu'en cas d'égalité, v choisira le voisin ayant le plus grand Max_u. Pour cela, v met à jour ses variables comme suit : $root_v = root\_cand_v, p_v = \max\{ID_u : u \in N(v), root_u = root_c and_v\}$ et $d_v = d_{p_v} + 1$. Dans le second cas, si le noeud feuille u de l'étoile de v a changé de parent (c'est-à-dire $root\_cand\_neig_v = ID_u$ et $p_u \neq ID_v$), alors v choisit u comme nouveau parent et met à jour ses variables comme suit : $root_v = root_{root\_cand\_neig_v}, p_v = root\_cand\_neig_v$ et $d_v = d_{p_v} + 1$.

Lemme 1 *Partant d'une configuration arbitraire, l'algorithme retourne une* $1/2$*-approximation d'un arbre couvrant maximisant le nombre de feuilles optimal dans un graphe quelconque. Sa complexité en temps est de* $O(n^2)$ *rounds et en mémoire de* $O(\log n)$ *bits, avec n le nombre de noeuds du graphe.*

Références

[AKY91] Y. Afek, S. Kutten, and M. Yung. Memory-efficient self stabilizing protocols for general networks. In Springer, editor, *WDAG*, volume LNCS 486, pages 15–28, 1991.

[DKKTre] S. Devismes, H. Kakugawa, S. Kamei, and S. Tixeuil. A self-stabilizing 3-approximation for the maximum leaf spanning tree problem in arbitrary networks. In *COCOON*, 2010 (à paraître).

[Dol00] S. Dolev. *Self-Stabilization*. MIT Press, 2000.

[SO98] R. Solis-Oba. 2-approximation algorithm for finding a spanning tree with maximum number of leaves. In *ESA*, pages 441–452, 1998.