Algorithme distribué d'orientation de graphes dans un environnement asynchrone et avec pannes

Noël Gillet, Nicolas Hanusse

Algorithme distribué d'orientation de graphes dans un environnement asynchrone et avec pannes

Noël Gillet[1] et Nicolas Hanusse[2]

[1]*LaBRI, Université de Bordeaux*
[2]*LaBRI, CNRS*

Nous nous intéressons dans ce papier à l'orientation de graphe de manière distribuée. Plus précisément, nous cherchons à calculer une *orientation minimum*, c'est-à-dire à minimiser le degré sortant maximum d'un nœud du graphe. Ce problème d'orientation est notamment une modélisation naturelle pour des problèmes d'allocation de ressources. Nous présentons l'algorithme `AvrDegAsync` qui fonctionne dans un environnement distribué où les communications sont *asynchrones* et où les nœuds peuvent être *en panne*. Notre algorithme garantit une $2(2+\varepsilon)$-approximation de l'orientation optimale en utilisant un nombre logarithmique de diffusion. De plus, il ne nécessite pas de connaissance sur le graphe comme le nombre de nœuds ou encore sa densité.

Mots-clefs : Algorithme distribué, Environnement asynchrone, Orientation de graphe, Tolérance aux pannes

1 Introduction

Contexte et motivations Dans ce papier, nous nous intéressons à un problème classique en théorie des graphes qui consiste à trouver une orientation des arêtes d'un graphe tel que le degré sortant maximum d'un nœud soit minimisé. Ce problème *d'orientation minimum* est une modélisation naturelle pour les problèmes d'allocation de ressources. Considérons par exemple le cas d'une base de données distribuée qui doit traiter un ensemble de requêtes. Supposons que chaque requête puisse être traitée par un ensemble de nœuds candidats. On peut représenter les requêtes et leurs candidats par un graphe de requêtes où une arête représente une requête et ses extrémités encodent les candidats pour cette requête. L'orientation des arêtes va symboliser l'allocation des requêtes correspondantes. Une autre application concerne les réseaux de capteurs sans fil qui sont des ensembles d'agents autonomes chargés notamment de collecter des données sur leur environnement. Une problématique essentielle dans ce contexte est de minimiser la consommation d'énergie afin de maximiser la durée de vie du réseau. Pour cela, trouver une bonne allocation des tâches à effectuer par les agents est une piste naturelle. On peut ici aussi modéliser ce problème par une orientation de graphe. On remarque que dans ces deux exemples, les décisions d'allocations doivent se prendre de manière distribuée et les communications sont principalement asynchrones. De plus, certains nœuds peuvent tomber en panne, ce qui complique le processus d'allocation. Nous nous intéressons donc à ces différents aspects (asynchronisme et présence de pannes) dans ce papier.

Une brève remarque Habituellement, le problème de l'orientation minimum décrit dans la littérature consiste à considérer un graphe non orienté en entrée et à calculer une orientation de ce graphe. Ici nous optons pour une approche différente qui consiste à considérer un graphe *bidirigé* et à conserver pour chaque paire de nœuds un seul des deux arcs qui les relient. On remarque facilement que ces deux versions sont équivalentes. L'utilisation d'un graphe bidirigé est justifiée par la présence potentielle de pannes. En effet, dans le cas où les communications sont asynchrones, il est a priori impossible de déterminer si un nœud est seulement lent à répondre à un message ou bien s'il est en panne. Dans le contexte de l'allocation de ressources, on veut pourtant que les nœuds puissent se mettre d'accord afin d'allouer le maximum de tâches. Avec l'utilisation d'un graphe bidirigé on suppose que tous les nœuds candidats pour une tâche veulent la

traiter et on garantit donc que si au moins un candidat est correct, la tâche sera allouée. On remarque cependant que si tous les candidats sont en panne, alors la tâche sera perdue.

Formalisation du problème Soit $\overrightarrow{G} = (V, \overrightarrow{E})$ un graphe simple bidirigé avec $|V| = n$ nœuds et $|\overrightarrow{E}| = m$ arcs. Comme $\overrightarrow{G}$ est bidirigé cela implique que si $(u,v) \in \overrightarrow{E}$ alors $(v,u) \in \overrightarrow{E}$. On utilisera le terme *d'arc bidirigé* pour désigner les arcs entre deux sommets u et v et un tel arc sera noté $[u,v]$. Le degré sortant d'un nœud u est noté $d^+(u)$ et le degré sortant maximum du graphe est noté $D^+(\overrightarrow{G})$. Le voisinage ouvert $N(S)$ d'un ensemble de nœuds S est défini par $N(S) = \{v \in V \mid (u,v) \in E \wedge u \in S\} \setminus S$. On définit de manière similaire le voisinage fermé $N[S] = N(S) \cup S$. Le sous-graphe induit $\overrightarrow{G}[S]$ a pour ensemble de nœuds S et pour ensemble d'arcs $\{(u,v) \mid u,v \in S \wedge (u,v) \in \overrightarrow{E}\}$. Un sous-graphe induit $\overrightarrow{H}$ est *valide* si pour toute paire de sommets u,v tel que $[u,v] \in \overrightarrow{E}(\overrightarrow{G})$ on a soit $(u,v) \in \overrightarrow{E}(\overrightarrow{H})$, soit $(v,u) \in \overrightarrow{E}(\overrightarrow{H})$ mais pas les deux en même temps. On considère ici que tout sous-graphe valide correspond à un orientation du graphe et qu'un algorithme d'orientation est un algorithme qui prend en entrée un graphe bidirigé et qui produit en sortie un sous-graphe valide. Soit $\mathcal{A}$ un algorithme d'orientation, on note $D^+_{\mathcal{A}}(\overrightarrow{G})$ le degré sortant maximum du sous-graphe valide produit par $\mathcal{A}$ avec le graphe $\overrightarrow{G}$ en entrée. Le degré sortant maximum obtenu avec un algorithme d'orientation optimal sera noté $D^+_{opt}(\overrightarrow{G})$. Le problème de *l'orientation minimum* consiste donc à trouver un algorithme d'orientation qui soit le plus proche de l'optimal. On considère également qu'un ensemble $F \subset V$ de nœuds peuvent être en panne *initialement*, c'est-à-dire qu'un nœud est soit en panne dès le début de l'algorithme et le restera, soit correct pour toute la durée de l'algorithme. Un nœud en panne peut recevoir des messages mais pas en envoyer. On note $C = V \setminus F$ l'ensemble des nœuds corrects. On considère que les arcs de $G[F]$ sont inutilisables et on cherche donc à orienter le sous-graphe $\overrightarrow{G_F} = (V_F, \overrightarrow{E_F})$ défini par $V_F = N[C]$ et $\overrightarrow{E_F} = \{(u,v) \in \overrightarrow{E} \mid u \in C \vee v \in C\}$. On remarque que tout arc (u,v) avec u un noeud correct et v un noeud en panne sera conservé dans tout sous-graphe valide de $\overrightarrow{G}$.

Lien entre orientation et densité L'orientation minimum optimale est étroitement liée avec la densité du graphe à orienter. La densité $\delta(G)$ d'un graphe $\overrightarrow{G}$ est défini par $\frac{|\overrightarrow{E}(\overrightarrow{G})|}{|V(\overrightarrow{G})|}$ et la densité maximum $\Delta(\overrightarrow{G})$ est le maximum des densités de tous les sous-graphes induits. En combinant les résultats de [AF76] et [FT14], on peut montrer que $D^+_{opt}(\overrightarrow{G}) = \left\lceil \Delta(\overrightarrow{G}) \right\rceil$. En utilisant cette notion de densité, on peut donc concevoir des algorithmes d'orientation proche de l'optimal. Plusieurs solutions exactes ou approchées du problème de l'orientation minimum existent dans un modèle centralisé [Kow06, AMOZ06] et se basent sur une réduction à un problème de flot maximum. Toutefois, il existe peu de solutions dans un environnement distribué. Barenboim et Elkin [BE10] proposent un algorithme d'orientation dans le modèle LOCAL qui nécessite $O(\log n)$ rondes de communication et fournit une $2(2+\varepsilon)$-approximation du problème si les nœuds n'ont pas d'information sur la densité du graphe. Par contre, tous les nœuds connaissent n. Farach-Colton et Tsai [FT14] proposent un algorithme $(2+\varepsilon)$-approché. L'algorithme est basé sur un mécanisme d'épluchage du graphe facilement adaptable dans un modèle de communication synchrone utilisant un nombre de rondes logarithmiques et garantissant la même approximation.

Cependant ces algorithmes sont difficilement adaptables dans un environnement asynchrone avec pannes. L'utilisation de synchroniseurs [Awe85] n'est pas envisageable car ils nécessitent de pouvoir détecter les pannes, ce qui n'est pas possible dans un mode de communication asynchrone. L'orientation minimum de graphe reste donc un problème ouvert dans ce contexte.

Enfin, Tel étudie dans [Tel94] le problème, plus faible que le notre, de l'orientation de réseaux. Il montre que ce problème nécessite soit l'existence d'un nœud distingué, soit que les nœuds aient des identifiants uniques.

2 Modèle et contributions

On considère un réseau de communication statique représenté par le graphe bidirigé $\overrightarrow{G}$ à orienter avec n nœuds qui ont chacun un identifiant unique. Il existe un nœud distingué appelé la racine et noté R. On considère un mode de communication par passage de messages et on suppose que les canaux de communi-

cation sont FIFO (les messages sont reçus dans l'ordre d'émission) et qu'il n'y a pas de perte de messages. Les communications sont asynchrones, mais pour l'analyse on supposera qu'un message est délivré en au plus une unité de temps. On définit une ronde de communication asynchrone de la manière suivante : soit t_i le temps où une ronde i commence et M_i l'ensemble des messages envoyés mais pas encore reçu au temps t_i. On considère qu'une nouvelle ronde $i+1$ ne peut pas commencer tant que les conditions suivantes ne sont pas remplies : (1) tous les messages de M_i ont été reçus (2) pour chaque message reçu au cours de la ronde, un noeud peut appliquer le traitement correspondant (3) un nœud peut envoyer (resp. recevoir) un message à tous ses voisins (de tous ses voisins). Finalement, il y a des algorithmes de routage et de diffusion connus de tous les nœuds pour communiquer avec R. On fait aussi l'hypothèse que le chemin de routage entre deux nœuds u et v est toujours identique, quel que soit le message envoyé. On note T_B le nombre de rondes maximum nécessaire pour chacune des deux opérations (routage et diffusion).

Nous proposons l'algorithme `AvrDegAsync` qui calcule un sous-graphe valide de $\overrightarrow{G}$. A notre connaissance, c'est le premier algorithme d'orientation de graphe fonctionnant de manière totalement asynchrone. Dans le modèle de panne que nous considérons, on peut résoudre le consensus (*cf.* [FLP85]) et donc élire un leader. L'hypothèse de l'existence d'un nœud racine est donc justifiée. Enfin, l'algorithme ne nécessite pas la connaissance d'un paramètre global du graphe, comme le nombre de nœuds ou bien la densité. On définit M_B comme le nombre maximum de messages échangés pour une diffusion. On prouve le théorème suivant :

Théorème 1. *Soit $\overrightarrow{G}$ un graphe bidirigé avec un ensemble de F pannes initiales et $\varepsilon > 0$. Pour $f = |F| = 0$, `AvrDegAsync`$(\overrightarrow{G}, \varepsilon)$ calcule une $2(2+\varepsilon)$-approximation de l'orientation optimale du graphe $\overrightarrow{G}$. Pour $f > 0$, après exécution de `AvrDegAsync`$(\overrightarrow{G}, \varepsilon)$ on a $d^+(u) \leq \max\{(2+\varepsilon)f, 2(2+\varepsilon)D^+_{opt}(\overrightarrow{G})\}$ pour tout nœud u. Dans tous les cas, l'algorithme s'exécute en $O((T_B + \log_s n) \cdot \log_{1+\varepsilon} \Delta)$ rondes, avec $s = \frac{2+\varepsilon}{1+\varepsilon}$. La mémoire additionnelle utilisée par nœud est de taille $O(n)$ bits et l'algorithme nécessite l'échange de $O(m + M_B \log_{1+\varepsilon} n)$ messages de taille $O(\max\{\log m, n \log n\})$ bits.*

3 Description de l'algorithme

Par la suite, on note α_R la dernière valeur de densité émise par la racine, m_r le nombre d'arcs comptabilisés par la racine, n le nombre de nœuds identifiés et n_a le nombre de nœuds activés.

Identification des nœuds Au début de l'algorithme, R ne connaît pas le nombre de nœuds du système ni le nombre d'arcs du graphe. Elle va envoyer un premier message dit de *diffusion*, noté $\langle \mathbf{br}, \alpha_R \rangle$, avec $\alpha_R = 1$ la dernière estimation de densité émise, à tous les nœuds de G. A noter que nous ne faisons pas d'hypothèse sur l'algorithme de diffusion utilisé. Lorsqu'un nœud u reçoit pour la première fois un message de diffusion, il envoie un message d'identification $\langle \mathbf{id}, u, N(u) \rangle$ à R, où $N(u)$ correspond au voisinage ouvert de u.

Diffusion et activation Lorsqu'un nœud u reçoit un message de diffusion $\langle \mathbf{br}, \alpha \rangle$, si $d(u) \leq 2(2+\varepsilon)\alpha$ alors u s'*active*. Informellement, un nœud qui s'active va prendre en charge les doubles arcs qui lui sont adjacents. Il va envoyer un message d'activation $\langle \mathbf{act}, u, d^+(u) \rangle$ à la racine R ainsi qu'à chaque voisin $v \in N(u)$. Lorsqu'un voisin v reçoit le message d'activation et qu'il n'est pas lui même activé, il va supprimer l'arc (v,u) correspondant. Son degré sortant va donc diminuer et v va tester s'il est activable. A la réception du message $\langle \mathbf{act}, u, d^+(u) \rangle$, R va mettre à jour le nombre d'arcs m_R restants non pris en charge par un nœud ainsi que la liste des nœuds activés. Lorsque la valeur $m_R/2(n-n_a)$ dépasse d'un certain seuil la valeur de α_R courante, alors α_R est mise à jour et cette valeur est diffusée à tous les nœuds.

Conflits entre nœuds Supposons maintenant que deux nœuds voisins u et v s'activent à un intervalle de temps réduit. Plus précisément, le nœud u (resp. v) s'active avant d'avoir reçu le message d'activation de v (resp. u). Les deux nœuds vont donc vouloir prendre en charge l'arc bidirigé $[u,v]$ et conserver l'arc sortant qui leur est adjacent. On dit que ces deux nœuds sont *en conflit*. Le sous-graphe ainsi obtenu ne serait pas valide. Pour palier à ce problème on choisit de manière arbitraire un des nœuds pour conserver l'arc. Dans notre cas, c'est le nœud de plus petit identifiant qui conserve l'arc correspondant. Toutefois, un message d'activation a été envoyé à la racine par chacun d'eux. Intuitivement, R va comptabiliser *deux fois* $[u,v]$ et donc sous-estimer le nombre d'arc restant. Si l'estimation de la densité est trop faible, cela peut stopper les

activations des nœuds et donc l'algorithme lui-même. Supposons que v soit le nœud de plus petit identifiant. Afin de corriger cette estimation, un message correctif $\langle \textbf{patch}, u, d^+(u) \rangle$ va être envoyé par v à R afin de l'avertir du conflit entre u et v. Lorsque R reçoit $\langle \textbf{patch}, u, d^+(u) \rangle$, il va d'abord vérifier si $\langle \textbf{act}, u, d^+(u) \rangle$ a déjà été reçu. Si ce n'est pas le cas, R va déduire $2d^+(u) - 2$ de m_R, réglant ainsi le conflit et le message d'activation envoyé par u ne sera pas pris en compte lors de sa réception. Par contre si ce message a déjà été reçu, cela signifie que le double arc $[u, v]$ a déjà été comptabilisé 2 fois, m_R étant sous-estimée de 2 unités. Dans ce cas, R va additionner 2 à m_R afin de corriger cette sous-estimation.

4 Schéma de preuve du Théorème 1

L'alqorithme que nous proposons est basé sur l'estimation de la densité du graphe et sur la diffusion de cette valeur. Cependant l'asynchronisme couplé à la présence de pannes compliquent cette estimation, car il est impossible de distinguer les nœuds en panne des nœuds corrects. La racine va donc calculer son estimation avec une connaissance partielle du graphe. Cependant, il faut pouvoir garantir que l'estimation ne sera ni trop faible (au risque de ne voir aucun nœud s'activer) ni trop élevée (sinon les activations seront trop rapides). Nous présentons les grandes idées de la preuve du Théorème 1.

Tout d'abord nous considérons qu'aucun nœud n'est en panne. Nous montrons dans un premier temps que la racine ne diffuse jamais de valeur supérieure à la densité maximum. Nous montrons ensuite que l'algorithme termine en un nombre logarithmique de diffusion ce qui nous permet de déduire facilement l'énoncé du Théorème 1 dans le cas sans panne. Dans un deuxième temps, nous considérons un ensemble F de nœuds en panne. En utilisant des arguments similaires au cas sans panne, on peut facilement déduire que si un nœud s'est activé au cours de l'algorithme, alors son degré sortant ne dépasse pas $2(2+\varepsilon)\Delta(\overrightarrow{G_F})$. On peut également montrer que $\Delta(\overrightarrow{G_F}) \leq D^+_{opt}(\overrightarrow{G_F})$ en remarquant que si les nœuds de $\overrightarrow{G_F}$ n'étaient pas en panne, on pourrait obtenir une meilleur orientation, ce qui nous permet de déduire l'énoncé. Cependant certains nœuds, même corrects, peuvent ne pas être activés au cours de l'algorithme. En effet, R calcule et diffuse une sous-estimation de la densité de $\overrightarrow{G_F}$ parce que R ne peut distinguer si un nœud est fautif ou non. Si la densité estimée puis diffusée est trop faible, aucun nœud ne s'activera. Ce scénario peut notamment arriver si certains nœuds de fort degré ont un nombre important de voisins en panne. Nous montrons que les nœuds corrects non activés à la fin de l'algorithme ont un degré sortant ne dépassant pas le nombre de fautes f, à une constante près. L'argument pour le nombre de rondes est identique à celui du cas sans panne. Concernant la mémoire, l'énoncé découle directement de la structure de données où chaque nœud maintient un nombre constant de tableaux de n bits. Enfin, le nombre de messages échangés se déduit du fait qu'il y a au plus un nombre logarithmique de diffusion et qu'un nombre constant de message est envoyé pour chaque arc bidirigé.

Références

[AF76] A. Gyarfas A. Frank. How to orient the edges of a graph ? In *Combinatorics, 18Colloq. Math. Soc. János Bolyai*, pages 352–354, 1976.

[AMOZ06] Yuichi Asahiro, Eiji Miyano, Hirotaka Ono, and Kouhei Zenmyo. Graph orientation algorithms to minimize the maximum outdegree. In *Proceedings of the 12th Computing : The Australasian Theory Symposium-Volume 51*, pages 11–20. Australian Computer Society, Inc., 2006.

[Awe85] Baruch Awerbuch. Complexity of network synchronization. *J. ACM*, 32(4) :804–823, 1985.

[BE10] Leonid Barenboim and Michael Elkin. Sublogarithmic distributed MIS algorithm for sparse graphs using nash-williams decomposition. *Distributed Computing*, 22(5-6) :363–379, 2010.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2) :374–382, 1985.

[FT14] Martin Farach-Colton and Meng-Tsung Tsai. Computing the degeneracy of large graphs. In *LATIN*, volume 8392 of *Lecture Notes in Computer Science*, pages 250–260. Springer, 2014.

[Kow06] Lukasz Kowalik. Approximation scheme for lowest outdegree orientation and graph density measures. In *ISAAC*, volume 4288 of *Lecture Notes in Computer Science*, pages 557–566. Springer, 2006.

[Tel94] Gerard Tel. Network orientation. *Int. J. Found. Comput. Sci.*, 5(1) :23–57, 1994.