Un algorithme de test pour la connexité temporelle des graphes dynamiques de faible densité

Matthieu Barjon, Arnaud Casteigts, Serge Chaumette, Colette Johnen, Yessin M. Neggaz

Un algorithme de test pour la connexité temporelle des graphes dynamiques de faible densité †

Matthieu Barjon and Arnaud Casteigts and Serge Chaumette
and Colette Johnen and Yessin M. Neggaz

LaBRI, Université de Bordeaux

Nous considérons le problème de tester si un graphe dynamique donné est temporellement connexe, *i.e.* s'il existe un chemin temporel (aussi appelé *trajet*) entre toute paire de sommets. Nous considérons une version simplifiée du problème où la dynamique est représentée par un graphe évolutif non-temporisé $\mathcal{G} = \{G_1, G_2, ..., G_\delta\}$ dont l'ensemble des sommets est invariant et les arêtes sont orientées (arcs). Deux variantes du problème sont étudiées, selon que l'on autorise la traversée consécutive d'un seul ou d'un nombre illimité d'arcs à chaque étape (trajets stricts *vs* non-stricts).

Dans le cas des trajets stricts, deux algorithmes pré-existants pour d'autres problèmes peuvent être adaptés. Cependant, nous montrons qu'une approche dédiée permet d'obtenir une meilleure complexité en temps que le premier algorithme dans tous les cas, et que le second dans certaines familles de graphes, notamment les graphes dont la densité est faible à tout instant (bien que potentiellement élevée à travers le temps). La complexité de notre algorithme est en $O(\delta\mu n)$, où δ est le nombre d'*étapes* $|\mathcal{G}|$ et $\mu = max(|E_i|)$ est le nombre maximal d'arcs pouvant exister à un instant donné. Ce paramètre est à contraster avec $m = |\cup E_i|$, l'union de tous les arcs apparaissant au cours du temps. En effet, il n'est pas rare qu'un scénario de mobilité exhibe à la fois un μ petit et un m grand. Nous caractérisons les principales valeurs charnières de δ, μ et m permettant de décider quel algorithme utiliser. Dans le cas des trajets non-stricts, pour lesquels nous ne connaissons pas d'algorithme existant, nous montrons que notre algorithme peut être adapté pour répondre à la question, et ce, toujours en $O(\delta\mu n)$.

Nos deux algorithmes construisent graduellement la fermeture transitive des trajets stricts (notée $\mathcal{G}^*_{st}$) ou non-stricts (notée $\mathcal{G}^*$) à mesure que les arcs sont examinés. Ce sont des algorithmes de type *streaming* qui sont aussi capables d'arrêter leur exécution sitôt la connexité temporelle atteinte. Un sous-produit intéressant est de rendre $\mathcal{G}^*_{st}$ et $\mathcal{G}^*$ disponibles pour de futures requêtes d'accessibilité temporelle de type source-destination.

1 Introduction

Les appareils connectés et mobiles tels que les téléphones portables, satellites, voitures ou robots forment des réseaux très dynamiques où la connexité entre nœuds évolue rapidement et continuellement. De plus, la topologie du réseau à un instant donné n'est généralement pas connexe, voire même très peu dense dans certains scenarios. Cependant, même dans ces cas extrêmes, une autre forme de connexité s'établit à travers le temps et l'espace, par le biais de communications tolérantes aux délais (mécanismes de type « *store-carry-forward* »). On parle alors de *connexité temporelle*.

Nous nous intéressons au problème de tester automatiquement si un graphe dynamique donné est temporellement connexe. Autrement dit, déterminer s'il existe un chemin temporel (*journey* en anglais, *trajet* en français) entre toute paire de nœuds dans le réseau. Une notion clé est celle de fermeture transitive des trajets, introduite dans [2]. Il s'agit d'un graphe statique orienté (même si le graphe dynamique est non-orienté) dont les arcs représentent les possibilités de trajets. De cette structure peut être déduite l'appartenance d'un graphe dynamique à plusieurs familles de graphes [4], en particulier la famille des graphes temporellement

†Une version longue en anglais est disponible sur arXiv [1]. Ce travail est partiellement subventionné par la DGA via une bourse the thèse (n° 2013 60 0074).

connexes (*i.e.,* celle dont la fermeture transitive est un graphe complet). Nous nous intéressons à deux variantes : fermeture transitive stricte ($\mathcal{G}^*_{st}$) ou non-stricte ($\mathcal{G}^*$), selon que l'on autorise la traversée consécutive d'un seul arc ou d'un nombre d'arcs illimité à chaque étape (*i.e.* trajets stricts *vs.* non-stricts).

Dans le cas des trajets stricts, plusieurs algorithmes peuvent être adaptés pour calculer $\mathcal{G}^*_{st}$. Trois de ces algorithmes sont proposés dans [3], chacun permettant de calculer les trajets optimaux d'un sommet vers tous les autres selon un critère donné (au plus tôt, au plus court, au plus rapide). N'importe lequel peut être adapté au calcul de $\mathcal{G}^*_{st}$. Le plus rapide des trois (trajets au plus tôt) a un temps d'exécution en $O(m\log\delta + n\log n)$, d'où un temps total de $O(n(m\log\delta + n\log n))$ pour tester les trajets depuis chaque sommet.

Un autre algorithme, calculant une généralisation de la fermeture transitive des trajets, a été proposé dans [6]. Cette généralisation, appelée *graphe d'accessibilité dynamique*, correspond à une fermeture transitive des trajets paramétrée par une date de départ et une durée maximale pour les trajets, ainsi qu'un délai de traversée d'arête. Il s'applique à des graphes dynamiques donnés sous la forme de TVG [5] (*time-varying graphs*), à savoir un quintuplet $\mathcal{G} = (V, E, \mathcal{T}, \rho, \zeta)$ où $\mathcal{T}$ est le domaine temporel (en l'occurrence $\mathbb{R}^+$) et ρ et ζ sont des fonctions qui renseignent sur la présence et la latence d'une arête donnée à un instant donné. L'algorithme proposé peut également être utilisé pour calculer $\mathcal{G}^*_{st}$. La complexité de cet algorithme est en $O(\delta\log\delta\, mn\log n)$.

Nous proposons une approche dédiée au calcul de la fermeture transitive (d'abord stricte) d'un graphe évolutif non-temporisé orienté $\mathcal{G} = \{(V, E_i)\}$ qui permet d'obtenir une meilleure complexité en temps que l'adaptation de [6] dans tous les cas, et que l'adaptation de [3] pour une large famille de graphes dynamiques, en particulier ceux dont la densité est faible à tout instant, bien qu'arbitrairement dense à travers le temps. La complexité de notre algorithme est en $O(\delta\mu n)$, où $\delta = |\mathcal{G}|$ est le nombre d'étapes dans $\mathcal{G}$ et $\mu = max(|E_i|)$ est le nombre maximal d'arcs pouvant exister simultanément. Comme évoqué dans le résumé, ce dernier paramètre est à contraster avec $m = |\cup E_i|$, le nombre total d'arcs pouvant exister au cours du temps, l'écart entre les deux pouvant être très grand. Dans le cas des trajets non-stricts, pour lequel nous ne connaissons pas d'algorithme existant, nous montrons que l'algorithme que nous proposons peut être adapté directement pour répondre à la question, et ce, toujours en $O(\delta\mu n)$. Cette variation repose sur une double fermeture transitive, l'une relative aux étapes de $\mathcal{G}$ (comme dans le cas des trajets stricts), l'autre relative aux arcs dans un G_i donné. Autrement dit, l'une est de nature temporelle, l'autre de nature statique, les deux étant combinées pour aboutir au résultat.

Nos deux algorithmes sont de type *streaming* et sont capables d'arrêter leur exécution sitôt la connexité temporelle atteinte. Un sous-produit de l'exécution est de rendre $\mathcal{G}^*_{st}$ et $\mathcal{G}^*$ disponibles pour d'éventuelles requêtes ultérieures de type *st*-connexité (temporelle), qui se réduisent alors à de simples requêtes d'incidence dans un graphe statique.

2 Modèle et notations

Soit $\mathcal{G}$ un graphe évolutif non-temporisé orienté $\{G_i = (V, E_i)\}$. Il existe un trajet *non-strict* de u vers v dans $\mathcal{G}$ si et seulement si il existe une suite d'arcs $e_1, e_2, ..., e_p$ reliant u à v telle que pour tout $j \in 1..p-1$, $e_j \in E_i \implies \exists E_{i' \geq i}, e_{j+1} \in E_{i'}$. Si l'inégalité $i' > i$ est stricte, on parle de trajet *strict*. L'existence d'un trajet non-strict (resp. strict) de u vers v, lorsque le contexte est implicite, est notée $u \rightsquigarrow v$ (resp. $u \overset{st}{\rightsquigarrow} v$) sans préciser le graphe $\mathcal{G}$. Ainsi, dans un trajet strict, au plus un arc peut être traversé durant une même étape i, tandis que dans un trajet non-strict, le nombre d'arêtes pouvant être traversées lors d'une étape est illimité.

La fermeture transitive (non-stricte) d'un graphe dynamique $\mathcal{G}$ est le graphe *statique* orienté $\mathcal{G}^* = (V, E^*)$ tel que $(u, v) \in E^* \Leftrightarrow u \rightsquigarrow v$. La fermeture transitive stricte de $\mathcal{G}$ est le graphe statique orienté $\mathcal{G}^*_{st} = (V, E^*_{st})$ tel que $(u, v) \in E^*_{st} \Leftrightarrow u \overset{st}{\rightsquigarrow} v$. Notez que $\mathcal{G}^*$ est orienté quelle que soit la nature (orientée ou non) des arêtes de $\mathcal{G}$, car la dimension temporelle induit sa propre orientation.

Étant donné $\mathcal{G}$, on note $\delta = |\mathcal{G}|$ le nombre d'étapes dans $\mathcal{G}$. On distingue deux paramètres pour rendre compte du nombre d'arcs dans le graphe : le nombre maximal d'arcs existant à une même étape, i.e. $\mu = max(|E_i|)$, et le nombre total d'arcs pouvant exister au cours du temps, i.e. $m = |\cup E_i|$. Bien sûr, quelque soit le graphe considéré, on a $m \geq \mu$, et même souvent $m \gg \mu$.

3 Calcul de la fermeture transitive des trajets stricts

Nous proposons ci-dessous un algorithme de calcul de la fermeture transitive stricte $\mathcal{G}^*_{st}$ dans le cas général où $\mathcal{G}$ est orienté. Le principe de l'algorithme est de construire, étape après étape, la liste de tous les prédécesseurs de chaque sommet, i.e., pour un sommet v, l'ensemble $\{u : u \overset{st}{\rightsquigarrow} v\}$. Soit $\mathcal{P}(v,t)$ l'ensemble des prédecesseurs de v à l'issue des t premières étapes (i.e. en tenant compte des ensembles d'arêtes $E_1, ..., E_t$). A l'étape i, le cœur du traitement consiste à ajouter $\mathcal{P}(u, i-1)$ à $\mathcal{P}(v,i)$ pour chaque arête $(u,v) \in E_i$. En pratique, seules deux variables $\mathcal{P}(v)$ et $\mathcal{P}^+(v)$ sont maintenues pour chaque nœud v, où $\mathcal{P}^+(v)$ contient les nouveaux prédécesseurs de v (ajoutés durant l'étape courante). Le détail des traitements est donné par l'Algorithme 1 dans la version longue du papier [1].

Lemma 1. *Pour tout $v \in V$, $|\mathcal{P}(v)| \leq \delta\mu$, i.e., un noeud ne peut avoir plus de $\delta\mu$ prédécesseurs.*

Démonstration (*par l'absurde*)**.** *S'il existe un nœud v tel que $|\mathcal{P}(v) \setminus v| > \delta\mu$, alors, par définition, il existe plus de $\delta\mu$ sommets u différents de v tels que $u \rightsquigarrow v$. Chacun de ces sommets est donc l'origine d'au moins un arc, ce qui implique que plus de $\delta\mu$ arcs distincts ont existé.* □

Theorem 1. *L'Algorithme 1 calculant la fermeture transitive stricte d'un graphe $\mathcal{G}$ a une complexité en temps en $O(\delta\mu n)$.*

*Démonstration***.** *La boucle d'initialisation est linéaire en n. Vient ensuite la boucle principale, qui itère autant de fois qu'il y a d'étapes dans $\mathcal{G}$, i.e. δ fois. Elle comporte trois sous-boucles, chacune étant dominée par $O(|E_i| \cdot n) = O(\mu n)$. Enfin, la construction de la fermeture transitive, si cette dernière n'est pas complète prématurément, consiste en une boucle qui, pour chaque nœud, itère sur ses prédécesseurs. Or, on sait que le nombre de prédecesseur d'un nœud donné ne peut excéder $\delta\mu$ (Lemme 1). Cette dernière boucle est donc elle aussi contenue dans $O(\delta\mu n)$.* □

4 Calcul de la fermeture transitive des trajets non-stricts

Dans cette section, nous nous intéressons au calcul de $\mathcal{G}^*$, i.e. la fermeture transitive des trajets où un nombre illimité d'arêtes peut être traversé à chaque étape (trajets non-stricts). Une simple observation nous permet de réutiliser l'Algorithme 1 de manière quasiment directe. En effet, la relaxation de la contrainte que les trajets sont stricts implique qu'à chaque étape i, si un chemin (au sens classique) existe de u vers v, alors u peut joindre v à cette même étape. L'algorithme consiste donc à pré-calculer, à chaque étape, la fermeture transitive (au sens classique, statique du terme) des arcs présents dans G_i, résultant en un graphe G^*_i dont les arcs correspondent aux chemins dans G_i. L'Algorithme 1, appliqué ensuite au graphe dynamique $\{G^*_i\}$, produit ainsi la fermeture transitive $\mathcal{G}^*$ des trajets *non-stricts* de $\mathcal{G}$.

La complexité en temps de cet algorithme dépend essentiellement du coût requis pour calculer la fermeture transitive statique G^*_i des graphes G_i. Cela peut être fait par une recherche en profondeur (DFS) ou en largeur (BFS), exécutée depuis chaque sommet dans G_i, chacune de ces exécution ayant un coût en $O(|E_i|) = O(\mu)$. Ainsi, le surcoût engendré par ce traitement reste confiné dans le même ordre de grandeur que nous avons identifié précédemment, à savoir $O(\delta\mu n)$.

5 Comparaison

Cette section compare la complexité de notre algorithme à celle de la stratégie utilisant le calcul des trajets au plus tôt de [3]. Cette stratégie, qui revient à exécuter l'algorithme depuis chaque sommet, a une complexité totale en $O(n(m\log\delta + n\log n))$, où m est le nombre total d'arêtes pouvant exister au cours du temps, i.e. $|\cup E_i|$, et non μ.

Il s'agit donc de comparer cet ordre de grandeur à $O(\delta\mu n)$, ou après simplification par n, de comparer $O(\delta\mu)$ à $O(m\log\delta + n\log n)$. Ces grandeurs appartiennent à un espace à quatre dimensions : μ, m, δ et n ; il n'est donc pas aisé de les comparer. Nous proposons de les étudier asymptotiquement en n, en faisant varier les rapports entre μ, m et δ. Précisément, nous faisons varier les ordres de grandeur de μ et m (densité « instantanée » *vs.* densité « cumulée ») pour plusieurs ratios de valeurs possibles entre δ et n (i.e. nombre

d'étapes dans $\mathcal{G}$ en fonction de n). Le tableau proposé (Table 1) contient 60 résultats, dont une dizaine mettent en évidence là ou a lieu le basculement entre les deux algorithmes. Pour simplifier la vérification de ces résultats, nous fournissons dans la colonne de droite une expression intermédiaire, obtenue après simple substitution de μ et m dans les deux expressions à comparer.

$\mu=\Theta(.)$	$m=\Theta(.)$	$\delta=\Theta(\log n)$	$\delta=\Theta(\sqrt{n})$	$\delta=\Theta(n)$	$\delta=\Theta(n^2)$	$\delta=\Theta(e^n)$	Calcul intermédiaire $\Theta(.)\pm\Theta(.)$
$\log n$	n^2	$-$	$-$	$-$	$\approx$	$+$	$\delta\log n \pm n^2\log\delta$
$\sqrt{n}$	n^2	$-$	$-$	$-$	$+$	$+$	$\delta\sqrt{n} \pm n^2\log\delta$
n	n^2	$-$	$-$	$-$	$+$	$+$	$\delta n \pm n^2\log\delta$
$n\log n$	n^2	$-$	$-$	$\approx$	$+$	$+$	$\delta(n\log n) \pm n^2\log\delta$
n^2	n^2	$+$	$+$	$+$	$+$	$+$	$\delta n^2 \pm n^2\log\delta$
$\log n$	$n\log n$	$-$	$-$	$-$	$+$	$+$	$\delta\log n \pm (n\log n)\log\delta$
$\sqrt{n}$	$n\log n$	$-$	$-$	$+$	$+$	$+$	$\delta\sqrt{n} \pm (n\log n)\log\delta$
n	$n\log n$	$-$	$+$	$+$	$+$	$+$	$\delta n \pm (n\log n)\log\delta$
$n\log n$	$n\log n$	$+$	$+$	$+$	$+$	$+$	$\delta \pm \log\delta$
$\log n$	n	$-$	$-$	$\approx$	$+$	$+$	$\delta\log n \pm n\log\delta + n\log n$
$\sqrt{n}$	n	$-$	$-$	$+$	$+$	$+$	$\delta\sqrt{n} \pm n\log\delta + n\log n$
n	n	$\approx$	$+$	$+$	$+$	$+$	$\delta n \pm n\log\delta + n\log n$

TABLE 1: Comparaison de la complexité en temps de notre algorithme à l'adaptation de l'algorithme de [3]. *Les cases $-$ (resp $+$, $\approx$) indiquent les plages de paramètres pour lesquelles notre solution a une complexité asymptotique plus faible (resp plus forte, du même ordre de grandeur).*

En résumé, le tableau confirme que notre solution se comporte d'autant mieux que l'écart entre densité « instantanée » et densité « cumulée » est élevé, ce qui n'est pas surprenant. Il n'est pas surprenant non plus, au vu de la présence des facteurs δ versus $\log\delta$, que notre solution soit plus efficace lorsque le nombre d'étapes est relativement faible. En outre, le tableau révèle plusieurs éventails de valeurs naturelles où notre solution se comporte mieux, comme par exemple pour les trios (μ,m,δ) vallant $(O(n),\Theta(n^2),O(n))$, ou $(O(\log n),\Omega(n\log n),O(n))$, ou bien $(O(\log n),\Omega(n),o(n))$, ou encore $(O(\sqrt{n})),\Omega(n),O(\sqrt{n}))$.

Enfin, nous pensons que l'impact coûteux du paramètre δ dans la complexité théorique de notre algorithme doit être relativisé, eu égard au fait que l'algorithme termine dès que la connexité temporelle est atteinte. En effet, si l'on considère des modèles de graphes dynamiques aléatoires tels que les graphes à évolution arête-markovienne (*Edge-Markovian Evolving Graphs*), la connexité temporelle s'établit avec forte probabilité après un nombre sous-logarithmique d'étapes. La performance de notre algorithme dans les scénarios représentés par ce type de modèle correspondrait donc, en réalité, à la colonne la plus à gauche du tableau.

Références

[1] Matthieu Barjon, Arnaud Casteigts, Serge Chaumette, Colette Johnen, and Yessin M. Neggaz. Testing temporal connectivity in sparse dynamic graphs. *CoRR*, abs/1404.7634, 2014.

[2] Sandeep Bhadra and Afonso Ferreira. Complexity of connected components in evolving graphs and the computation of multicast trees in dynamic networks. In *Proc. of ADHOCNOW'03*. Springer, 2003.

[3] Binh-Minh Bui-Xuan, Afonso Ferreira, and Aubin Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *Int. J. of Foundations of Computer Science*, 14(02) :267–285, 2003.

[4] Arnaud Casteigts, Serge Chaumette, and Afonso Ferreira. Characterizing topological assumptions of distributed algorithms in dynamic networks. In *Proc. of SIROCCO'09*, pages 126–140. Springer, 2009.

[5] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *Int. J. of Parallel, Emergent and Distributed Systems*, 27(5) :387–408, 2012.

[6] John Whitbeck, Marcelo Dias de Amorim, Vania Conan, and Jean-Loup Guillaume. Temporal reachability graphs. In *Proc. of MOBICOM'12*, pages 377–388. ACM, 2012.