Détection de composantes connexes persistantes non dominées dans un graphe dynamique

Mathilde Vernet, Yoann Pigné, Éric Sanlaville

Normandie Univ, UNIHAVRE, UNIROUEN, INSA Rouen, LITIS, 76600 Le Havre, France
{mathilde.vernet,yoann.pigne,eric.sanlaville}@univ-lehavre.fr

Mots-clés : *graphe dynamique, connexité, algorithme online.*

1 Introduction et état de l'art

La connexité d'un graphe, outre son utilité en tant que telle dans certaines applications comme les réseaux de communication, les réseaux logistiques ou les réseaux sociaux, a aussi une importance pour la résolution d'autres problèmes sur les graphes. Beaucoup de problèmes peuvent être décomposés et résolus indépendamment sur les composantes connexes du graphe, comme les problèmes de coloration par exemple. Dans certains cas, la connexité est une condition initiale indispensable à la résolution de problèmes, comme pour les flots par exemple où la source et le puits doivent se trouver dans la même composante connexe.

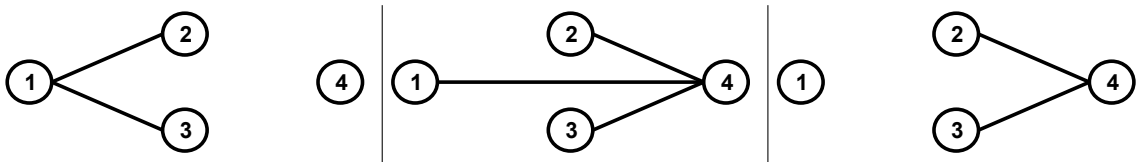
De plus, le temps est une variable importante à prendre en compte et ne pouvant pas être modélisé par un graphe statique. La logique extension d'un graphe statique est donc un graphe dynamique dans lequel les noeuds et arêtes peuvent apparaitre ou disparaitre dans le temps et les informations portées par les noeuds et arêtes peuvent dépendre du temps.

Un questionnement naturel est donc de savoir comment étendre la notion de composante connexe aux graphes dynamiques. Ce problème a été étudié dans la littérature. Bhadra et Ferreira (2003) proposent une définition se basant sur les chemins dynamiques. Dans ce travail, deux noeuds u et v sont considérés comme étant dans la même composante connexe si au cours de l'évolution du graphe, il existe une suite d'arêtes (possiblement présentes à des pas de temps différents mais dont les pas de temps de présence sont croissantes) permettant d'aller de u à v et de v à u. Gómez-Calzado *et al.* (2015) proposent une définition de composante connexe assez similaire à cette dernière, appelée Δ-composante, où le nombre de pas de temps associés au chemin dynamique est borné par Δ. Huyghues-Despointes *et al.* (2016) proposent aussi une définition basée sur les chemins dynamiques dans laquelle l'accessibilité de u à v et de v à u doit être possible par un chemin dynamique dans chaque fenêtre de temps de taille fixée de l'évolution du graphe. D'autres approches ont été proposées n'utilisant pas les chemins dynamiques, comme Casteigts *et al.* (2015) qui s'intéressent à la connexité du graphe statique résultant de l'intersection du graphe dynamique sur plusieurs pas de temps successifs, ou encore Akrida et Spirakis (2019) qui s'intéressent à l'intervalle de temps pendant lequel le graphe (ou un sous-ensemble de noeuds) reste connexe pour une date de départ donnée.

Dans ce travail, nous nous intéressons aux composantes connexes persistantes (Section 2) qui traduisent la persistance de la connexité des noeuds du graphe au cours du temps. Nous proposons un algorithme "online" polynomial pour trouver les composantes dominantes, qui ont le plus de noeuds et qui durent le plus longtemps (Section 3). Enfin, une étude expérimentale de cet algorithme est présentée (Section 4).

2 Modèle de graphe et composantes connexes persistantes

Graphe dynamique Nous considérons un graphe non orienté de n noeuds et son évolution au cours du temps. L'intervalle sur lequel est étudié le graphe est appelé *intervalle d'étude*



(a) t-graphe G_1 (b) t-graphe G_2 (c) t-graphe G_3

FIG. 1 – Évolution d'un graphe dynamique sur 3 pas de temps. Chaque t-graphe est représenté.

tandis que la fin de cet intervalle est nommé *horizon de temps*. Nous appelons *t-graphes* les graphes statiques qui représentent l'état d'un graphe à un instant donné de l'intervalle d'étude. Un graphe dynamique peut alors être défini comme une succession de t-graphes. Les noeuds du graphe restent présents pendant tout l'intervalle d'étude et les arêtes peuvent apparaître et disparaître d'un pas de temps à un autre.

Nous notons $\mathcal{T} = \{1, \ldots, T\}$ l'intervalle d'étude, T est alors l'horizon de temps. Le graphe dynamique est noté $G = (G_i)_{i \in \mathcal{T}}$ où $G_i = (V, E_i)$ est un t-graphe pour un pas de temps i donné. Tous les t-graphes d'un graphe dynamique sont définis sur le même ensemble de noeuds.

Composante connexe persistante Nous définissions une composante connexe persistante comme étant un ensemble K de k noeuds connectés directement ou via d'autres noeuds du graphe pendant l pas de temps consécutifs allant de la date $f - l + 1$ à la date f. On note la composante connexe persistante $p = (K, k, l, f)$.

Cette définition permet de rendre compte de la connexité d'un graphe au cours du temps. En effet, elle permet d'identifier les ensembles de noeuds qui restent connexes au cours de l'évolution du graphe, même si cette connexité se fait via des chemins différents et via des noeuds n'appartenant pas à la composante.

Dominance Une composante connexe persistante $p = (K, k, l, f)$ domine une autre composante $p' = (K', k', l', f')$ lorsque qu'elle possède au moins autant de noeuds mais est présente sur au moins un pas de temps de plus ($k \geq k'; l > l'$), ou lorsqu'elle est présente sur au moins autant de pas de temps mais possède au moins un noeud de plus ($l \geq l'; k > k'$), ou lorsqu'elle possède le même nombre de noeuds et est présente sur le même nombre de pas de temps mais se termine plus tôt ($k = k'; l = l'; f < f'$). Dans le cas où tous ces paramètres seraient égaux, il sera pris en compte un ordre lexicographique sur les ensembles de noeuds K et K'.

Exemple La figure 1 présente l'évolution d'un graphe dynamique à quatre noeuds sur trois pas de temps. Ce graphe n'est pas connecté à chaque pas de temps. Les noeuds 1, 2 et 3 forment une composante connexe persistante sur les deux premiers pas de temps du graphe. Les noeuds 2, 3 et 4 en forment une sur les deux derniers pas de temps du graphe, les noeuds 2 et 3 en forment une sur les trois pas de temps que constituent l'intervalle d'étude de ce graphe, et tous les noeuds du graphe en forment une sur le pas de temps 2. Les composantes formées par $\{1, 2, 3\}$ d'une part et $\{2, 3, 4\}$ d'autre part ont le même nombre de noeuds et la même durée de vie, mais la composante $p = (\{1, 2, 3\}, 3, 2, 2)$ termine avant la composante $p' = (\{2, 3, 4\}, 3, 2, 3)$ donc elle la domine.

3 Algorithme PICCNIC

L'algorithme L'objectif de l'algorithme PICCNIC (PersIstent Connected CompoNent In-Cremental Algorithm) est de calculer les composantes connexes persistantes non dominées dans un graphe dynamique donné. Il fonctionne de façon incrémentale sur chaque pas de temps du graphe et a donc calculé, à la fin de chaque itération, toutes les composantes connexes persistantes non dominées jusqu'à ce pas de temps.

L'algorithme PICCNIC, présenté dans l'algorithme 1, est composé de deux étapes pour chaque itération. La première (donnée dans l'algorithme 2) a pour but de calculer les composantes connexes persistantes présentes au pas de temps courant : celles qui commencent à ce pas de temps mais n'existaient pas au pas de temps précédent et celles qui existaient au pas de temps précédent et existent toujours au pas de de temps courant. La seconde étape de l'algorithme (donnée dans l'algorithme 3) traite les composantes terminées, c'est-à-dire qu'elle identifie les composantes qui étaient présentes au pas de temps précédent mais n'existent plus au pas de temps courant, puis ne conserve que les composantes dominantes.

Plusieurs ensembles sont utilisés pour conserver les composantes connexes persistantes. L'ensemble PCC_n contient, à chaque pas de temps i, les composantes qui sont présentes sur le t-graphe G_i. L'ensemble PCC_c se souvient, au début d'une itération, des composantes présentes à l'itération précédente, et contient à la fin d'une itération les composantes courantes. L'ensemble PCC_o contient, à chaque itération, les composantes qui viennent de se terminer. Enfin, l'ensemble PCC_f contient, à chaque itération, les composantes connexes persistantes terminées et non dominées.

Trois sous-procédures sont utilisées par ces différents algorithmes. *SuppressDouble*, utilisé entre les deux étapes de PICCNIC, permet de supprimer les composantes pour lesquelles il en existe déjà une ayant le même ensemble de noeuds mais plus de pas de temps de présence. Les fonctions booléennes *DomEarlier* et *DomLaterEqual* utilisées dans la seconde étape de PICCNIC permettent de déterminer si la première composante passée en paramètre domine la seconde (si elle termine plus tôt dans le cas de *DomEarlier*, ou plus tard ou au même moment pour *DomLaterEqual*).

Exemple Observons l'exécution de l'algorithme PICCNIC sur le graphe de la Figure 1. Dans la première itération de l'algorithme, G_1 (Figure 1a) est pris en compte. Ce t-graphe a deux composantes connexes, dont un singleton qui n'est pas considéré. L'ensemble PCC_c est initialisé à $\{(\{1,2,3\},3,1,1)\}$ et PCC_f à l'ensemble vide.

La deuxième itération traite le t-graphe G_2 (Figure 1b) qui n'est composé que d'une seule composante connexe. La composante connexe persistante formée par les noeuds 1, 2 et 3 existe toujours donc elle vieillit et une nouvelle composante formée par tous les noeuds du graphe apparait. À la fin de l'itération, $PCC_c = \{(\{1,2,3\},3,2,2), (\{1,2,3,4\},4,1,2)\}$. On a toujours $PCC_f = \emptyset$ car aucune composante persistante n'est encore terminée.

La troisième itération traite le t-graphe G_3 (Figure 1c) qui possède deux composantes connexes dont un singleton qui n'est pas pris en compte. La composante formée par les noeuds 1, 2 et 3 ainsi que celle formée par les noeuds 1, 2, 3 et 4 n'existent plus car le noeud 1 est déconnecté. Mais la composante persistante formée par les noeuds 2, 3 et 4 apparaît avec une durée de vie de 2 et vient de la composante persistante $(\{1,2,3,4\},4,1,2)$. De la même manière, la composante formée par les noeuds 2 et 3 apparaît avec une durée de vie de 3 et vient de la composante persistante $(\{1,2,3\},3,2,2)$. À la fin de l'itération, on a donc $PCC_c = \{(\{2,3\},2,3,3)$, $(\{2,3,4\},3,2,3)\}$ et $PCC_f = \{(\{1,2,3\},3,2,2)$, $(\{1,2,3,4\},4,1,2)\}$.

La quatrième et dernière itération permet de traiter les composantes encore actives au pas de temps 3. Toutes les composantes sont à présent terminées. La composante persistante $(\{2,3,4\},3,2,3)$ est dominée par la composante persistante $(\{1,2,3\},3,2,2)$ donc elle n'est pas conservée. L'ensemble final de composantes connexes persistantes non dominées obtenu pour ce graphe est donc $PCC_f = \{(\{1,2,3\},3,2,2)$, $(\{1,2,3,4\},4,1,2)$, $(\{2,3\},2,3,3)\}$.

Correction Il est possible de prouver que PICCNIC est correct en prouvant tout d'abord, par induction, qu'à chaque pas de temps $t \leq T$, toute composante connexe persistante dominante terminé est présente soit dans l'ensemble des composantes courantes (PCC_c) si elle se termine exactement au temps t, soit dans l'ensemble des composantes terminées (PCC_f) si elle s'est terminé avant. Ensuite, on prouve que l'ensemble des composantes terminées PCC_f ne contient que des composantes dominantes. Pour cela, on suppose qu'il existe une composante dominée dans l'ensemble et montre que ce n'est pas possible.

Complexité En utilisant la manière dont l'ensemble PCC_c, qui contient les composantes courantes, est construit, on peut montrer qu'à la fin de chaque itération de l'algorithme, il ne contient que des ensembles soit strictement inclus les uns dans les autres soit strictement disjoints. Par exemple, sur le graphe de la Figure 1, à une itération donnée de l'algorithme, il est impossible d'avoir les composantes persistantes composées des noeuds $\{1, 2, 3\}$ et $\{2, 3, 4\}$ cohabitant au même pas de temps. En revanche, il est possible d'avoir des composantes formées des noeuds $\{1, 2, 3\}$ et $\{1, 2, 3, 4\}$ présentes en même temps, comme c'est le cas à la deuxième itération de l'algorithme. En utilisant ce résultat, on montre que la taille de l'ensemble PCC_c est bornée par le nombre de noeuds du graphe.

Après avoir prouvé que l'ensemble PCC_f ne contient que des composantes non dominées, il est aisé de prouver que la taille de cet ensemble est aussi bornée par le nombre de noeuds du graphe.

En utilisant ces bornes sur la taille des ensembles manipulés, on peut prouver que la complexité d'une itération est $O(n^2)$. L'algorithme PICCNIC comporte $T + 1$ itérations. La complexité totale est donc $O(n^2 \cdot T)$.

Algorithme 1 Algorithme $PICCNIC$

Input : Graphe dynamique G, intervalle d'étude $\mathcal{T} = \{1, \ldots, T\}$
Output : Composantes connexes persistantes dominantes

1: **for all** $i \in \{1, \ldots, T + 1\}$ **do** ▷ Boucle sur le nombre de pas de temps
2: $G' = G_i$ ▷ Récupère le graphe pour le $i^{ème}$ pas de temps, rien si $i = T + 1$
3: $CC =$ ensemble de composantes connexes de G' de taille ≥ 2
4: **if** i=1 **then** ▷ Toutes les CCs sont des PCCs en cours
5: $PCC_c = CC$
6: $PCC_f = \emptyset$
7: **else**
8: $PCC_n = PICCNIC_{Step1}(CC, PCC_c)$
9: $SuppressDouble(PCC_n)$
10: $PCC_f = PICCNIC_{Step2}(PCC_n, PCC_c)$
return PCC_f

4 Étude expérimentale

Génération de graphes dynamiques Nous avons testé notre algorithme sur des graphes dynamiques générés aléatoirement. Afin de tester l'impact du type de graphe sur les composantes connexes présentes, nous avons fait des tests sur quatre types de graphes : les graphes aléatoires (correspondant au modèle d'Erdős-Rényi (Erdős et Rényi, 1960), appelés *random* dans la suite comme le nom du générateur utilisé), les graphes réguliers (qui sont des tores, appelés *grid* dans la suite comme le nom du générateur utilisé), les graphes scale-free (utilisant le modèle de Barabasi-Albert (Albert et Barabási, 2002) et appelés par ce nom dans la suite comme le générateur utilisé), et les graphes aléatoire géométriques (appelés *random euclidean* dans la suite comme le nom du générateur utilisé). Le modèle de Barabasi-Albert est particulièrement adapté pour les réseaux sociaux par exemple. Les graphes random eucliean sont, quant à eux, adaptés pour les applications avec une dimension spatiale comme les réseaux logistiques par exemple.

Une fois le graphe sous-jacent (V, E) généré avec un nombre de noeuds et un degré moyen donné, nous ajoutons de la dynamique sur les arêtes grâce à une chaine de Markov qui donne la probabilité pour chaque arête de E d'être présente ou absente à un pas de temps en fonction de sa présence ou absence au pas de temps précédent.

Nous avons choisi d'observer les résultats de l'algorithme ainsi que son temps d'exécution sur des graphes avec 1000 pas de temps, 1000 noeuds, un degré moyen des noeuds de 4. La probabilité pour une arête de rester présente, qui correspond à la probabilité stationnaire de

Algorithme 2 $PICCNIC_{Step1}$

Input : CC, PCC_c
Output : PCC_n, l'ensemble de potentielles nouvelles PCCs

$PCC_n = \emptyset$
for all $c \in CC \& |c| > 1$ **do**
  $Pers = FALSE$ ▷ Si c est déjà une PCC ou non
  for all $p \in PCC_c$ **do**
    if $p = c$ **then**
      $Pers = TRUE$
      $p' = p$
    else
      $p' = p \cap c$
    if $|p'| \geq 2$ **then**
      $l(p') = l(p) + 1$
      $f(p') = i$
      $PCC_n = PCC_n \cup \{p'\}$
  if $\neg Pers$ **then** ▷ Ajout des nouvelles composantes si elles ne sont pas déjà incluses
    $PCC_n = PCC_n \cup \{c\}$
    $l(c) = 1$
    $f(c) = i$
return PCC_n

Algorithme 3 $PICCNIC_{Step2}$

Input : PCC_n, PCC_c
Output : PCC_f mis à jour

$PCC_o = PCC_c - PCC_n$ ▷ Ensemble des PCCs terminées
$PCC_c = PCC_n$
for all $p \in PCC_o$ **do**
  for all $p' \in PCC_o \& p \neq p'$ **do**
    if $DomLaterEqual(p', p)$ **then**
      $PCC_o = PCC_o - \{p\}$
      STOPLOOP
    else
      if $DomLaterEqual(p, p')$ **then**
        $PCC_o = PCC_o - \{p'\}$
for all $p \in PCC_o$ **do**
  for all $p' \in PCC_f$ **do**
    if $DomEarlier(p', p)$ **then**
      $PCC_o = PCC_o - \{p\}$
      STOPLOOP ▷ p est dominé par une PCC terminée
    else
      if $DomLaterEqual(p, p')$ **then**
        $PCC_f = PCC_f - \{p'\}$
$PCC_f = PCC_f \cup PCC_o$
return PCC_f

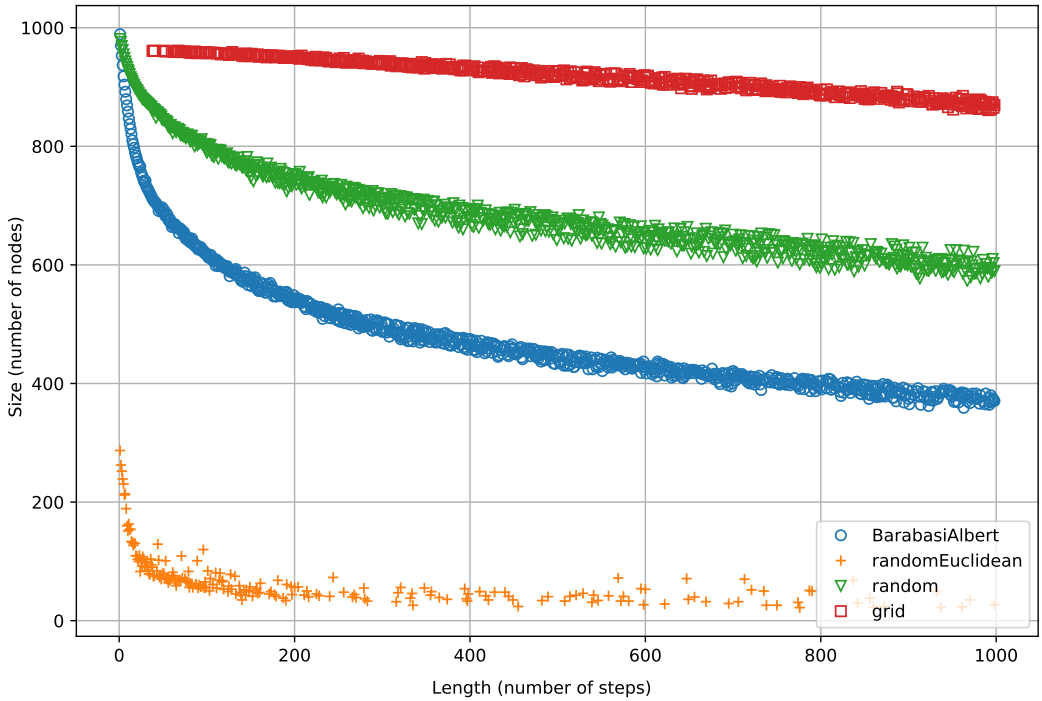


FIG. 2 – Valeurs moyenne du front de Pareto résultat de l'algorithme PICCNIC pour chaque type de graphe.

présence d'une arête, a été fixée à 0.9. La probabilité pour une arête de rester absente correspond à la probabilité stationnaire d'absence d'une arête. L'algorithme est testé 10 fois avec le même jeu de paramètres afin d'avoir une meilleure valeur statistique des résultats. D'autres jeux de paramètres ont été expérimentés pour lesquels des résultats similaires ont été obtenus.

Les calculs ont été fait sur une machine virtuelle disposant d'un processeur Inter Core 64 bits avec 8 coeurs, 4 Mo de cache, 2 GHz de fréquence et 96 Go de RAM. L'algorithme a été programmé en utilisant la bibliothèque de graphes Java GraphStream (Dutot *et al.*, 2007).

Résultats de PICCNIC La figure 2 représente la taille moyenne, en nombre de noeuds, d'une composante connexe persistante dominante en fonction de sa durée de vie en nombre de pas de temps de présence. On remarque que les graphes random euclidean n'ont pas de composantes persistantes avec un nombre important de noeuds.

Dans les graphes Barabasi-Albert, la taille d'une composante persistante diminue drastiquement lorsque sa durée de vie augmente. Cela s'explique par le fait que ce type de graphes possède beaucoup de noeuds avec un très faible degré. Ces noeuds seront donc facilement déconnecté du reste du graphe lorsque les arêtes apparaissent et disparaissent.

Les graphes random quant à eux ont des composantes persistantes de taille importante et ont même des composantes persistantes "géantes" même si elles ne sont pas présentes sur beaucoup de pas de temps.

Les grilles ont beaucoup de composantes persistantes "géantes" et celles-ci ont une durée très importante comparé à l'intervalle d'étude du graphe. Ce type de graphe est très robuste et ne se déconnecte pas facilement.

Temps d'exécution Nous avons aussi observé le temps d'exécution de l'algorithme en fonction du nombre de noeuds du graphe. Pour cela, nous avons fait varier le nombre de noeuds de 100 à 4500 et avons conservé les mêmes valeurs pour les autres paramètres. Les résultats, donnés en secondes, sont présentés dans la Figure 3 pour chaque type de graphes. Une fonction de régression de la forme n^2 a été calculée et représentée pour chaque type de graphe. La forme

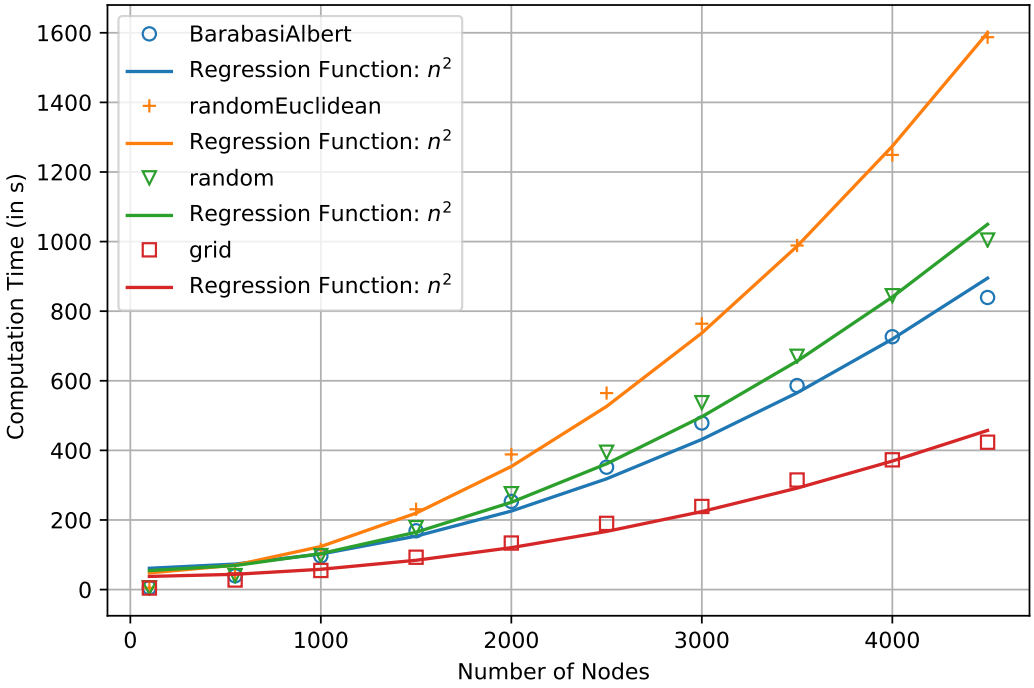


FIG. 3 – Temps d'exécution médian de l'algorithme PICCNIC en fonction du nombre de noeds du graphe, pour chaque type de graphe.

de la fonction correspond bien à la complexité expérimentale de l'algorithme PICCNIC quand T est fixé.

La valeur R^2 de la régression vaut 0.984 pour les graphes Barabasi-Albert, 0.979 pour les grilles, 0.992 pour les graphes aléatoires et 0.997 pour les graphes random euclidean. Ces valeurs prouvent que les résultats expérimentaux correspondent à ceux attendus et l'exécution est cohérente avec la complexité théorique de l'algorithme. Les temps d'exécution sont assez faibles (moins d'une demie heure de calcul pour trouver toutes les composantes persistantes non dominées sur un graphe random euclidean de 4500 noeuds), ce qui rend cet algorithme utilisable en pratique.

Cet algorithme est beaucoup plus rapide sur les grilles que sur les autres types de graphes. En comparant cela avec les résultats de l'algorithme Figure 2, on confirme que l'algorithme prend plus de temps pour les graphes qui présentent de nombreuses composantes persistantes de petite taille.

5 Conclusion

Dans ce travail, nous avons proposé une définition permettant d'étendre la notion de connexité et de composante connexe aux graphes dynamiques. Un algorithme capable de calculer les composantes connexes persistantes non dominées a été présenté. Cet algorithme a une complexité polynomiale ($O(n^2 \cdot T)$). Il fonctionne "online" donc l'évolution du graphe dynamique n'a pas besoin d'être connue à l'avance pour l'utiliser. Il donne, à chaque pas de temps, les composantes connexes persistantes non dominées pour l'évolution du graphe jusqu'à ce pas de temps. Nous avons également proposé une étude expérimentale, qui a permis de montrer que le temps d'exécution de l'algorithme est cohérent avec la complexité théorique de celui-ci et que les temps d'exécution restent raisonnables pour des graphes à plus de 4000 noeuds.

Bien que la définition de composante connexe persistante présentée ici s'applique aux graphes non-orientés, il est tout à fait possible de l'étendre aux graphes orientés. L'algorithme PICCNIC pourrait être utilisé dans ce cas.

Nous considérons l'âge des composantes en utilisant le nombre de pas de temps sur lesquels elles sont présentes. Il est cependant envisageable de considérer qu'un pas de temps du graphe correspond à l'intervalle de temps entre deux modifications de celui-ci. Dans ce cas, il faudrait considérer l'âge des composantes, non pas en comptant le nombre de pas de temps de présence, mais en calculant la différence de temps réel entre l'apparition et la disparition de ladite composante. Et sous conditions de modifications mineures à l'algorithme PICCNIC, il pourrait être utilisé de cette façon.

En supposant une définition plus souple des composantes connexes persistantes où la connexité ne se ferait pas sur des pas de temps successifs, une même composante pourrait être interrompue et recommencer plus tard. Bien qu'une telle définition puisse trouver des applications intéressantes, le problème n'est alors plus polynomial. En effet, tant que l'on ne connait pas la date à laquelle se termine une composante, on ne peut pas tester sa dominance par rapport aux autres composantes. De ce fait, tous les sous-ensembles de noeuds, de cardinalité supérieure strictement à 1, devraient être considérés comme potentiellement dominants tant que le graphe n'est pas exploré jusqu'à son horizon de temps. Donc dans le pire des cas, $2^n - n - 1$ sous-ensembles devraient être conservés. Ce n'est donc pas faisable en pratique.

Références

Eleni C Akrida et Paul G Spirakis : On verifying and maintaining connectivity of interval temporal networks. *Parallel Processing Letters*, 29(02):1950009, 2019.

Réka Albert et Albert-László Barabási : Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

Sandeep Bhadra et Afonso Ferreira : Complexity of connected components in evolving graphs and the computation of multicast trees in dynamic networks. *In International Conference on Ad-Hoc Networks and Wireless*, pages 259–270. Springer, 2003.

Arnaud Casteigts, Ralf Klasing, Yessin M Neggaz et Joseph G Peters : Efficiently testing t-interval connectivity in dynamic graphs. *In International Conference on Algorithms and Complexity*, pages 89–100. Springer, 2015.

Antoine Dutot, Frédéric Guinand, Damien Olivier et Yoann Pigné : Graphstream : A tool for bridging the gap between complex systems and dynamic graphs. *In Emergent Properties in Natural and Artificial Complex Systems. Satellite Conference within the 4th European Conference on Complex Systems (ECCS'2007)*, 2007.

Paul Erdős et Alfréd Rényi : On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

Carlos Gómez-Calzado, Arnaud Casteigts, Alberto Lafuente et Mikel Larrea : A connectivity model for agreement in dynamic systems. *In European Conference on Parallel Processing*, pages 333–345. Springer, 2015.

Charles Huyghues-Despointes, Binh-Minh Bui-Xuan et Clémence Magnien : Forte Δ-connexité dans les flots de liens. *In ALGOTEL 2016-18èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, 2016.