Grands réseaux complexes : mettre de l'ordre dans les triangles

F. Lécuyer [1], *L. Jachiet* [2], *C. Magnien* [1], *L. Tabourier* [1]. *(1) Sorbonne Université, CNRS, LIP6, F-75005 Paris, France, (2) LTCI, Télécom Paris, Institut Polytechnique de Paris, fabrice.lecuyer@lip6.fr*

La structure des réseaux complexes peut être caractérisée par de nombreux indicateurs, dont l'un des principaux est le nombre de triangles. Connaître la position des triangles permet de mesurer le regroupement des nœuds (clustering), ce qui a été utilisé dans la littérature pour décrire la structure du cerveau, l'évolution des réseaux sociaux, les architectures logicielles ou les citations scientifiques. Nous proposons une méthode qui accélère le listage des triangles jusqu'à 30 % sur de grands réseaux complexes réels.

Pour lister les triangles dans des réseaux de millions de nœuds, des algorithmes efficaces sont nécessaires. La méthode de référence, proposée dès 1985 par Chiba et Nishizeki [1], consiste à ordonner les nœuds : on donne un indice à chaque nœud et on oriente les arêtes des petits vers les grands indices. Ainsi, un nœud u n'a plus seulement un degré $d(u)$, mais un degré entrant $e(u)$ et un degré sortant $s(u)$. Les triangles peuvent alors être cherchés dans une seule direction, du plus petit au plus grand indice, ce qui accélère leur listage et permet de donner des bornes théoriques à la complexité des algorithmes.

Ortmann et Brandes [2] ont montré que la plupart des publications utilisant cette méthode correspondent à deux algorithmes génériques : A++ qui a une complexité asymptotique C++ = $\sum_{\text{nœud } u} s(u)^2$, et A+– qui a une complexité C+– = $\sum_{\text{nœud } u} s(u) \cdot e(u)$. Comme le montre la Figure 1, un même réseau, initialement non orienté, peut avoir différents coûts selon l'ordre qu'on lui applique. Pour réduire C++, il faut un ordre où les nœuds de grand degré ont surtout des voisins entrants, donc sont à la fin de l'ordre. Pour réduire C+–, en revanche, il faut que les nœuds aient soit un petit degré sortant, soit un petit degré entrant. Les deux algorithmes nécessitent donc différents ordres.

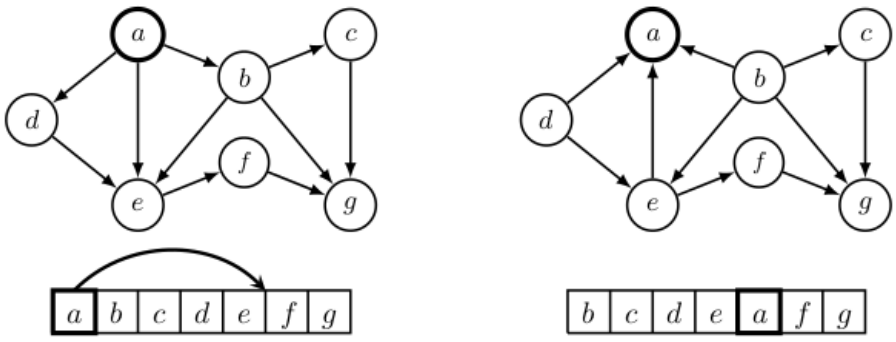

Figure 1: Exemple de réseau avec deux ordres qui ne diffèrent que par la position du nœud *a*. Le coût C++ vaut 22 à gauche et 30 à droite. Le coût C+– vaut 9 à gauche et 6 à droite, et pourrait même être réduit à 3 avec l'ordre « b c d a f e g ».

Pourtant, les seuls ordres étudiés dans [2] sont des classements par degré ou par dégénérescence (une mesure de la densité locale). Leurs expériences indiquent que le listage de triangles le plus rapide est obtenu avec A++ et l'ordre par degrés croissants. Danisch, Balalau et Sozio [3] ont pour leur part trouvé que A+– était plus rapide avec l'ordre par dégénérescence.
Ces deux ordres bornent le degré sortant s des nœuds, mais pas le degré entrant e : ils sont donc mieux adaptés pour C++ que pour C+–. De plus, ils sont assez basiques, dans le sens qu'ils proviennent d'autres problèmes algorithmiques et ne sont pas forcément adaptés à la question des triangles. À l'inverse, nous cherchons des ordres spécifiques pour diminuer le coût C+–, soit le temps d'exécution approximatif de l'algorithme A+–. Comme le montre la Table 1, cette démarche comble un vide dans les résultats existants.

Ordres considérés	Algorithme A++ (complexité C++)	Algorithme A+– (complexité C+–)
Faible coût C++ (degré, dégénérescence)	[2]	[3]
Faible coût C+– (nos nouveaux ordres)	–	notre contribution

Table 1: Combinaisons d'algorithme et d'ordre pour lister les triangles. La littérature utilise des algorithmes de complexité C++ ou C+– , mais seulement des ordres de bas coût C++. Nous proposons donc de combler ce manque par de nouveaux ordres réduisant C+–.

Nous avons prouvé que minimiser exactement C+– ou C++ est NP-difficile. Pour trouver des ordres à bas coût en un temps raisonnable, nous avons implémenté deux heuristiques. L'une considère chaque nœud un par un et calcule la meilleure position qu'il peut prendre dans son voisinage ; l'ordre obtenu a un très bas coût C+–. L'autre privilégie la rapidité : les nœuds sont pris par degré décroissant et placés alternativement au début ou à la fin de l'ordre. Le code est disponible librement et simple à utiliser : https://github.com/lecfab/volt .
Testée sur des réseaux réels jusqu'à cent millions de nœuds et deux milliards d'arêtes, notre méthode réduit le coût C+– de plus de 50 % par rapport aux ordres par degré ou dégénérescence. Cela se traduit directement dans le temps d'exécution : en moyenne, le listage des triangles est accéléré de 30 % si on ignore le temps de l'heuristique, et de 15 % si on l'inclut.
Au-delà des triangles, nos résultats montrent que l'on peut directement travailler sur le coût des algorithmes au lieu de borner leur complexité par une constante spécifique au réseau étudié. Cette méthode est donc susceptible d'améliorer le listage d'autres motifs tels que les cliques, mais il faudra alors définir de nouveaux coûts et de nouveaux ordres adaptés.

Références

[1] N. Chiba, T. Nishizeki, *Arboricity and subgraph listing algorithms* (1985).
[2] M. Ortmann, U. Brandes, *Triangle listing algorithms: back from the diversion* (2014).
[3] M. Danisch, O.D. Balalau, M. Sozio, *Listing k-cliques in sparse real-world graphs* (2018).