Théorie des Graphes

Évaluation de différents algorithmes de calcul des composantes fortement connexes

Couvreur Alexis

Table des matières

1 Description de l'environnement

Le programme d'évaluation a été fait sous Java 1.8, il nécessite une augmentation de la taille de la pile pour fonctionner ($-Xss4m$). Un processeur IntelCore i7 4 cœurs physiques à 4.3 GHz chacun , mais comme le programme n'est pas parallélisé, le nombre de cœurs importe peu.

2 Description de l'implémentation des différents modèles de données

Les graphes sont des graphes orientés à matrice d'adjacence de taille $V*V$. Le nombre de nœuds et d'arcs est gardé en mémoire respectivement dans les variables V et E. Tous les nœuds et arcs sont donc déduits de la matrice d'adjacence et aucune classe n'a été créée pour les représenter, il s'agit seulement d'entiers tel que $\forall_v, v \in [0, V[$. Ainsi on a comme coût :

- Est-ce que i et j sont reliés ? : $\mathcal{O}(V)$
- Combien i a-t-il de voisins ? : $\mathcal{O}(V)$
- Quels sont les voisins de i ? : $\mathcal{O}(V)$
- Ajouter l'arc *(a,b)* : $\mathcal{O}(1)$
- Supprimer l'arc *(a,b)* : $\mathcal{O}(1)$
- Combien y a-t-il de nœuds ? : $\mathcal{O}(1)$
- Combien y a-t-il d'arcs ? : $\mathcal{O}(1)$

Ce qui nous intéresse le plus ici est donc de pouvoir ajouter ou supprimer des arcs, car pour évaluer les algorithmes, on va augmenter le nombre d'arcs uniquement pour un cas de test. On augmentera la taille du graphe (son nombre de nœuds) pour comparer l'effet de la taille du graphe sur les algorithmes.

3 Description de l'implémentation des différents algorithmes

3.1 Algorithmes de calcul de composantes fortement connexes

Les algorithmes sont généralisés à travers une interface **SCC** et implémentent les méthodes :

- *boolean isStronglyConnected(int v, int w)* : renvoie vrai si v et w sont dans la même composante fortement connexe, renvoie faux sinon.
- *int getCount()* : renvoie le nombre de composantes fortement connexes.
- *int id(int v)* : renvoie l'id de la composante fortement connexe.

3.2 Algorithme de Tarjan

3.2.1 Complexité

La complexité de l'algorithme de Tarjan est de $\mathcal{O}(|V| + |E|)$.

3.2.2 Implémentation

Implémenté en suivant le pseudo code du cours :

- Une *pile P*
- Un *tableau booléen dans_P* de taille V, cela évite de faire appel à la méthode *contains* de *Stack* qui est en $\mathcal{O}(N)$ tandis que le tableau est en $\mathcal{O}(1)$
- Chaque sommet a des propriétés : son numéro, et son numéro accessible
- Une partition représentant les différentes CFC.

3.2.3 Appréciation

L'algorithme de Tarjan n'est pas instinctif mais n'est pas très compliqué à comprendre, ce qui rend son implémentation plus simple et agréable.

3.3 Algorithme de Kosaraju

3.3.1 Complexité

La complexité de l'algorithme de Kosaraju est de $\mathcal{O}(|V| + |E|)$.

3.3.2 Implémentation

Implémenté en suivant le pseudo code du cours :

- Une *pile S* : les sommets visités depuis un sommet v au travers d'une dfs dans le reverse graphe
- Un *tableau booléen visited* de taille V, qui permet de savoir si un sommet a été visité

3.3.3 Appréciation

L'algorithme de Kosaraju est simple à comprendre, il est très élégant car on utilise l'inverse du graphe, ce qui lui coûte beaucoup en performance.

3.4 Algorithme de Gabow

3.4.1 Complexité

La complexité de l'algorithme de Gabow est de $\mathcal{O}(|V|^3)$.

3.4.2 Implémentation

Implémenté en suivant le pseudo code du cours :

- Une *pile S* : pile des sommets non affectés à une CFC
- Une *pile P* : pile des sommets dont on ne sait pas si ils appartiennent à une CFC différentes de celles trouvées
- Chaque sommet a son numéro (tableau I dans le code, il contiendra à la fin l'id de la CFC)

3.4.3 Appréciation

L'algorithme de Gabow n'est pas évident à comprendre et difficile à mettre en œuvre, mais est relativement très puissant.

4 Évaluation des différents algorithmes

4.1 Jeu de données

Un graphe G avec V={5000, 10000, 25000} et E=0 en premier lieu. Nous cherchons à évaluer le temps de calcul des composantes fortement connexes en fonction de la densité du graphe G. On suppose que plus le graphe sera dense, plus le temps de calcul sera élevé. Pour évaluer nous allons effectuer X itérations dans lesquelles nous allons augmenter le nombre d'arcs tel que

Data: Integer X : nombre d'itérations, DirectedGraph G
for $i \in X$ **do**
  $sauvegarderTemps(Tarjan(G))$;
  $sauvegarderTemps(Kosaraju(G))$;
  $sauvegarderTemps(Gabow(G))$;
  pour chaque arc de G **for** $v \in G.V$ **do**
    ajouter X arcs;
    for $x \in X$ **do**
      ajouter un arc entre v et $x * i$ dans G;
    end
  end
end

ce qui nous donne une densité de graphe allant de 0 (à la première itération) à 1 (à la dernière itération) et donc de V composantes fortement connexes à 1.

Voici le nombre de composantes fortement connexes en fonction de la densité :

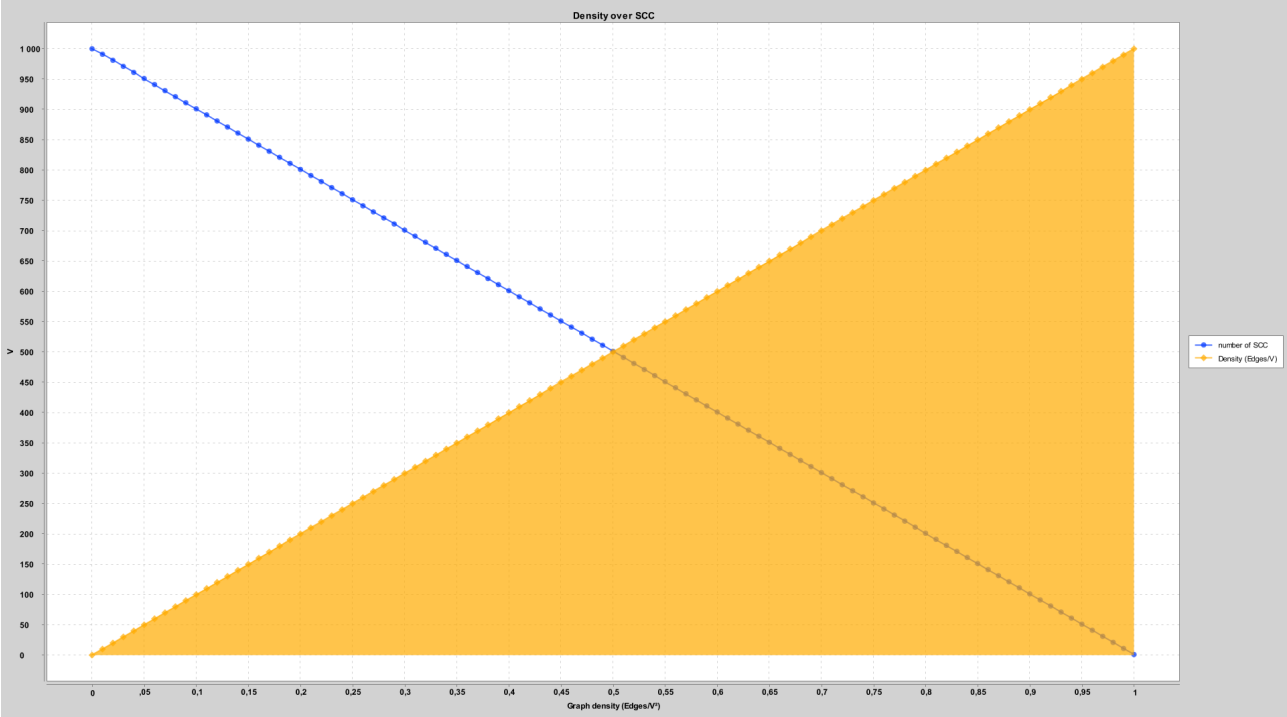

Image 1 – Rapport entre le nombre de CFC dans le graphes et la densité du graphe

4.2 Résultats

4.2.1 V=5000

Soit un total de 25 000 000 d'arcs lorsque la densité vaut 1.

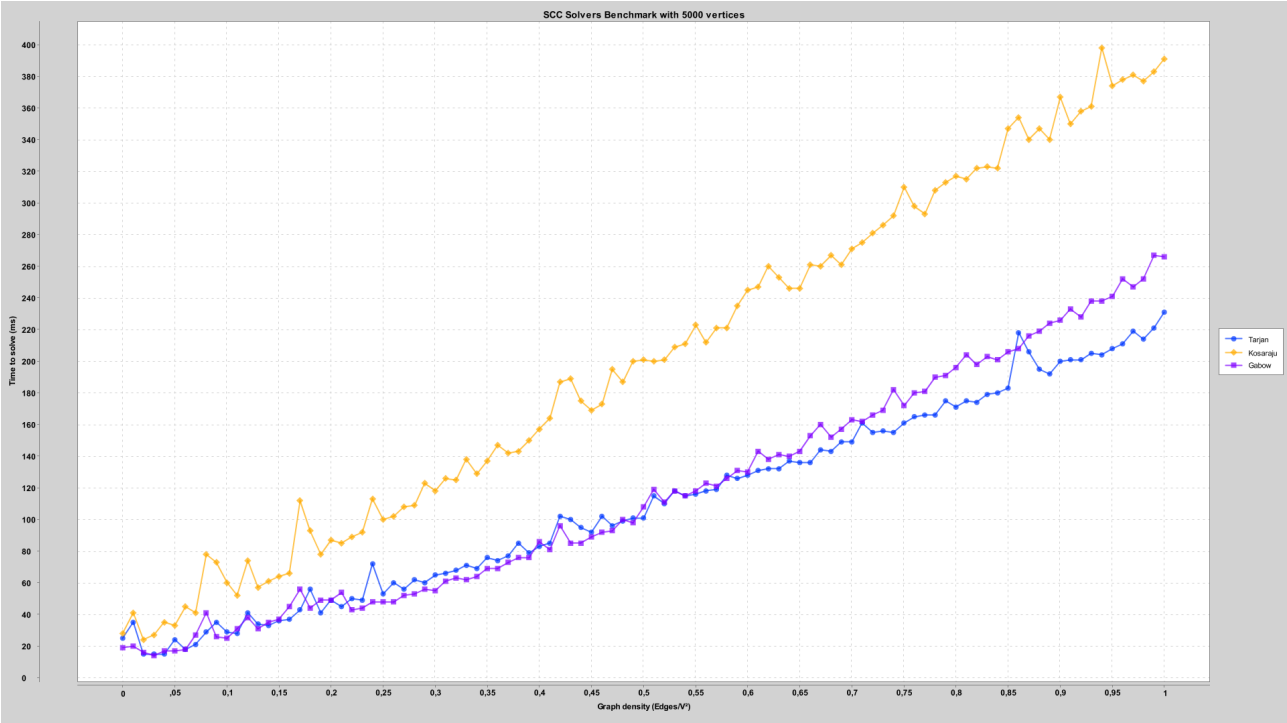

Image 2 – Évaluation de différents algorithmes de calcul de composantes fortement connexes en fonction de la densité d'un graphe à 5000 nœuds

4.2.2 V=10000

Soit un total de 100 000 000 d'arcs lorsque la densité vaut 1.

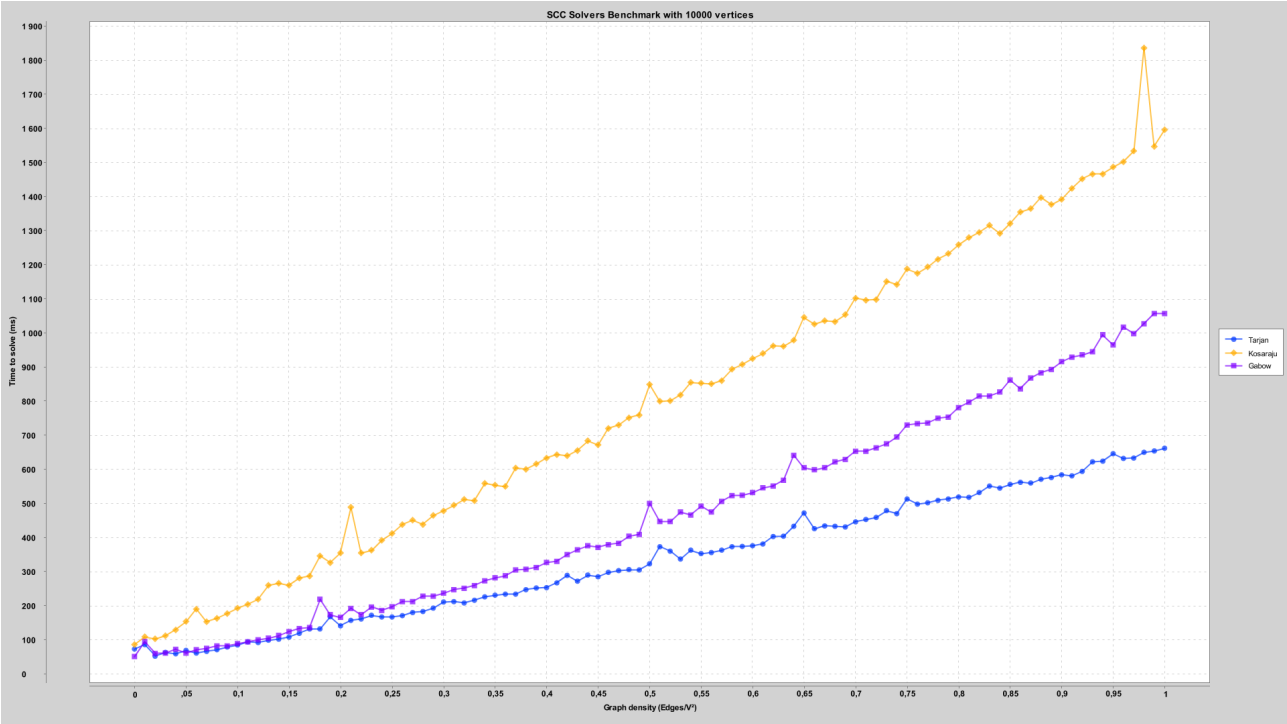

Image 3 – Évaluation de différents algorithmes de calcul de composantes fortement connexes en fonction de la densité d'un graphe à 10000 nœuds

4.2.3 V=25000

Soit un total de 625 000 000 d'arcs lorsque la densité vaut 1.

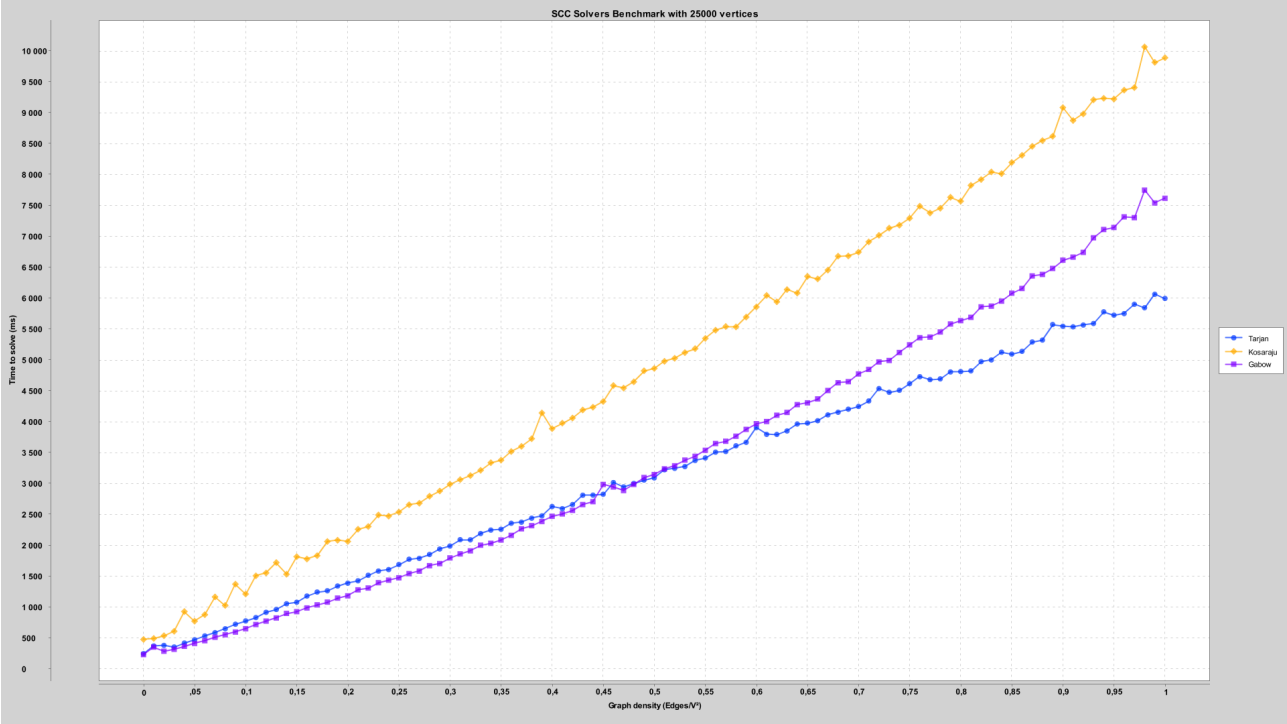

Image 4 – Évaluation de différents algorithmes de calcul de composantes fortement connexes en fonction de la densité d'un graphe à 25000 nœuds

4.3 Analyse des résultats

Classement des algorithmes de calcul de CFC par performance :

1. Tarjan ;
2. Gabow ;
3. Kosaraju.

Tarjan et Gabow se valent sur des graphes qui ne sont pas trop dense, après cela Tarjan est plus performant que Gabow, Kosaraju ne peut malheureusement pas entrer en concurrence avec les autres. On retiendra que les algorithmes élégants et intuitifs ne sont pas forcément les plus performants !