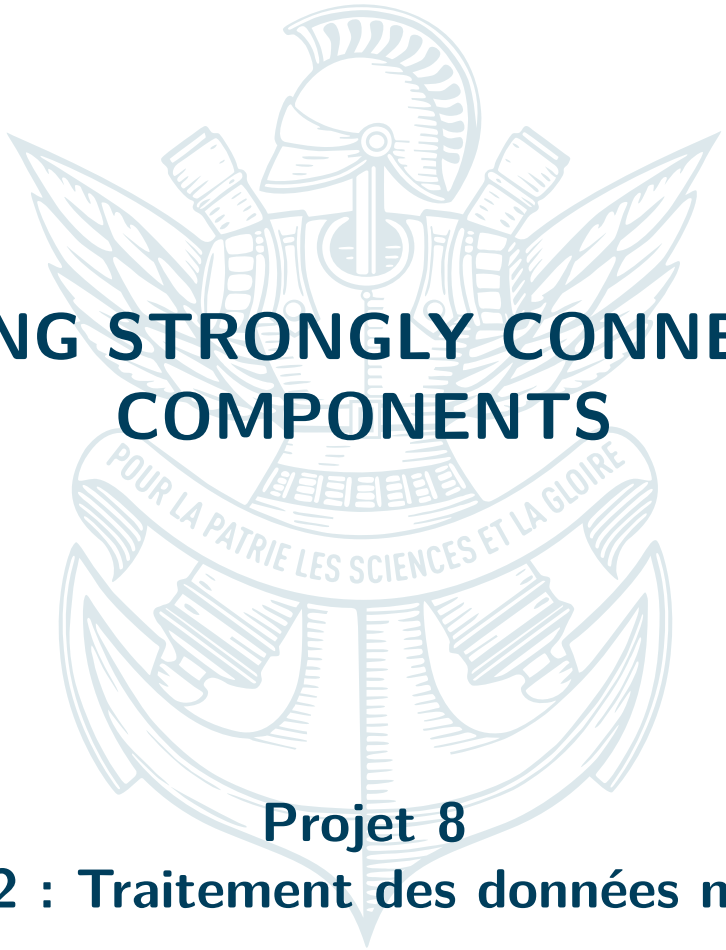

FINDING STRONGLY CONNECTED COMPONENTS

Projet 8
INF 442 : Traitement des données massives

2 juin 2017

LAPINHA DALLA STELLA Luis
MADRID CANALES Ignacio

TABLE DES MATIÈRES

1 PRÉLIMINAIRES : STRUCTURES DES DONNÉES CHOISIES POUR LE TRAITEMENT DE GRAPHES

En ce qui concerne les structures des donnés choisies, on a défini les graphes à travers listes d'adjacence plutôt qu'à travers matrices d'adjacence. Alors, on a une occupation en mémoire de $O(V + E)$, ainsi qu'une complexité de $O(V + E)$ pour les parcours en profondeur (utilisés pour l'algorithme séquentiel de recherche de SCC, ainsi que pour trouver les ensembles **Succ** et **Pred** dans l'algorithme parallèle).

Pour les listes d'adjacence de chaque sommet on a utilisé des objets du type `list<int>` de la STL de C++ plutôt que `vector<int>`, car on fait plusieurs insertions et *pops* de la liste. De même, on n'a pas besoin de connaître la position exacte des voisins au sein de la liste pour les algorithmes proposés. Ainsi la liste d'adjacence du objet `Graph` résout finalement une table de hachage où chaque sommet est un clé et la liste de ses voisins sortants sa valeur. Pour ce faire on utilise objets du type `map<int, list<int> >` de la STL.

2 PREMIER PARTIE : ALGORITHME SÉQUENTIEL DE RECHERCHE DE SCC

Algorithme de Kosaraju

Pour l'identification séquentielle de SCC on a décidé d'utiliser l'algorithme de Kosaraju pour sa simplicité. En revanche, cet algorithme est moins performant que l'algorithme de Tarjan (optimal en temps) car il requiert deux parcours en profondeur (DFS) au lieu d'un. L'algorithme est décrit a continuation :

Data : $G(V, E)$, $SCC = \emptyset$
Result : void() (SCC mis à jour, les éléments de SCC étant les strongly connected components)
$S \longleftarrow \emptyset$;
Faire DFS du graph G en ajoutant chaque sommet visité dans S;
Transposer G;
while S *n'est pas vide* **do**
  $v \longleftarrow$ premier element de S (pop);
  $L \longleftarrow \emptyset$;
  Faire DFS du G (transposé) à partir du sommet v en ajoutant chaque sommet visité dans L;
  Ajouter L à SCC
end

Algorithme 1 : Méthode de Kosaraju pour la recherche de SCC

Comme on fait les parcours DFS en utilisant listes d'adjacence on obtient un complexité linéaire $O(V + E)$ pour cet algorithme.

On a identifié alors SCC de graphes issues de la vie réelle (de la collection SNAP). Il était possible d'observer un grand nombre de SCC singletons, ce qui suggère que c'est peut-être un bon idée d'éliminer les sommets qui n'ont ni sorties ni entrées. Ensuite, on a programmé des graphes aléatoires pour analyser la distribution de la quantité de SCC's dans le contexte de la formation des graphes de Erdös-Rényi.

Graphes de Erdös-Rényi

Pour tester l'identification de SCC on va d'abord générer des graphes aléatoires de Erdös-Rényi (ER), où chaque arête possible est construite avec un probabilité p de manière indépendante.

On fait plusieurs essaies avec probabilités de l'ordre de $1/n$, plus précisément des probabilités c/n avec différents valeurs de c. On note d'ailleurs que c correspond à l'espérance du degré de sommets du graphe.

Probabilité	Nombre de SCC identifiés	Probabilité	Nombre de SCC identifiés
0.0025	1.8	0.0925	9.48
0.0125	7.5	0.1025	7.49
0.0225	11.84	0.1125	6.61
0.0325	13.96	0.1225	5.53
0.0425	15.24	0.1325	4.54
0.0525	15.63	0.1425	3.6
0.0625	15.15	0.1525	3.2
0.0725	14.00	0.2025	1.43
0.0825	11.95	0.4025*	1.00

Table 1 – Nombre moyen de SCC identifiés dans graphes ER de 20 sommets avec différentes probabilités de liaison. Pour chaque probabilité on a fait 100 simulations. *À partir d'une probabilité de 0.4025 on a obtenu tout le temps 1 SCC en moyenne (le graphe complète était fortement connecté)

On observe qu'à partir d'une probabilité de 0.4025 le graphe commence à être totalement fortement connecté. De même le nombre maximal de SCC pour le graphe ER de 20 sommets semble être autour de 15, et il est atteint quand la probabilité d'adjacence est autour de 0.05, c'est-à-dire avec un degré moyen attendu de 1 pour chaque sommet ($1/20 = 0.05$). En effet, avec graphes avec d'autres quantités de sommets, on atteint la quantité maximal de SCC quand le degré moyen est de 1 (83 SCC pour un graphe de 100 sommets quand $p = 0.105$, 24 SCC pour 30 sommets quand $p = 0.03$, etc.). Ces résultats ne sont pas détailles ici car ils seraient redondants à côté des donnés présentés dans le tableau 1.

3 DEUXIÈME PARTIE : ALGORITHME PARALLÈLE DE RECHERCHE DE SCC

Graphes ER en parallèle

On a écrit un programme pour générer graphes ER en parallèle. Nonobstant l'utilisation préférentielle de listes d'adjacence, on a décidé d'utiliser matrices d'adjacence pour la formation de digraphes ER en parallèle. On les décompose par lignes entre les processus (cf. TD7). Ainsi, chaque processus génère lignes indépendantes de la matrice et finalement on somme les matrices issues de chaque processus pour obtenir la matrice d'adjacence globale. Comme pour la TD7 on a préféré des appels à `MPI_Send` et `MPI_Recv`, en évitant `MPI_Reduce` avec `MPI_SUM`. En effet, l'implémentation de la communication entre les processus avec des matrices (somme de `int**`) a résout bien plus simple qu'un éventuel partage de listes d'adjacence (union de `list<int>` au sein d'un `map<int, list<int> >`), ce qui requiert connaître les sommets qui ont été assignés à chaque processus. En revanche, chaque processus doit générer une matrice complète de $V \times V$ remplie de zéros, pour après mettre aléatoirement les adjacences dans les lignes assignées, ce qui est assez inefficace en termes de mémoire ($O(V^2)$).

Recherche de SCC en parallèle