UNIVERSITÉ PARIS 8 DES CRÉATIONS

**UE : Algorithme Avancée**

Département Informatique – Année 2025

# Rapport - Graphes

Composantes connexes 1 sur de grands graphes (plus de 10000 noeuds)

**Auteur :**
BATHILY Mariame

**Enseignant :**
Mme Benjamin Dupont

Décembre 2025

# Rapport Technique : Algorithmique des Graphes à Grande Échelle

## 1. Présentation du Projet

L'objectif de ce projet est d'étudier l'efficacité des structures de données pour le stockage et le parcours de graphes pondérés de grande taille. Pour faire tout cela on a reprit le TP3 sur les Graphes en l'utilisant comme base. Nous nous concentrons sur deux algorithmes fondamentaux :

— La recherche de **composantes connexes (CC)** par parcours en largeur (BFS).

— Le calcul des **plus courts chemins** via l'algorithme de Dijkstra.

Le défi principal réside dans la gestion du passage à l'échelle pour des graphes dépassant les **10 000 nœuds**. À cette échelle, l'efficacité de la structure de données est le facteur déterminant de la performance globale.

## 2. Analyse des Structures de Données

Nous avons implémenté trois structures pour répondre aux différentes phases du projet :

— **Vecteur de Triplets** (`Tri`) : Utilisé dans `triplets.c` comme structure intermédiaire pour la génération aléatoire du graphe.

— **Matrice d'adjacence** (`Mat`) : Implémentée dans `matriceadj.c`. Bien que simple, son occupation mémoire est quadratique ($O(n^2)$), ce qui représente environ 381 Mo pour 10 000 nœuds.

— **Tableau de Brins** (`Brin` / `CSR`) : Implémenté dans `brin.c`. Cette structure compacte utilise trois vecteurs (`S`, `B`, `W`) pour ne stocker que les arêtes réelles, réduisant la complexité spatiale à $O(n + m)$.

## 3. Développement et Implémentation

### A. Gestion du Tableau de Brins (`brin.c`)

La fonction `convertir_triplets_vers_brin` transforme la liste de triplets en format CSR (Compressed Sparse Row). Cette étape est cruciale car elle organise les successeurs de manière contiguë en mémoire, optimisant ainsi les accès lors des parcours.

### B. Composantes Connexes (`CC_mat` et `CC_brin`)

Le calcul des composantes connexes repose sur un parcours BFS utilisant une file (`fifo.c`).

— Dans la version `Mat`, chaque sommet impose un scan complet de sa ligne ($n$ colonnes).

— Dans la version `Brin`, l'algorithme accède directement aux voisins via le tableau d'index `S`, évitant ainsi des millions de tests inutiles sur des cases vides.

### C. Plus court chemin avec Tas Binaire (`heap.c`)

Pour la structure `Brin`, nous avons couplé Dijkstra à un **Tas Binaire (Min-Heap)**. Cela permet d'extraire le sommet à distance minimale en $O(\log n)$ au lieu de $O(n)$, abaissant la complexité totale à $O(m \log n)$.

## 4. Procédure de Test et Benchmark

Le fichier `main.c` (Section 4) exécute les tests sur des graphes de 10 000, 15000 et 20 000 nœuds avec une densité d'arêtes de 1%. Nous mesurons :

1. L'occupation mémoire théorique calculée en Mo.
2. Le temps CPU réel via la fonction `clock()`.

## 5. Analyse des Résultats

Les tests réalisés confirment une supériorité écrasante de la structure compacte :

| **Métrique (n=10 000)** | **Matrice (Mat)** | **Brin (CSR)** | **Gain constaté** |
|---|---|---|---|
| Mémoire vive | 381.47 Mo | 7.67 Mo | $\approx$ 50x plus léger |
| Temps CC | 0.1699 s | 0.0026 s | x66.5 plus rapide |
| Temps Dijkstra | 0.6617 s | 0.0100 s | x66.0 plus rapide |

TABLE 1 – Synthèse des performances observées

**Analyse :** L'écart de performance est dû à la densité du graphe (1%). La structure Brin ne traite que les 1% d'arêtes existantes, tandis que la Matrice traite 100% des cases, dont 99% sont des zéros. À $n = 20\,000$, la Matrice nécessite plus de 1.5 Go de RAM, illustrant la limite physique de cette structure pour le Big Data.

**Conclusion :** Ce projet démontre l'importance capitale du choix de la structure de données. Pour les réseaux de grande taille, la structure `Brin` couplée à un `Tas` binaire est la seule solution viable, offrant des gains de performance et de mémoire supérieurs à un facteur 50.