Étude comparative d'algorithmes d'énumération des cliques maximales d'un graphe simple

Virgil Surin

Université de Mons

Directeur: *Hadrien Mélot*

August 29, 2024

Table des matières

Introduction

Motivation

L'énumération des cliques maximales dans un graphe simple est un problème couteux en temps avec de nombreuses applications concrètes (biologie, sociologie, informatique,...).

Introduction

Motivation

L'énumération des cliques maximales dans un graphe simple est un problème couteux en temps avec de nombreuses applications concrètes (biologie, sociologie, informatique,...).

Objectif

L'objectif de ce projet était d'étudier plusieurs algorithmes d'énumération de cliques maximales dans un graphe simple non orienté et de comparer leurs performances.

- Basé sur un article de Conte et Tomita[1]
- Implémentation en *Python* de 4 algorithmes
- Comparaison des performances des algorithmes
- Implémentation en *Rust* et comparaison avec *Python*

Les graphes simples

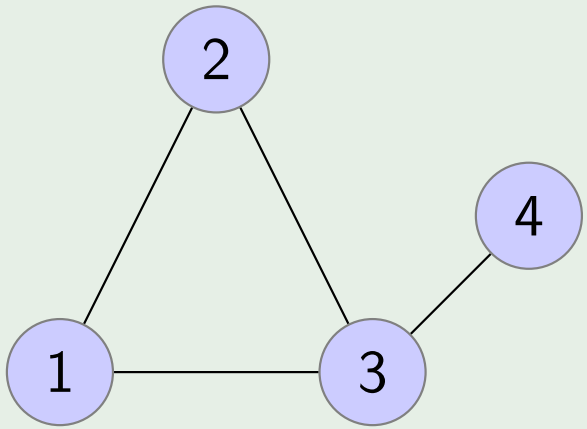
Definition

Un graphe simple non orienté est défini par un ensemble de nœuds V et un ensemble d'arêtes E. Chaque arête relie une paire de nœuds distincts sans direction associée, ce qui signifie que l'arête (v, w) est identique à l'arête (w, v).

Example

Graphe avec 4 noeuds :

$V = \{1, 2, 3, 4\}$
$E = \{(1, 2), (1, 3), (2, 3), (3, 4)\}$



Cliques maximales

Definition

Une clique dans un graphe est un sous-ensemble de nœuds tel que chaque paire de nœuds est connectée par une arête.

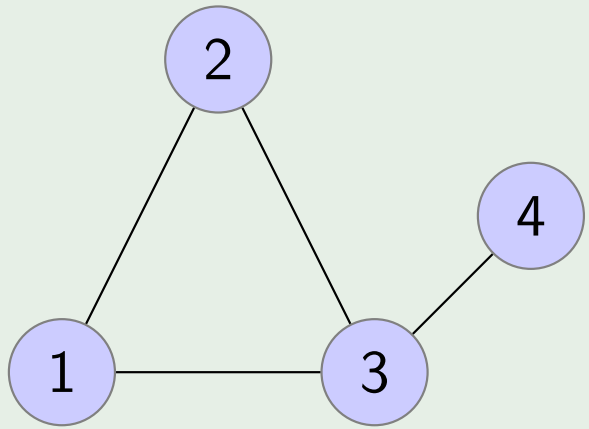
Definition

Une clique est dite maximale si aucun nœud supplémentaire ne peut être ajouté sans perdre la propriété de clique. Une clique maximale est donc un sous-graphe complet qui ne peut être étendu.

Example

Le graphe suivant possède les cliques maximales suivantes :

- $\{1, 2, 3\}$
- $\{3, 4\}$



Structure générale

Algorithm 1 Squelette de base

Input: un graphe $G = (V, E)$
Output: toutes les cliques maximales de G

```
1:  procedure ALG(SUBG, CAND)
2:      if SUBG = ∅ then                          ▷ Q est une clique maximale
3:          print (Q)
4:      else
5:          u ← un noeud pivot de SUBG
6:          while il reste des noeuds candidats do
7:              p ← un noeud dans CAND \ N(u)
8:              Q ∪ p                              ▷ on ajoute p à Q
9:              // Mise à jour des paramètres
10:             SUBG_p ← SUBG ∩ N(p)
11:             CAND_p ← CAND ∩ N(p)
12:             ALG(SUBG_p, CAND_p)
13:             CAND ← CAND \ p
14:             Q \ p                              ▷ on retire p de Q
15:         end while
16:     end if
17: end procedure
18: ALG(V, V)
```

Algorithme de Bron-Kerbosch

L'algorithme de Bron-Kerbosch est une méthode récursive pour énumérer toutes les cliques maximales dans un graphe. Il existe plusieurs variantes de cet algorithme :

- **Sans pivot :** Cette version explore toutes les combinaisons possibles de nœuds sans aucune optimisation particulière. Elle itère sur tous les candidats possibles pour construire les cliques maximales.
- **Avec pivot aléatoire :** Un nœud pivot est choisi aléatoirement parmi les nœuds disponibles.
- **Avec pivot de degré maximum :** Ici, le pivot est choisi comme étant le nœud ayant le plus grand nombre de voisins, ce qui permet de réduire davantage les appels récursifs en élaguant la recherche.

La complexité temporelle de ces variantes dans le pire des cas est $O(3^{n/3})$, correspondant au nombre maximum de cliques maximales dans un graphe.

Algorithme CLIQUES

L'algorithme CLIQUES est une optimisation de Bron-Kerbosch qui sélectionne un pivot de manière à minimiser le nombre de récursions nécessaires. Ce pivot est choisi pour maximiser l'intersection entre les candidats et les voisins du pivot, c'est-à-dire $CAND \cap N(u)$.
CLIQUES est la variante de Bron-Kerbosch dont on attend les meilleures performances.

Test sets

Premier ensemble de test

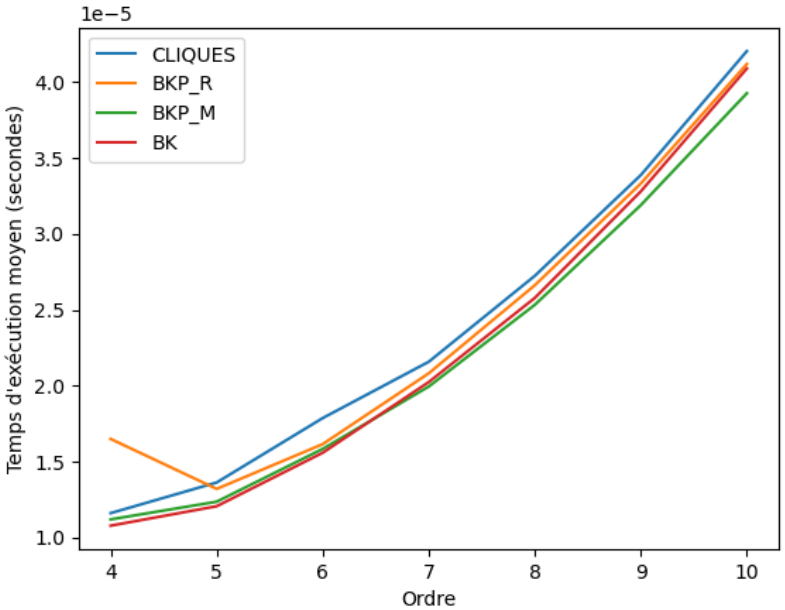
Ensemble des graphes d'ordre 4 à 10 inclus. (Soit 12 278 357 graphes)

Deuxième ensemble de test

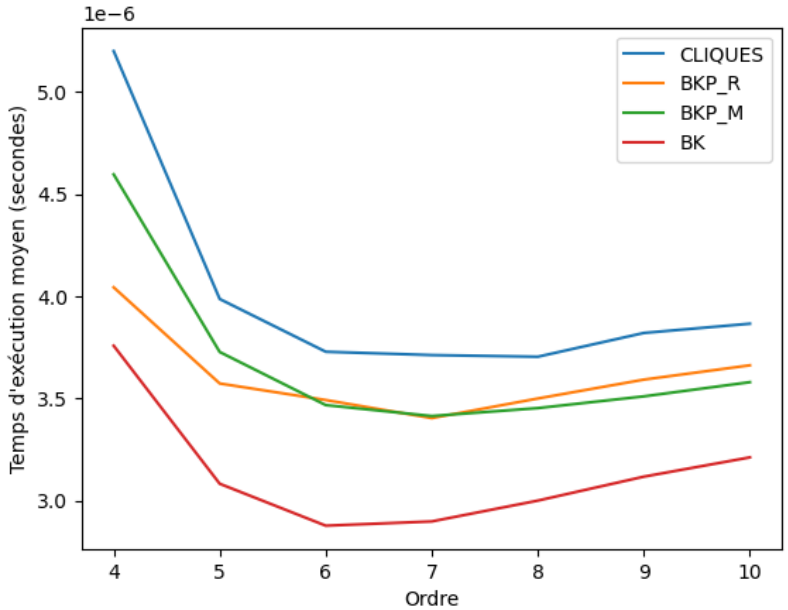
3 types de graphes différents, d'ordre 3 à 45 sélectionnés pour leurs propriétés spécifiques:

- Graphe vide (autant de cliques maximales que de noeuds)
- Graphe complet (une seule clique maximale contenant tous les noeuds)
- Graphe de Moon-Moser (contient le nombre maximum de cliques maximales possible, soit $3^{n/3}$, où n est le nombre de noeuds)

Résultats en Python - Premier test set

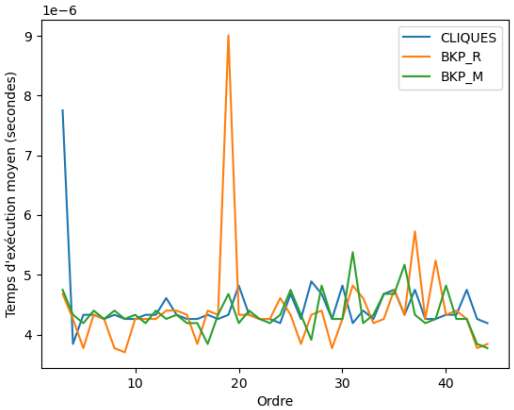


Temps total moyen pour énumérer toutes les cliques maximales

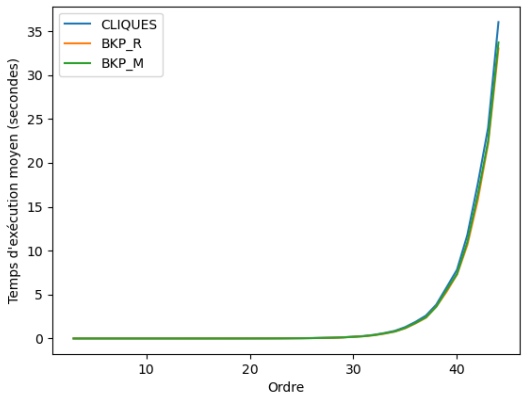


Délai moyen pour énumérer une clique maximale

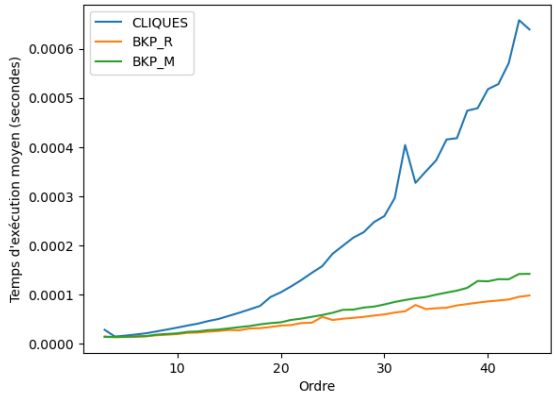
Résultats en Python - Second test set



Graphes vides

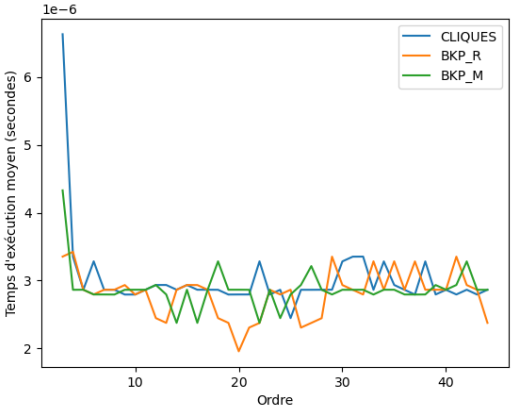


Moon-Moser

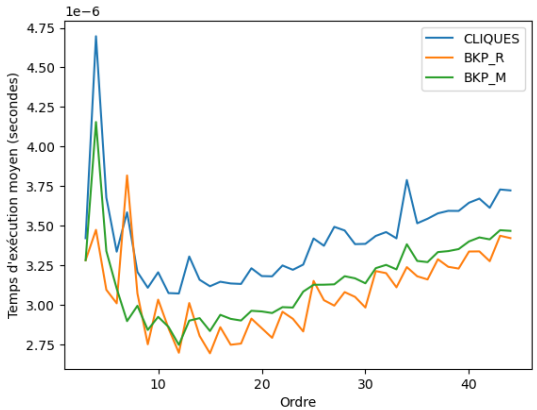


Graphes complets

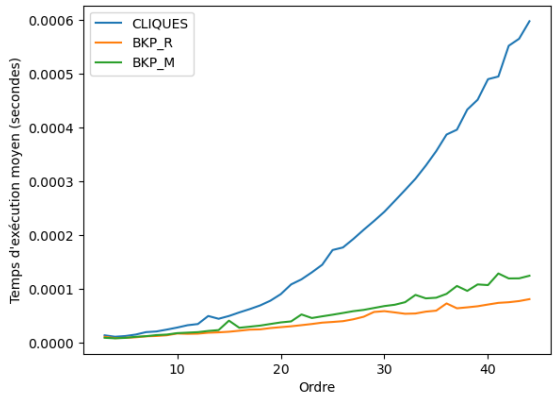
Temps totaux moyens



Graphes vides



Moon-Moser



Graphes complets

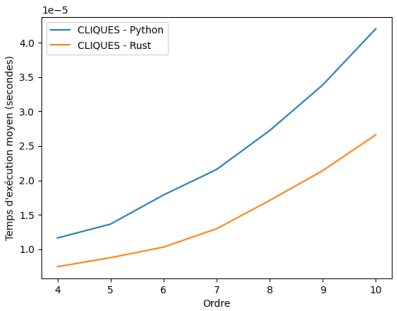
Délais moyens

Résultats en Python - Conclusion

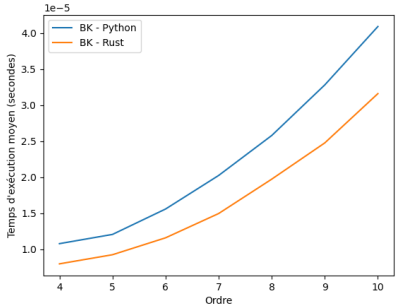
Conclusion

- Courbes exponentielles peut importe l'aglgorithme (se voit surtout sur les graphes de Moon-Moser)
- Le délai est bien linéaire
- Certains algorithmes sont meilleurs selon le type de graphe (exemple : CLIQUES ne performe pas bien sur les graphes complets)

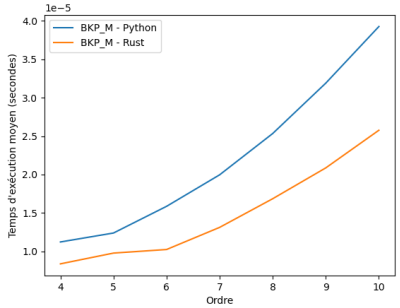
Comparaison Python vs Rust - Premier test set



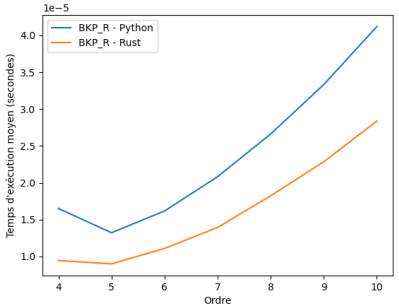
CLIQUES



Bron-Kerbosch



BKP_M

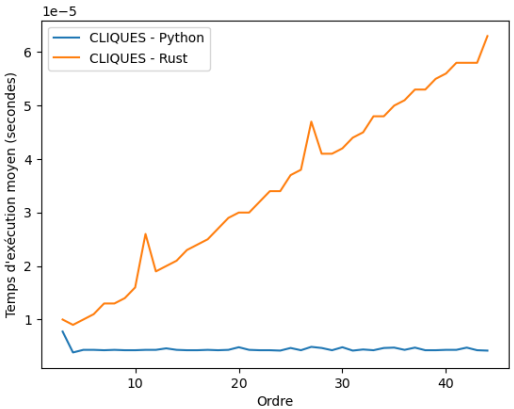


BKP_R

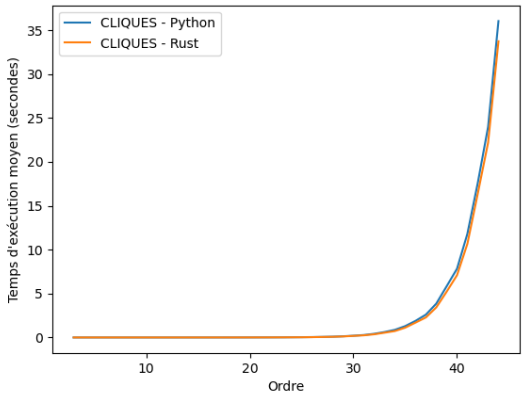
Temps totaux moyens

Comparaison Python vs Rust - Second test set

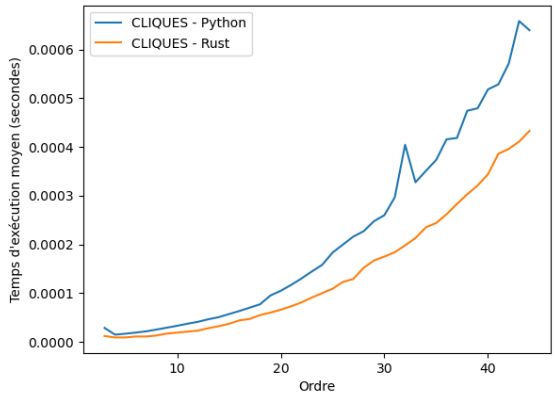
CLIQUES



Graphes vides



Moon-Moser

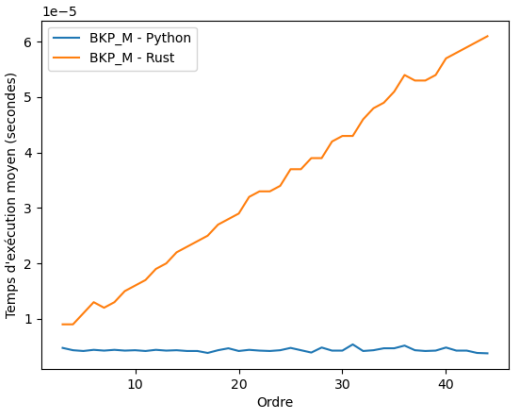


Graphes complets

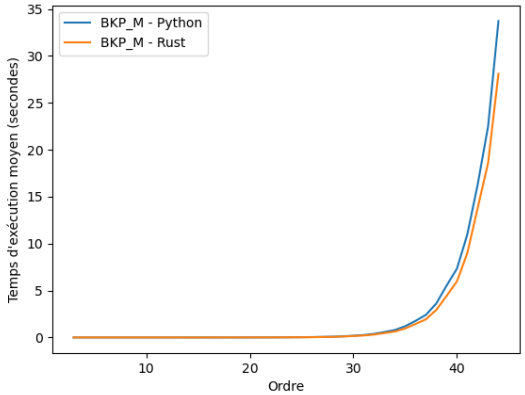
Temps totaux moyens

Comparaison Python vs Rust - Second test set

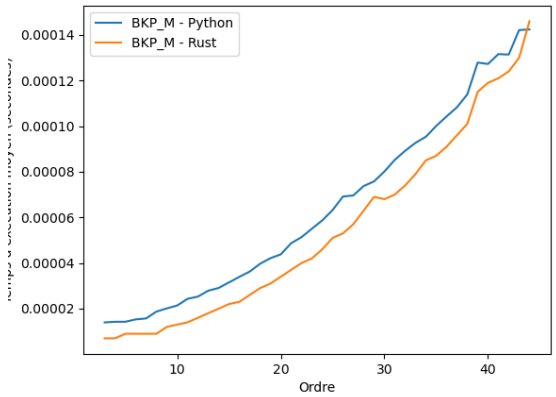
Bron-Kerbosch avec pivot de degré maximum



Graphes vides



Moon-Moser

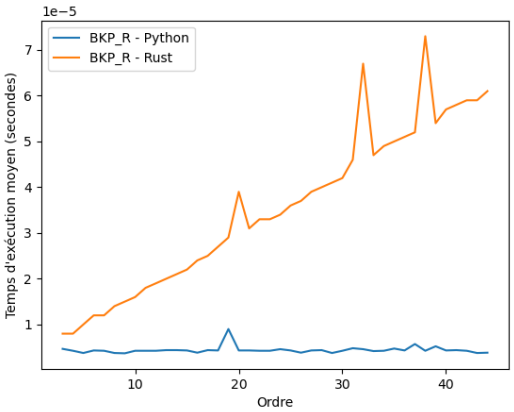


Graphes complets

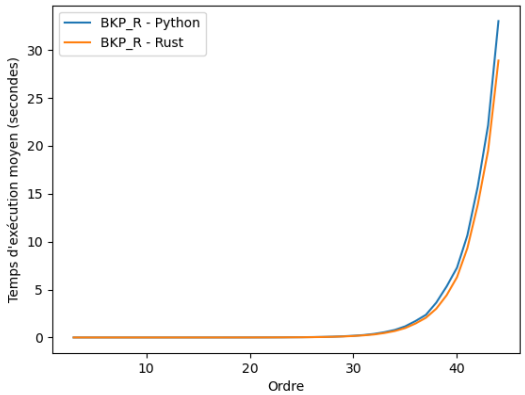
Temps totaux moyens

Comparaison Python vs Rust - Second test set

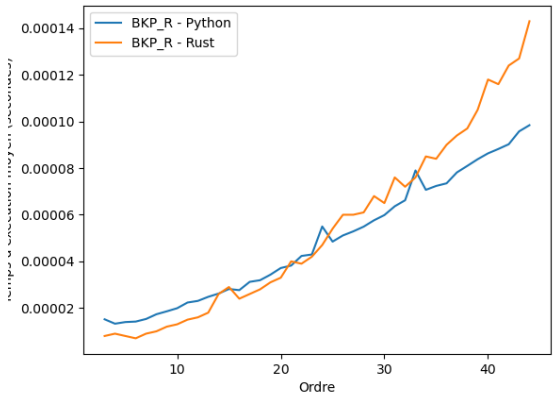
Bron-Kerbosch avec pivot aléatoire



Graphes vides



Moon-Moser



Graphes complets

Temps totaux moyens

Comparaison Python vs Rust

```rust
        let subg_vec: Vec<&u32> = subg.iter().collect();
        let u = subg_vec.choose(&mut rng).unwrap();
        let u_neighbors: HashSet<_> = g.adj[&u].iter().copied().collect();
        for p in cand.difference(&u_neighbors).copied().collect::<Vec<u32>>() {
            let p_neighbors: HashSet<_> = g.adj[&p].iter().copied().collect();

```

Figure: Extrait de code Rust

```python
        u = random.choice(list(SUBG))
        for p in CAND - G.adj[u]:
            p_neighbors = G.adj[p]

```

Figure: Extrait de code Python

Comparaison Python vs Rust - Conclusion

Conclusion

- Rust est bien meilleur que Python de manière générale
- Résultat clairement inférieur ou mitigé sur des graphes spéciaux, probablement à cause de l'implémentation (*HashSet*)

Conclusion

Conclusion

- Les algorithmes de Bron-Kerbosch avec pivot et CLIQUES sont bien efficaces
- Les algorithmes sont sensibles au type du graphe
- Le choix du pivot est important (cout en performance)
- Moyennant une bonne implémentation, on peut avoir des gains significatifs en Rust par rapport à Python

Questions

Merci pour votre attention. Des questions ?

Bibliographie I

[1] Alessio Conte and Etsuji Tomita. "On the overall and delay complexity of the CLIQUES and Bron-Kerbosch algorithms". In: *Theoretical Computer Science* 899 (2022), pp. 1–24. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2021.11.005. URL: https://www.sciencedirect.com/science/article/pii/S0304397521006538.