1. Création et interrogation de graphes de de Bruijn colorés

Comparaison d'une approche naïve et d'une approche exploitant les unitigs

1. Introduction

Les graphes de de Bruijn colorés (cDBG) sont des structures de données largement utilisées pour l'analyse alignment-free de collections de génomes. Ils permettent de représenter l'ensemble des k-mers issus de plusieurs génomes tout en conservant, pour chacun d'eux, l'information d'appartenance aux différents génomes, appelée couleur. Cette structure rend possible le calcul de mesures de similarité entre une séquence requête et chaque génome de la collection, par exemple via le ratio de k-mers partagés.

Dans ce projet, nous avons implémenté et comparé deux versions d'un outil de création et de requête sur des graphes de de Bruijn colorés. Une première version, dite naïve, associe directement à chaque k-mer la liste des génomes dans lesquels il apparaît. Une seconde version, dite avancée, exploite une propriété structurelle du graphe : tous les k-mers appartenant à un même unitig partagent exactement le même ensemble de couleurs. Cette propriété permet de factoriser l'information de couleur et de réduire la taille globale de la structure.

L'objectif de ce rapport est de comparer ces deux approches du point de vue des performances temporelles et de l'occupation mémoire sur disque, pour différentes valeurs du paramètre k (avec $k \geq 17$), et d'en discuter les avantages et limites.

2. Méthodes

2.1 Version naïve du cDBG

Dans la version naïve, le graphe de de Bruijn coloré est représenté par un dictionnaire Python associant chaque k-mer observé à la liste des identifiants des génomes dans lesquels il apparaît. La phase de construction consiste à parcourir l'ensemble des génomes d'entrée, à extraire tous les k-mers possibles pour une valeur donnée de k, puis à mettre à jour cette structure. Dans la suite, chaque couleur correspond à un génome de la collection et est notée Gi.

Une fois le graphe construit, celui-ci est sérialisé sur disque à l'aide du module pickle. Lors de la phase de requête, le cDBG est désérialisé, puis chaque k-mer issu de la séquence requête est recherché afin de calculer, pour chaque génome, le ratio de k-mers partagés.

Cette approche est simple à implémenter et à comprendre, mais elle présente une redondance importante : l'information de couleur est stockée séparément pour chaque k-mer, y compris lorsque plusieurs k-mers consécutifs partagent exactement les mêmes couleurs.

2.2 Version avancée exploitant les unitigs

La version avancée repose sur l'observation que, dans un graphe de de Bruijn coloré, tous les k-mers appartenant à un même unitig possèdent le même ensemble de couleurs. Un unitig correspond à un chemin maximal du graphe ne présentant ni bifurcation entrante ni sortante.

Plutôt que d'associer une liste de couleurs à chaque k-mer individuellement, nous associons une unique information de couleur à chaque unitig. Les k-mers sont ensuite rattachés à leur unitig correspondant. Cette factorisation permet d'éviter la répétition de la même information de couleur pour des k-mers consécutifs.

Le principe de cette optimisation est illustré en Figure 1, qui compare la représentation naïve, où chaque k-mer stocke explicitement ses couleurs, et la représentation avancée, où les couleurs sont stockées une seule fois au niveau de l'unitig. Cette approche permet de réduire la taille mémoire du graphe, au prix d'une complexité algorithmique légèrement accrue lors de la construction.

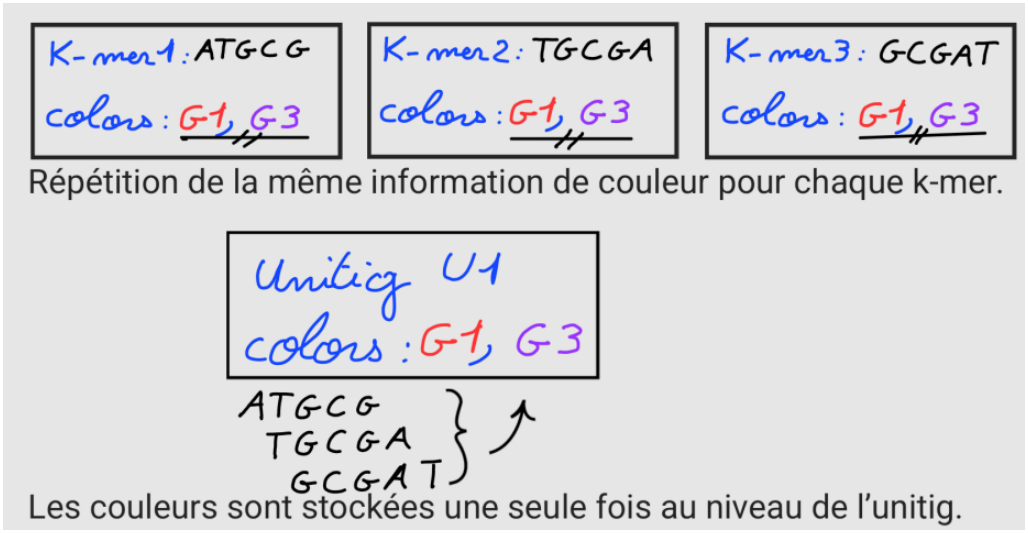

Figure 1 :

Comparaison entre la représentation naïve d'un graphe de de Bruijn coloré, où chaque k-mer stocke explicitement sa liste de couleurs (génomes Gi), et la représentation avancée exploitant les unitigs, où l'information de couleur est factorisée au niveau des unitigs.

3. Protocole expérimental

Les graphes de de Bruijn colorés ont été construits à partir de jeux de données génomiques fournis dans le cadre du projet. Trois valeurs du paramètre k ont été considérées : 17, 24 et 31.

La construction de l'index a été réalisée via la commande :

./dbg_indexer.py build -k {k} -i {liste_genomes} -o {cdbg}

La phase de requête s'appuie sur la commande :

./dbg_indexer.py query -cdbg {cdbg} -q {requêtes.fasta} -o {résultats.txt}

Les temps mesurés correspondent aux valeurs affichées par le programme lors de son exécution. En particulier, nous avons mesuré :

- le temps de construction du graphe avant sérialisation (OUT TIME_BUILD) ;
- le temps de sérialisation sur disque (OUT TIME_SERIALISATION).

La Figure 2 présente le pipeline général des phases de construction et de requête, ainsi que les points précis où les mesures de temps sont effectuées.

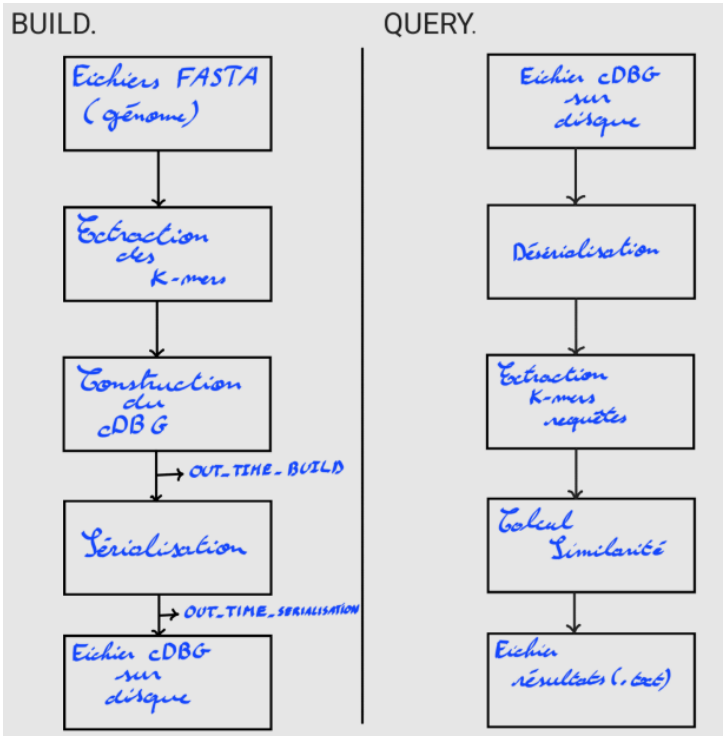

Figure 2 :
Pipeline général des phases de construction (*build*) et de requête (*query*) du graphe de de Bruijn coloré. Les emplacements des mesures de temps utilisées dans l'analyse des performances (construction, sérialisation et désérialisation) sont indiqués..

4. Résultats

4.1 Temps de construction et de sérialisation

La Figure 3 présente les temps de construction du graphe de de Bruijn coloré (OUT TIME_BUILD) et de sérialisation sur disque (OUT TIME_SERIALISATION) pour les versions naïve et avancée, en fonction de la valeur du paramètre k. En abscisse est représentée la valeur de k (17, 24 et 31), et en ordonnée le temps mesuré en secondes.

Pour les deux versions, les temps augmentent lorsque k augmente, ce qui s'explique par l'augmentation du nombre de k-mers distincts à traiter. La version avancée présente généralement un temps de construction légèrement supérieur à celui de la version naïve. Ce surcoût est dû à la gestion des unitigs et à la factorisation de l'information de couleur. En revanche, la sérialisation bénéficie d'une structure plus compacte, ce qui peut réduire le temps d'écriture sur disque dans certains cas.

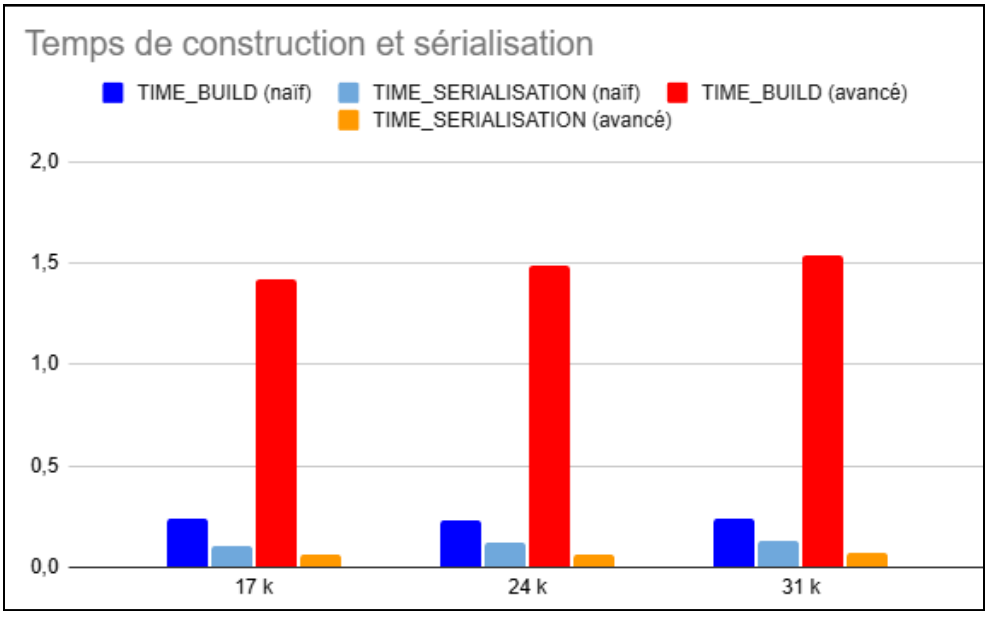

Figure 3 : Temps de construction du graphe de de Bruijn coloré (OUT TIME_BUILD) et de sérialisation sur disque (OUT TIME_SERIALISATION) pour les versions naïve et avancée.
En abscisse est représentée la valeur du paramètre *k* (17, 24 et 31), et en ordonnée le temps mesuré en secondes.

4.2 Temps de désérialisation et de requête

La Figure 4 présente les temps de désérialisation du graphe de de Bruijn coloré (OUT TIME_DESERIALISATION) pour les versions naïve et avancée. En abscisse est représentée la valeur du paramètre k (17, 24 et 31), et en ordonnée le temps de désérialisation mesuré en secondes.

Les temps de désérialisation de la version naïve sont compris entre 0,12 s et 0,13 s, tandis que ceux de la version avancée sont compris entre 0,055 s et 0,066 s. Cette différence s'explique par la taille plus compacte du cDBG dans la version avancée, qui réduit le volume de données à relire depuis le disque. L'écart tend légèrement à augmenter avec k, en cohérence avec l'augmentation de la taille des graphes sérialisés.

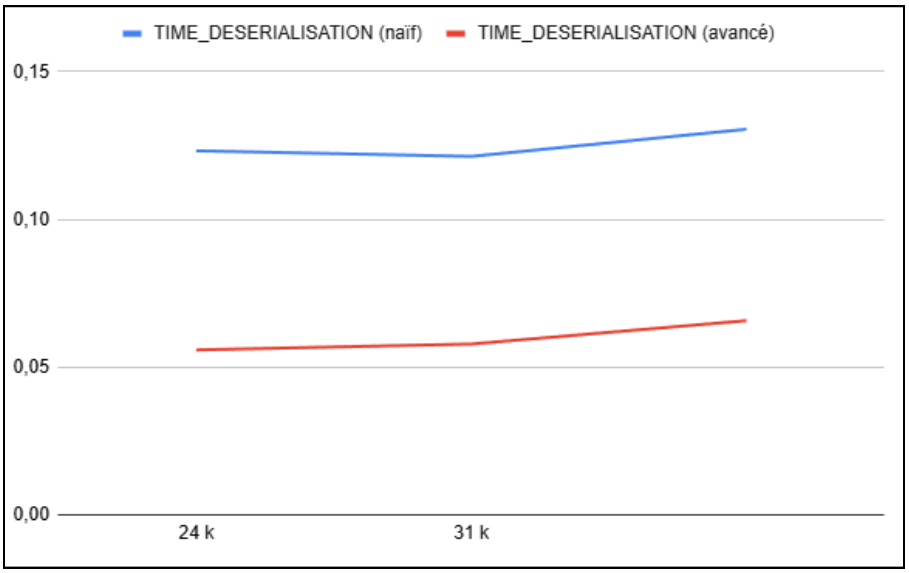

Figure 4 : Temps de désérialisation du graphe de de Bruijn coloré (OUT TIME_DESERIALISATION) pour les versions naïve et avancée.
En abscisse est représentée la valeur du paramètre *k* (17, 24 et 31), et en ordonnée le temps de désérialisation mesuré en secondes.

La Figure 5 présente les temps de requête proprement dits (OUT TIME_QUERY) pour les versions naïve et avancée. En abscisse est représentée la valeur du paramètre *k* (17, 24 et 31), et en ordonnée le temps de requête mesuré en secondes.

Pour la version naïve, les temps de requête sont très faibles et quasi constants, autour de 0,03 s, quelle que soit la valeur de *k*. En revanche, la version avancée présente des temps de requête nettement plus élevés, compris entre environ 13 s et 15 s, bien que ces temps diminuent légèrement lorsque *k* augmente.

Cette différence importante s'explique par la complexité accrue de la phase de requête dans la version avancée, notamment liée à l'indirection supplémentaire nécessaire pour accéder aux informations de couleur via les unitigs, ainsi qu'aux choix d'implémentation de la structure de données utilisée pour effectuer les correspondances entre k-mers et unitigs.

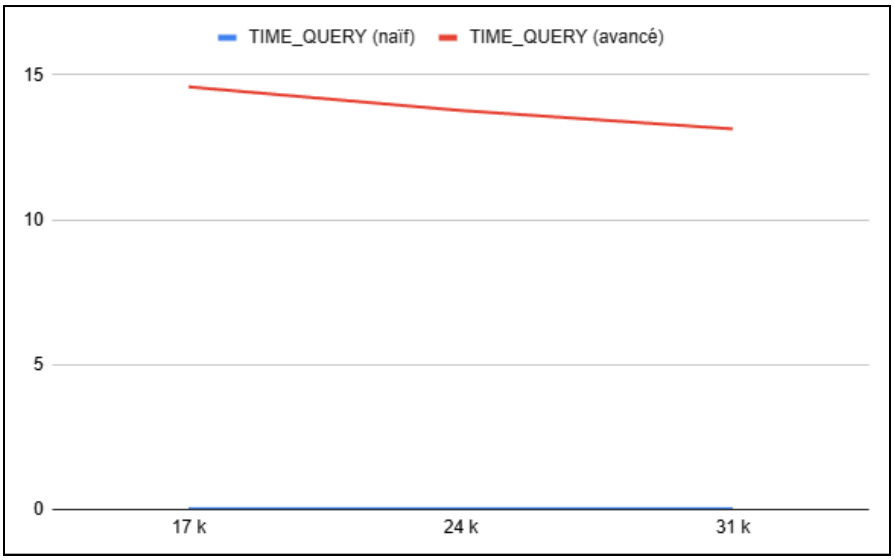

Figure 5 : Temps de requête (OUT TIME_QUERY) pour les versions naïve et avancée.
En abscisse est représentée la valeur du paramètre *k* (17, 24 et 31), et en ordonnée le temps de requête mesuré en secondes.

4.3 Taille des graphes sérialisés

La Figure 6 présente la taille des fichiers représentant les graphes de de Bruijn colorés sérialisés sur disque pour les versions naïve et avancée.
En abscisse est représentée la valeur du paramètre *k* (17, 24 et 31), et en ordonnée la taille du fichier sérialisé, mesurée en mégaoctets.

Pour toutes les valeurs de *k* testées, la version avancée produit des fichiers plus petits que la version naïve. L'écart entre les deux versions tend à augmenter lorsque *k* augmente. Ces résultats confirment que la factorisation des couleurs au niveau des unitigs permet une réduction significative de la taille mémoire du graphe, sans compromettre la capacité de requête.

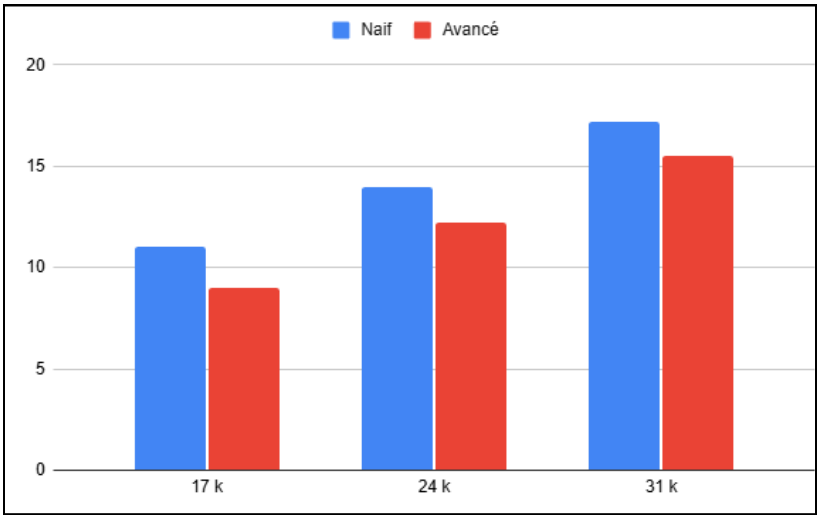

Figure 6 : Taille des fichiers représentant les graphes de de Bruijn colorés sérialisés sur disque pour les versions naïve et avancée.
En abscisse est représentée la valeur du paramètre *k* (17, 24 et 31), et en ordonnée la taille du fichier sérialisé, mesurée en mégaoctets.

5. Discussion et perspectives

La comparaison des deux versions met en évidence un compromis clair entre simplicité et efficacité. La version naïve est plus simple à implémenter et présente des temps de construction et de requête très faibles, mais elle souffre d'une consommation mémoire supérieure due à la redondance de l'information de couleur. À l'inverse, la

version avancée permet de réduire significativement la taille du graphe sérialisé, ce qui se traduit par des temps de désérialisation plus faibles.

Cependant, l'analyse des performances de la phase de requête met en évidence une limite importante de notre implémentation avancée. Bien que la compacité accrue du graphe soit bénéfique pour les accès disque, le temps de requête proprement dit est nettement plus élevé que dans la version naïve. Ce surcoût est principalement dû à la complexité accrue de l'accès aux informations de couleur via les unitigs, qui introduit plusieurs niveaux d'indirection et augmente le nombre d'opérations nécessaires par k-mer.

Ces résultats soulignent que le gain mémoire apporté par l'approche avancée ne se traduit pas automatiquement par un gain en temps lors de la phase de requête. Il est important de noter que ces performances dépendent fortement des choix d'implémentation, et qu'une version avancée plus optimisée pourrait réduire significativement le temps de requête observé.

Du point de vue de l'utilisation, une amélioration possible aurait été d'introduire un paramètre de ligne de commande permettant de sélectionner dynamiquement la version naïve ou avancée, plutôt que de séparer les deux implémentations dans des répertoires distincts. Sur le plan algorithmique, des optimisations supplémentaires telles que l'utilisation de minimizers ou l'encodage binaire des séquences pourraient permettre de limiter le nombre de comparaisons effectuées lors de la phase de requête et d'améliorer les performances globales.

6. Conclusion

Dans ce projet, nous avons implémenté et comparé deux approches pour la construction et l'interrogation de graphes de de Bruijn colorés. Les résultats montrent que l'exploitation des unitigs permet de réduire significativement la taille de la structure sérialisée et d'améliorer les temps de désérialisation, au prix d'un surcoût notable lors de la phase de requête dans notre implémentation actuelle. La version avancée constitue ainsi un compromis intéressant du point de vue mémoire, tandis que la version naïve reste plus efficace pour des requêtes rapides.