*Victor BERTHOD* *Yann LEFRANCOIS*


**UE ALG : Algorithmique des Séquences**

# Rapport utilisateur du Projet ReadOrganizer

*Claire Lemaitre et Pierre Peterlongo*
*2023-2024*

# Sommaire :

## Introduction

Ce rapport utilisateur a pour objectif de présenter le projet ReadOrganizer avec une approche pratique dans le but de préciser son cadre d'utilisation. Le projet vise à améliorer la compression des fichiers de lecture issus du séquençage haut débit en réorganisant ces lectures de manière à optimiser l'efficacité de la compression via le compresseur générique gzip.

Cet outil aidant à la compression de données issus de séquençages répond à la problématique de stockage de ces données. Les séquençages devenant toujours plus accessibles et la quantité de data en constante croissance nécessitent une attention particulière dans la compression dite "loss-less". Ces compressions réduisent ainsi l'espace de stockage nécessaire tout en facilitant le partage et la manipulation de ces fichiers massifs.

Le script proposé s'appuie sur l'utilisation d'un génome de référence pour réorganiser les fichiers de reads de manière à les rendre plus efficients dans la compression. Une compression sans cette étape de réorganisation permet de diviser la taille du fichier avec un facteur d'environ 3. Cet outil propose 2 approches de mapping appelées méthodes 1 et méthodes 2 permettant de s'adapter au mieux aux jeux de données. L'idée est d'aligner chacun des reads sur le génome en utilisant une méthode de mapping pour déterminer leurs positions. Des données de séquençage de mauvaise qualité issues d'un séquençage avec une faible précision nécessitera de réduire la taille des sous-séquences pour obtenir un mapping correct. Ce projet vise à garantir une efficacité optimale en termes de temps de calcul, d'utilisation de la mémoire vive et de taux de compression.

## Comment l'utiliser ?

L'outil est un ensemble de fichiers python développé en python 3.10.11 :

- BTW_package.py
- tools_karkkainen_sanders.py
- read.py
- ReadOrganizer.py

Ces scripts doivent se trouver dans le même répertoire que votre génome de référence et vos données de séquençage. L'outil se lance dans un terminal de la manière suivante :

```
python .\ReadOrganizer.py -i .\sequencing_data.fasta -r
.\reference_genome.fasta -k 5 -m 1 -o reorganized_reads_k5_m1.fasta
```

Voici les paramètres configurables :

- i : le fichier de reads issus du séquençage
- r : le génome à utiliser en référence
- k : la taille de la seed à utiliser pour le mapping
- m : la méthode de mapping à utiliser 1 ou 2
- o : le nom du fichier de sortie de vos reads réorganisés

## 1. Méthode 1

### a. Présentation du fonctionnement

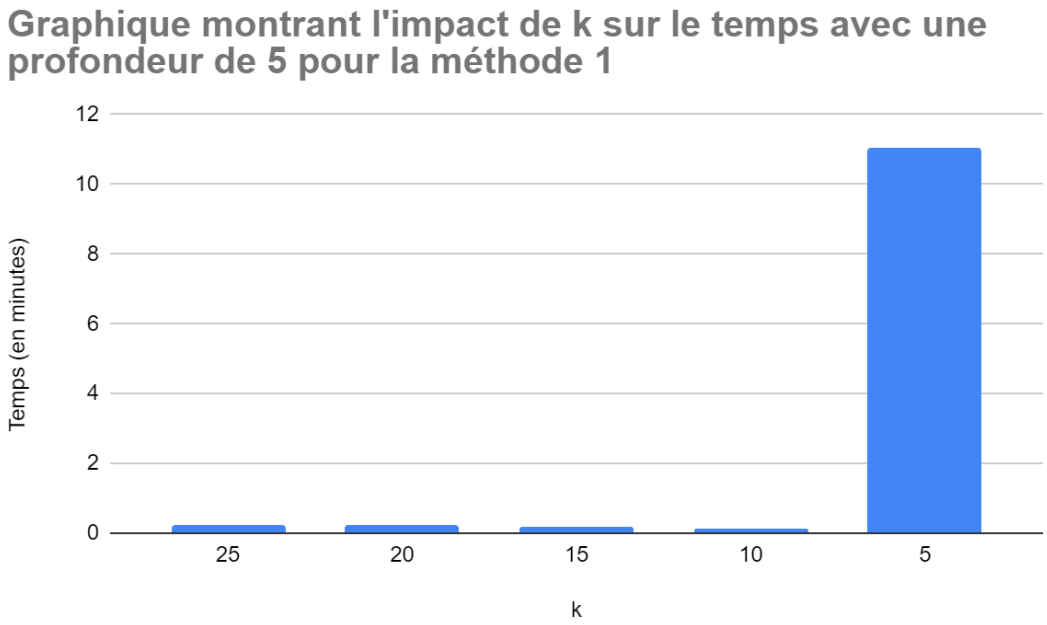
La méthode 1 est la méthode utilisée lors de l'exécution du script avec l'argument "-m 1". Lors de son exécution, elle va parcourir et lire le read en listant toutes les sous-séquences possibles de taille k. La taille de k est modifiable avec l'argument "-k". L'ensemble de ces sous-séquences sont ensuite alignés au génome de référence pour rechercher des matchs. La position de chaque match est conservée en mémoire pour pouvoir par la suite sélectionner la meilleure d'entre elle avec pour critère de sélection : la position dans le génome présentant le moins de substitution avec la séquence du read. Ce processus est répété pour tous les reads une première fois, puis pour les reads qui n'ont pas trouvé de positions sur le génome, on regarde si la séquence reverse complémentaire à une position dans le génome.

L'ensemble des reads est ensuite trié dans l'ordre croissant de leur positions dans le génome avec l'algorithme Tim, implémenter dans la fonction sort() de python. D'autres algorithmes de tri ont été essayés cependant, étant donné que la fonction sort() de python est codée en C, aucun algorithme de tri codé manuellement ne permettrait d'aller plus vite pour simplement ordonner des nombres entiers.

Et pour finir l'entièreté des reads sera écrit dans l'ordre croissant de leur position dans le génome dans un fichier, et les reads qui n'ont pas trouvé de positions dans le génome de référence sont tous écrits au début du fichier de sortie.

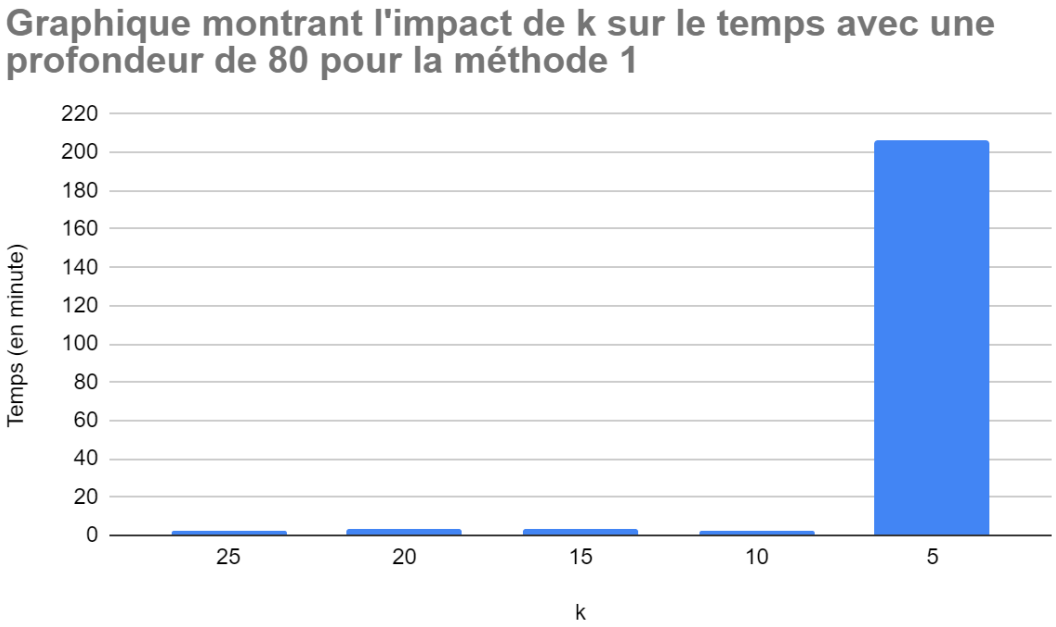
C'est une méthode exhaustive qui recherche toutes les sous-séquences possibles et toutes les occurrences de celles-ci dans le génome. Toutes les occurrences sont stockées, puis une sélection de la meilleure selon le nombre de substitutions dans l'alignement est effectuée.

### b. Effet de la taille de k sur le temps de calcul



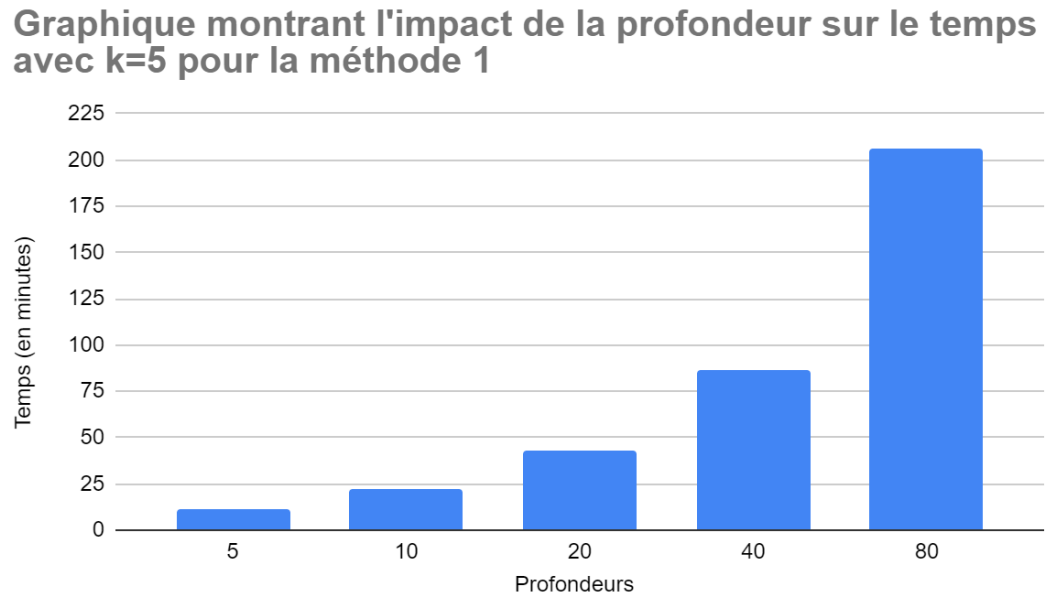

Le graphique ci-dessus montre une évolution brutale du temps de calcul lors de la variation de k, la taille des sous-séquences. Le jeu de données à réorganiser est "humch1_100Kb_reads_5x.fasta", c'est le fichier de reads séquencés avec une profondeur de

5. On observe que pour un k allant de 25 à 10, le script prend une dizaine de secondes environ. À noter qu'il y a une légère diminution, pour k=25 le script prend 14 secondes et pour k=10, il prend 9 secondes. Cela s'explique par le fait que la vitesse du pattern matching dépend de la taille du pattern k. Plus le pattern est grand, plus la recherche est longue.
Cependant l'écart se creuse quand le k diminue, k=5 prend 11 minutes. Le temps de calcul suit une croissance exponentielle très forte. Cela s'explique simplement par le nombre de sous-séquences qui augmente lui aussi en exponentielle le nombre de séquences à traiter pour trouver la meilleure position.




La méthode 1 réagit toujours de la même façon peu importe la profondeur à laquelle les k sont étudiés. On remarque toujours un temps très conséquent pour des k faibles.

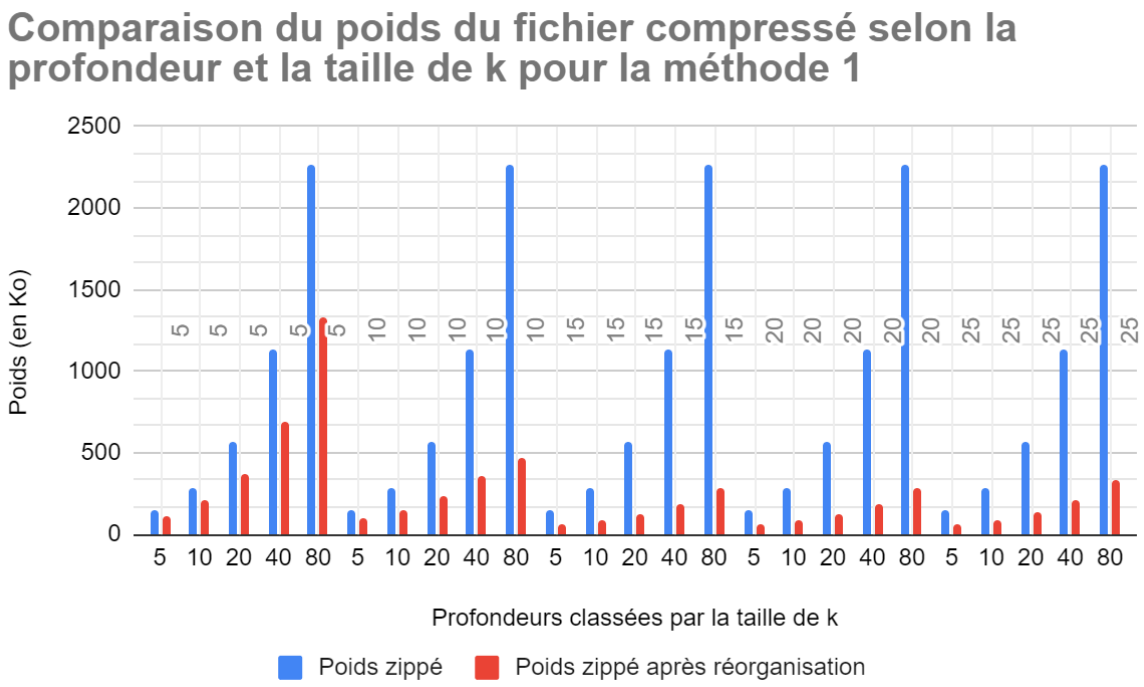
### c. Effet de la profondeur sur le temps de calcul




Le paramètre étudié ici est la profondeur de séquençage des fichiers de reads. On observe que le temps de calcul semble augmenter proportionnellement avec la profondeur.

On apprend ici que la méthode 1 peut prendre longtemps pour traiter des fichiers avec beaucoup de profondeur. Ici pour le fichier “humch1_100Kb_reads_80x.fasta”, le script prend 3h26 pour réorganiser. Les temps selon les profondeurs se répartissent toujours de cette manière peu importe le paramètre k rentré. Plus le séquençage est profond, plus la réorganisation prendra du temps.

### d. Taux de compression




Ce graphique montre le poids du fichier compressé après sa réorganisation par la méthode 1 en rouge et le poid zippé sans réorganisation en bleu pour servir de référence. Les poids sont classés par la taille de k pour toutes les profondeurs. On observe qu’avec des k faibles, ici k=5, que la compression a plus de mal. On constate pour k=5 un rapport de compression allant de 1.25 pour une profondeur de 5 à 1.7 pour une profondeur de 80. Tandis que pour des k plus grands comme k=20 sur une profondeur de 80, le rapport de compression est de 8. La meilleure compression est obtenue avec k =20.On observe donc que la compression avec un k faible est moins bonne. Cela s'explique par le fait qu’il va mapper des reads qui n’appartiennent pas au génome et les insérer au milieu d'autres séquences cassant la continuité des reads.

| **Taille de k** | **5** | **10** | **15** | **20** | **25** |
|---|---|---|---|---|---|
| **Poids compressé sans réorganisation (en Ko)** | 2269 | 2269 | 2269 | 2269 | 2269 |
| **Poids compressé avec réorganisation (en Ko)** | 1327 | 472 | 290 | 282 | 334 |
| **Rapport de compression** | 1,7 | 4,8 | 7,8 | 8 | 6,8 |

Tableau récapitulatif du rapport de compression pour une profondeur de 80 :

### e. Conseils d’utilisation

L’utilisation de la méthode 1 pour réorganiser les reads est donc pertinente pour des k élevés et moyens. La taille de k est le facteur le plus impactant sur le temps de réorganisation et sur le facteur compression. Il n’est pas recommandé d’utiliser cette méthode pour des k faibles. Un k = 20 est préconisé.

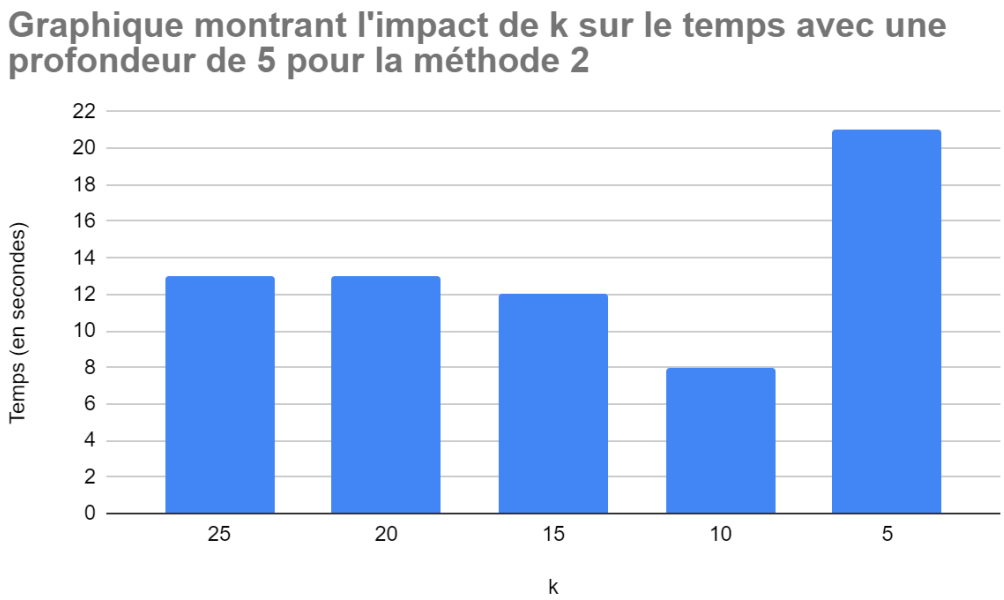
## 2. Méthode 2

### a. Présentation du fonctionnement

La méthode 2 est la méthode utilisée lors de l'exécution du script avec l’argument “-m 2”. Lors de son exécution, elle aussi va parcourir lire le read sur toutes les sous-séquences possibles de taille k. À l’instar de la méthode 1, elle va récupérer les positions pour les occurrences des matchs potentielles mais cette fois sous la forme d'un dictionnaire. Le critère de sélection du meilleur match se fait sur la fréquence des positions d'occurrences. Ce processus est répété pour tous les reads qu’ils soient en sens direct ou en reverse complement. L’ensemble sélectionné est ensuite conservé après avoir été trié dans un fichier prêt à être compressé.

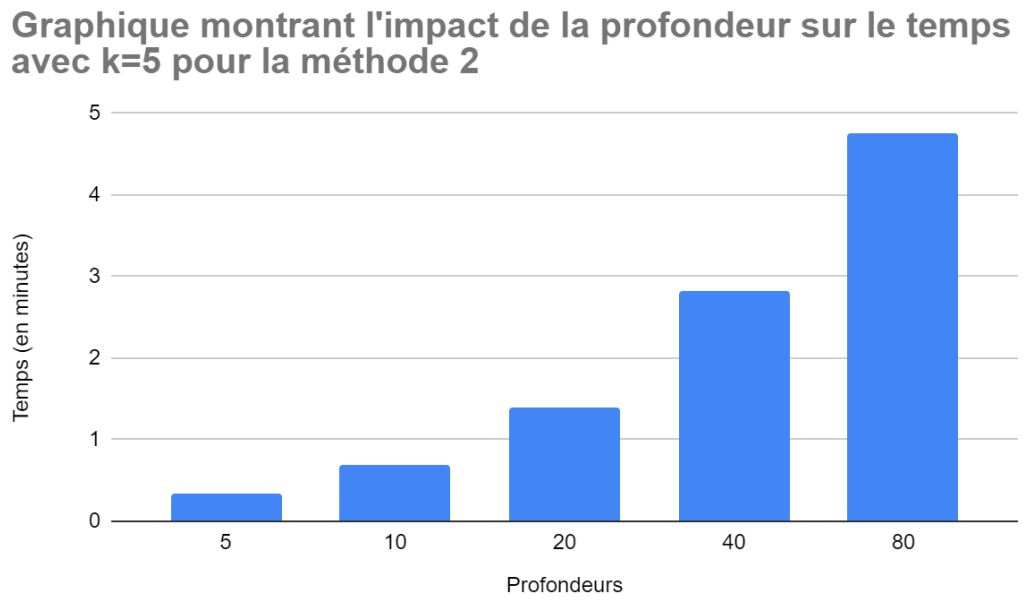
C’est une méthode basée sur les fréquences, cela permet de réduire le nombre de positions potentielles et donc avoir théoriquement de meilleurs performances pour des k faibles.

### b. Effet de la taille de k sur le temps de calcul



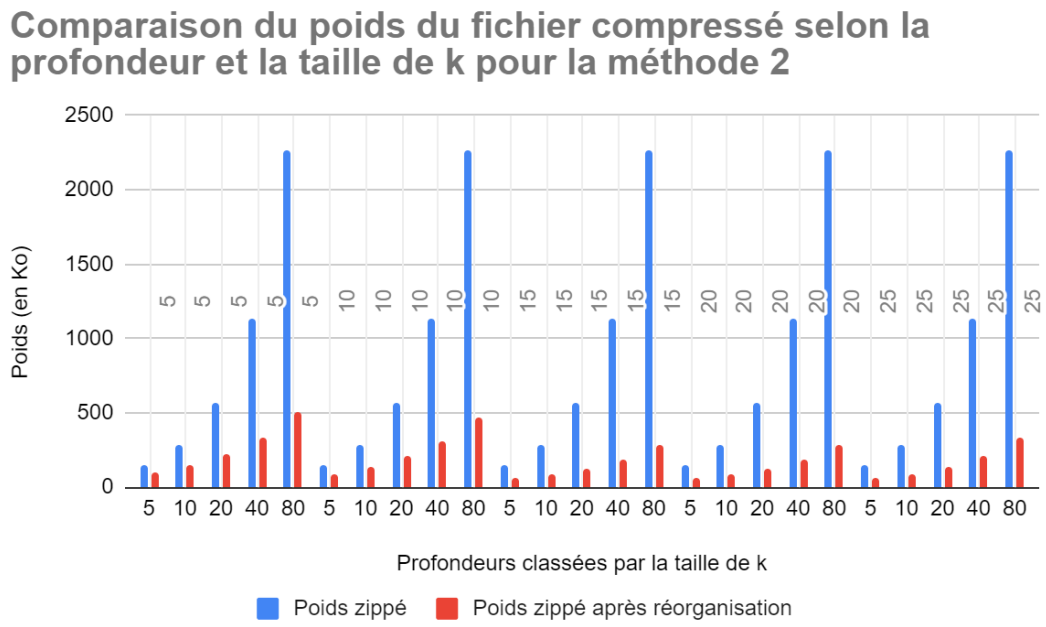

L’impact de k sur le temps de calcul est léger pour la méthode 2. On observe toujours une diminution du temps de calcul grâce à la vitesse du pattern matching puis une augmentation due au nombre de positions à tester pour les petits pattern. Toutefois le temps d'exécution reste raisonnable est de l’ordre de la vingtaine de secondes pour des k faibles.

### c. Effet de la profondeur sur le temps de calcul




La profondeur de séquençage des fichiers de reads provoque une croissance du temps de calcul proportionnelle au nombre de reads. Pour le fichier “humch1_100Kb_reads_80x.fasta” avec une profondeur de séquençage de 80, l'exécution se fait en 4 min 46. On reste dans des ordres de grandeur de quelques minutes pour de grandes profondeurs.

### d. Taux de compression




Ce graphique montre le poids du fichier compressé après sa réorganisation par la méthode 2 en rouge et le poid zippé sans réorganisation en bleu pour servir de référence. On constate toujours que le rapport de compression est plus élevé pour les fichiers séquencés avec une grande profondeur. C’est logiquement les fichiers où il y a le plus de poids à gagner. Cependant les facteurs de compressions sont relativement similaires peu importe la taille de k.

| Taille de k | 5 | 10 | 15 | 20 | 25 |
|---|---|---|---|---|---|
| **Poids compressé sans réorganisation (en Ko)** | 2269 | 2269 | 2269 | 2269 | 2269 |
| **Poids compressé avec réorganisation (en Ko)** | 507 | 472 | 288 | 281 | 333 |
| **Rapport de compression** | 4,5 | 4,8 | 7,8 | 8 | 6,8 |

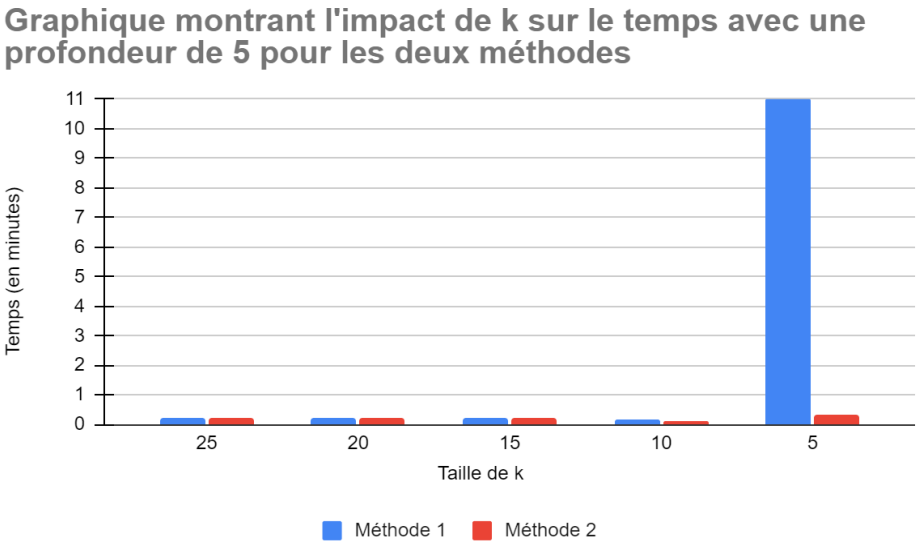
Tableau récapitulatif du rapport de compression pour une profondeur de 80 :

### e. Conseils d'utilisation

L'utilisation de la méthode 2 pour réorganiser les reads est pertinente pour toute taille de k. Le taux de compression est peu affecté par des k faibles.

## 3. Comparaison des deux méthodes

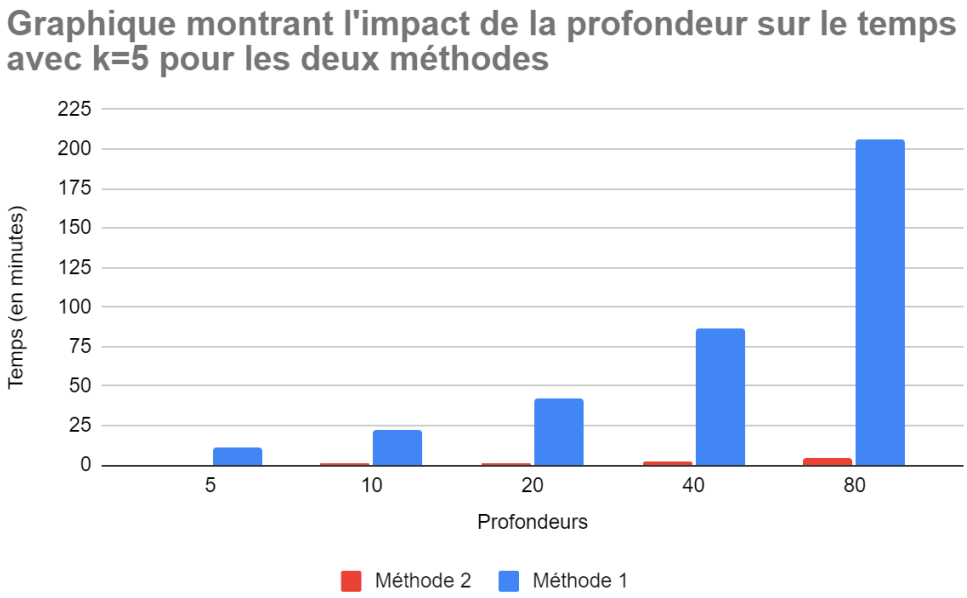
### a. Variation de la taille de k



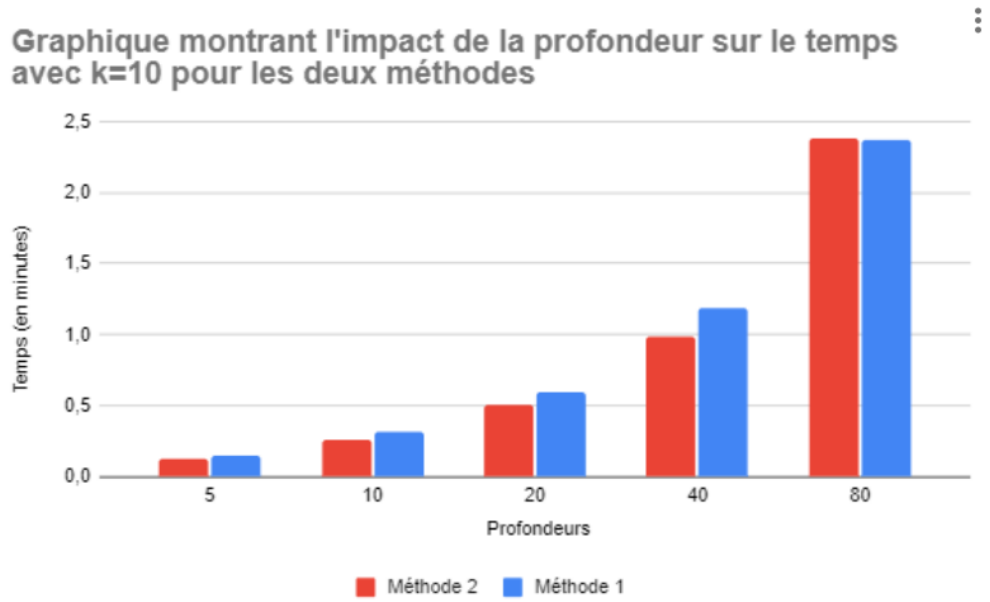

L'observation est sans équivoque : la méthode 2 prend bien mieux en charge la réorganisation pour un k faible. L'utilisation de la méthode par fréquence réduit drastiquement le nombre de positions potentielles et est donc très efficace pour des k faibles qui induisent le parcours de nombreuses sous-séquences pour la méthode 1. L'écart entre les deux méthodes ne fait que s'accroître au fur et à mesure qu'on augmente la profondeur.

### b. Variation de la profondeur

Nous avons vu précédemment une croissance du temps de calcul cohérente avec l'augmentation du nombre de reads à traiter. Les deux méthodes réagissent de la même manière pour les profondeurs, sauf dans le cas des k faibles. C'est pourquoi les deux méthodes sont comparées avec k = 5.

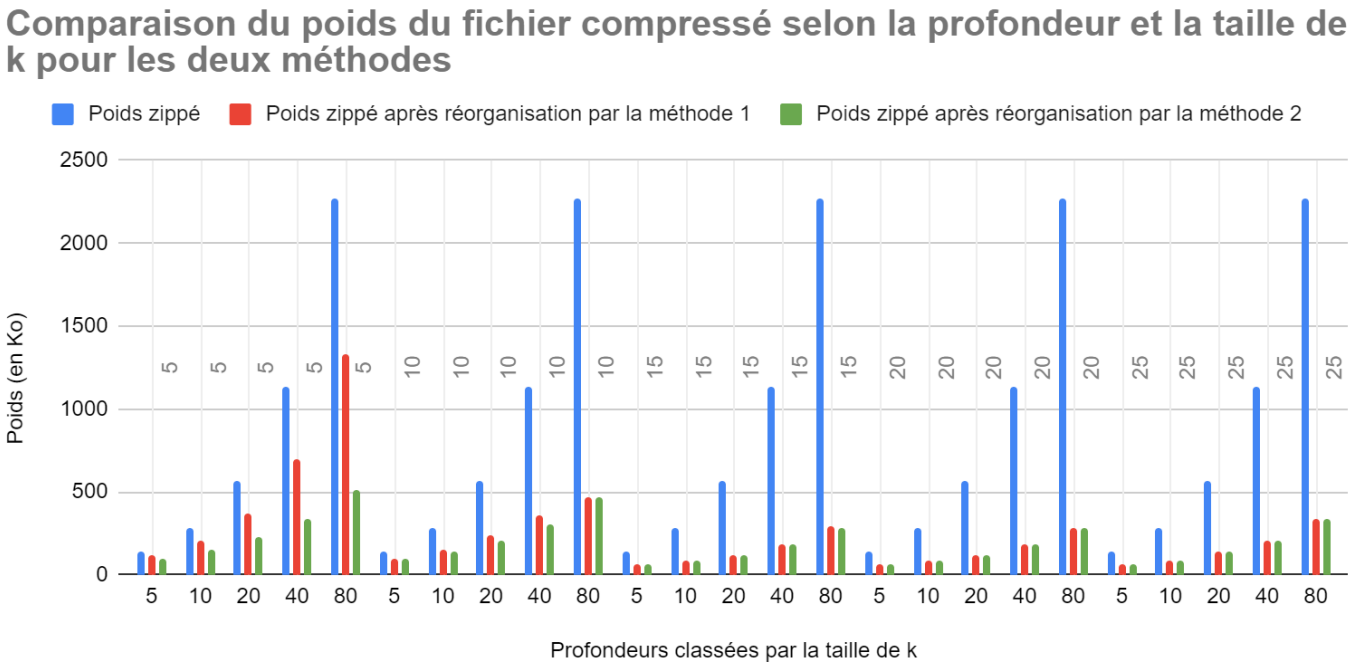



À l'observation de ce graphique, on constate bien un écart de temps de plus en plus conséquent entre les deux méthodes pour un k faible.




Toutefois les deux méthodes sont relativement similaires pour des k moyens et grands, ici k = 10. On remarque un léger avantage pour la méthode 2 mais avec des écarts temporels bien moins conséquents.

### c. Comparaison de la compression des deux méthodes




Ce graphique résume le résultat de toutes les compressions pour toutes les profondeurs avec tous les k étudiés :

- en bleu, le poids compressé des fichiers sans réorganisation
- en rouge, le poids compressé des fichiers réorganisés selon la méthode 1
- en vert, le poids compressé des fichiers réorganisés selon la méthode 2

Les compressions observées pour des k moyens et grands sont similaires pour les deux méthodes. Cependant on note un léger avantage pour la méthode 2 pour k = 10 et une nette différence pour k = 5.

## Conclusion

La méthode 2 est meilleure ou équivalente à la méthode 1 sur tous les aspects : un temps d'exécution bien plus rapide et une meilleure compression pour des k faibles. Pour les k moyens et les k grands la méthode 2 reste équivalente si ce n’est meilleure en terme de performance. Dans le cas de données de séquençage de mauvaise qualité issues d’un séquençage avec une faible précision, la méthode 2 est particulièrement pertinente car ces données nécessitent de réduire la taille de k pour obtenir un mapping correct. Attention cependant à ne pas trop réduire le k, sinon le mapping ne sera pas approprié et le temps de réarrangement prendra un temps très conséquent.