Université de Rennes 1

Master 2 Bio-informatique (BIS)

UE Algorithmique des séquences

2021 - 2022

**Jacky AME et Charles BROTTIER**

# RAPPORT SCIENTIFIQUE

# Représentation compressée et indexation d'un ensemble de k-mers d'un jeu de données de séquençage

Responsables d'UE :

Claire Lemaitre et Pierre Peterlongo

# Sommaire

# 1 Contexte et problématique

## 1.1 Contexte

Le séquençage à haut débit produit aujourd'hui des données, qui, sous forme brute, représentent un espace mémoire conséquent. L'amélioration des capacités de stockage évoluant peu comparée au développement rapide des techniques et méthodes dédiées au séquençage lui-même, il devient crucial d'optimiser l'espace mémoire utilisé par les données produites, c'est-à-dire de les compresser (sans perte d'information).

Une des clés de ce processus à trait à la redondance des données de séquençage. Celles-ci étant produites à haut degré de couverture, cela entraîne une multiplicité de lectures couvrant une même portion de génome. Il en découle intuitivement que cette redondance doit être prise en compte lors du traitement des données. Concernant l'analyse des données de séquençage, et notamment, par exemple, dans des contextes d'assemblage de génomes, celle-ci est maintenant quasi systématiquement abordée via des approches basées sur la construction de k-mers et d'unitigs. Ces deux concepts/outils clés ont largement été développés afin d'améliorer les performances du traitement des données de séquençage. Dans la section suivante, nous commencerons par rappeler brièvement des définitions de ces termes et des termes dérivés ou associés importants.

## 1.2 Définitions

Un **k-mer** est une sous-séquence $s$ contenue dans une séquence $S$, telle que $|s| < |S|$ et $|s| = k$.

Un k-mer est dit **canonique** si sa séquence est la première dans l'ordre lexicographique, comparativement à sa séquence complémentaire inverse. Par exemple, la forme canonique du k-mer '**TCG**' est '**CGA**'. Ne stocker que les k-mers canoniques permet de réduire l'espace de stockage utilisé, les k-mers non canoniques pouvant facilement être reconstruits au moment opportun.

Un k-mer est dit **solide** s'il apparaît fréquemment dans le génome considéré. Un seuil est ainsi fixé pour les définir. L'intérêt réside dans le fait que plus un k-mer est solide, moins il a de chances d'être erroné, contrairement à un k-mer dit "faible" (peu fréquent). La sélection de k-mers solides est l'une des principales méthodes de correction des erreurs de séquençage, d'où son importance. Dans le cas présent, c'est un seuil de fréquence **absolue** qui est utilisé.

Un set de k-mers extraits de lectures de séquençage peut être considéré comme un graphe

de de Bruijn, avec les k-mers comme noeuds et leurs chevauchements comme arêtes. Ceux-ci ne sont donc représentés qu'une seule fois, ce qui permet de réduire la redondance des données. Il demeure que le stockage des k-mers bruts demande k lettres par k-mer, soit 2k bit pour des séquences d'ADN, soit un espace de stockage qui croît rapidement avec la taille des k-mers. Pour réduire cet espace, les concepts d'unitig et Spectrum-preserving string set ont été développés.

Un **unitig**, en considérant un graphe de Bruijn défini tel que précédemment, est une séquence obtenue par la fusion des k-mers d'un chemin sans branchements de ce graphe. Autrement dit, un unitig $u$ est une forme compressée de la séquence $U$, telle que les k-mers de $u$ sont des séquences unitaires entières dans $U$. Par exemple, de la séquence non compressée $U$ = ATC,TCG,CGT, l'on obtient $u$ = ATCGT. Cela permet de représenter |u| k-mers par |u| + k − 1 caractères au lieu de k|u| caractères. Plus les unitigs sont longs, plus l'espace économisé est important. Un **unitig maximal** est défini comme un unitig qui ne peut plus être étendu par aucune de ses deux extrémités.

Les unitigs (maximaux ou non) peuvent eux-mêmes être considérés comme un type de séquences appelées **Spectrum-preserving string sets** (SPSS), soit des séquences dont les sous-séquences sont les k-mers issus des lectures. De nombreux travaux ont cours pour trouver des algorithmes efficaces dans la création de SPSS de tailles minimales, en particulier par rapport aux simples unitigs. Par "efficaces", deux aspects principaux sont entendus : l'efficacité en temps et l'efficacité en mémoire (espace de stockage), l'une allant souvent au détriment de l'autre.

Ce rapport présente l'approche que nous avons mise en place dans l'étude de cette problématique.

# 2 Construction de la SPSS

Bien qu'en cherchant à optimiser l'efficacité en mémoire, notre démarche s'est voulue principalement centrée sur l'efficacité en temps, d'où l'implémentation d'algorithmes à caractère "glouton".

## 2.1 Résumé

Pour construire la SPSS, nous avons suivi les quatre étapes principales suivantes, à partir des données de séquençage :

• extraction des **k-mers canoniques solides** ;

• génération ”gloutonne” de tous les **unitigs maximaux** à partir de ces k-mers ;

• **compression** ”gloutonne” des tous les unitigs maximaux ;

• enfin, concaténation des unitigs maximaux compressés pour former la SPSS.

À noter que c'est la SPSS ainsi produite qui est utilisée pour l'analyse des variations de temps de requête avec ou sans index. Nous avons néanmoins mis en place une étape supplémentaire pour la **compression maximale** de cette SPSS, afin de maximiser le gain d'espace de stockage. Cette compression, plus poussée, ne permet cependant plus de requêtes telles qu'implémentées.

## 2.2 Détail de la procédure

À des fins de rapidité d'exécution, la structure de stockage préférentiellement utilisée pour conserver tout ensemble de séquences d'intérêt est le set python. La SPSS est quant à elle représentée en format texte (par un objet python de type str).

### 2.2.1 Extraction des k-mers canoniques solides

Chaque lecture est validée avant d'être traitée (ne doit pas être vide ni contenir d'autres caractères que A, T, G ou C). Tous les k-mers sont extraits et réduits à leur forme canonique afin d'économiser l'espace de stockage. Leurs fréquences absolues sont calculées dans le même temps, et seuls les k-mers égalant ou dépassant le seuil de solidité défini sont retenus.

### 2.2.2 Génération des unitigs maximaux

De façon gloutonne, tous les k-mers sont étendus en unitigs maximaux.

Pour cela, nous avons repris le concept et le mode de construction de **”simplitigs”**, tels que définis par Brinda et al. (Genome Biology, BioMed Central, 2021). Nous continuerons cependant à nous y référer par le terme d'unitigs maximaux.

Chaque k-mer est d'abord étendu dans le sens *forward*, l'extension est transformée en sa complémentaire inverse, puis elle-même étendue dans le sens *forward*. Au fur et à mesure, les k-mers utilisés sont retirés du set, jusqu'à ce que celui-ci soit vide. Soulignons qu'il n'est pas nécessaire d'inverser à nouveau l'extension finale, puisque nous travaillons avec des k-mers canoniques et que leurs recherches sont faites en double (canonique et non-canonique).

### 2.2.3 Compression des unitigs maximaux pour former la SPSS

De façon gloutonne, les unitigs maximaux sont d'abord compressés avant d'être concaténés.

Pour cela, les unitigs partageant une même séquence "ancre", l'un comme suffixe, l'autre comme préfixe, sont recherchés et fusionnés. Ainsi, l'on économise à chaque fusion un nombre de caractères égal à la taille de l'ancre. Partant de deux séquences 'AAAAAATGC' et GG-GAAAAAA', d'une longueur totale de 18 caractères, et partageant une ancre de taille 3, on obtient la séquence 'GGGAAAAAATGC', de taille 18 - 6 = 12.

L'important dans cette procédure est de procéder en commençant par fusionner les unitigs partageant l'ancre la plus longue possible, puis de recommencer en diminuant la taille de l'ancre (d'un caractère) à chaque itération. Après la dernière fusion (ancre de taille 1), les unitigs ainsi compressés sont concaténés pour former la SPSS. Aucun k-mer original n'est "perdu" lors de cette procédure.

Pour déterminer la taille maximale de l'ancre, nous avons empiriquement vérifié (au moyen d'algorithmes recherchant la taille des chevauchements maximaux au sein des sets de k-mers, non présentés ici) que celle-ci était toujours égale à | k-mers | - 2. Ainsi, pour des k-mers de taille 55, 39 et 31, les tailles maximales des ancres observées furent systématiquement de 53, 37 et de 29, respectivement.

### 2.2.4 Compression maximale de la SPSS

Pour un gain d'espace mémoire maximal, nous avons implémenté une fonction qui réduit toute sous-séquence *s* composée d'un même caractère *c* se succédant *n* fois en la remplaçant par *c* * *n*. Par exemple, la séquence 'AAAAAA' devient 'A6'. Cette fonction s'exécute en 0,05 seconde sur notre ordinateur test, peu importe la taille de la SPSS (du moins dans le cadre de nos essais).

# 3 Analyse

Cette section se consacre à l'analyse de notre programme sous deux critères évalués en fonction de la taille des k-mers (21, 31, 41, 51 et 61) et du seuil de solidité (de 2 à 7) :

• la validité de la SPSS, mesurée en s'intéressant au nombre de faux positifs (FP) et au nombre de faux négatifs (FN) par rapport au génome de référence ;

• la mise en regard du taux de compression de la SPSS et de son temps de construction.

## 3.1 Validation de la SPSS en fonction des taux de FP et de FN

La figure 1 présente le nombre de faux positifs et de faux négatifs selon la taille des k-mers $k$ et du seuil de solidité $t$.

On constate que le nombre de **faux positifs** diminue :

• progressivement quand la taille des k-mers augmente (de 8800 FP environ pour $k = 21$ à environ 3000 pour $k = 61$) ;

• et très rapidement quand le seuil de solidité $t$ augmente : en effet, dès que l'on passe à 3, on atteint un maximum d'environ 52 FP pour $k = 21$, 0 pour $k = 61$. À $t = 4$, il n'y a plus aucun FP, peu importe la taille des k-mers.

Ainsi, plus la taille des k-mers augmente, plus un seuil de solidité bas entraîne une chute drastique du taux de faux positifs. Un seuil de solidité fixé à 3 ou plus semble donc convenir pour éviter les faux positifs. Un seuil de solidité égal à 2 peut être aussi considéré acceptable en augmentant la taille des k-mers (k = 51).

Concernant les **faux négatifs**, on observe que leur nombre augmente :

• progressivement quand la taille des k-mers augmente (à $t = 3$, d'environ 3600 pour $k = 21$ à environ 14,700 pour $k = 61$) ;

• de façon exponentielle avec la hausse du seuil de solidité, ce qui se remarque surtout à une taille de k-mer élevée. Leur taux dépasse les 5% (du nombre de k-mers solides) pour les valeurs suivantes :

- k >= 31 et t >= 6 ;

- k >= 41 et t >= 5 ;

- k >= 51 et t >= 4 ;

- k >= 61 et t >= 3.

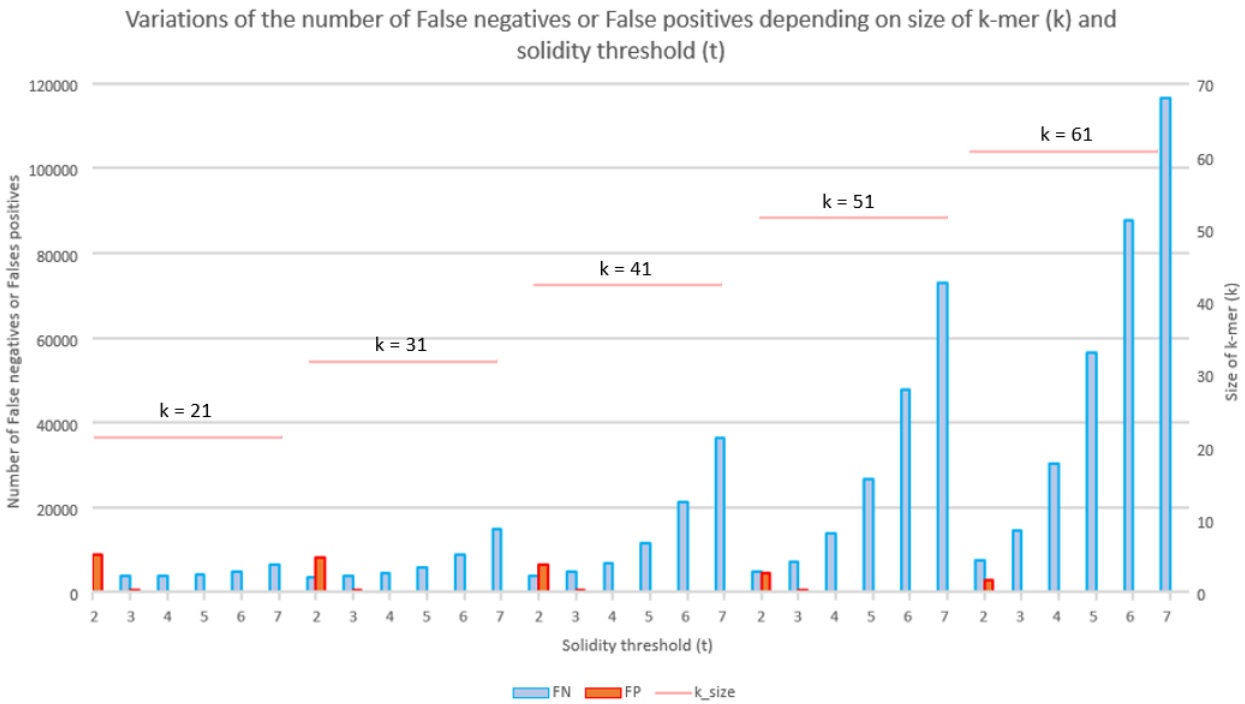


**Figure 1 : Variation du nombre de faux positifs et du nombre de faux négatifs en fonction de la taille des k-mers (k) et du seuil de solidité (t).** Le nombre de faux négatifs augmente considérablement à la fois en fonction de *k* et de *t*. Le nombre de faux positifs décroît quant à lui surtout avec le seuil de solidité.

Ainsi, plus la taille des k-mers augmente, plus un seuil de solidité bas entraîne une hausse drastique du taux de faux négatifs. Ils peuvent atteindre des valeurs supérieures à 25% pour $k = 51$ et $t = 5$, et supérieure à 30% pour $k = 61$ dès $t = 4$.

En considérant les taux de faux négatifs et de faux positifs, l'on pourrait revoir la valeur du seuil de solidité par défaut à $t = 3$, tandis que la taille des k-mers (31) ne gagne pas à être augmentée.

## 3.2 Taux de compression et temps de construction de la SPSS

Le taux de compression est exprimé en tant que pourcentage de la SPSS construite en concaténant directement les unitigs maximaux **non compressés** après extension de tous les k-mers. Ainsi, un taux de compression de 80% indique la production d'une SPSS de 80 caractères, contre 100 caractères si l'on avait procédé à la concaténation directe des unitigs.

La figure 2 montre les valeurs moyennes du taux de compression et du temps de construction de la SPSS en fonction de la taille des k-mers (pour un seuil *t* allant de 2 à 7). La figure 3 montre ces valeurs en fonction du seuil de solidité (pour des tailles de k-mers allant de 21 à 61, avec un pas de 10).

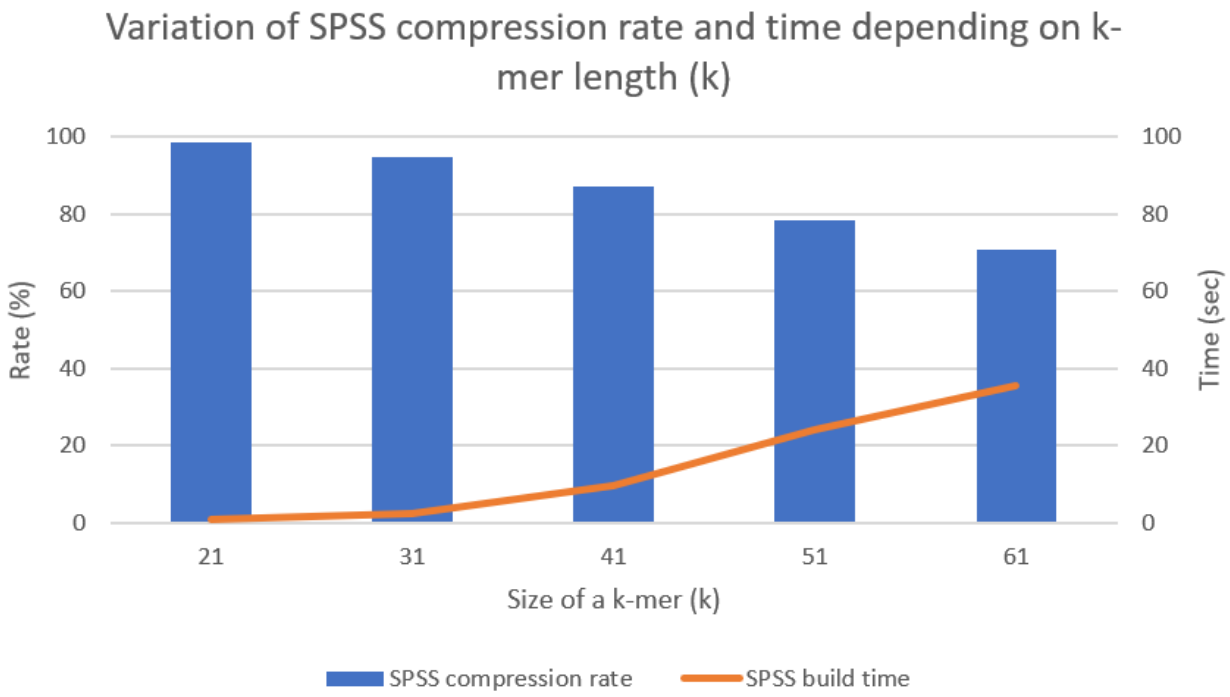


**Figure 2 : Variation du taux de compression et du temps de construction de la SPSS en fonction de la taille des k-mers (k).** Plus celle-ci augmente, plus le taille de la SPSS diminue. À l'inverse, le temps de compression augmente en fonction de k (notamment à partir de k = 41).

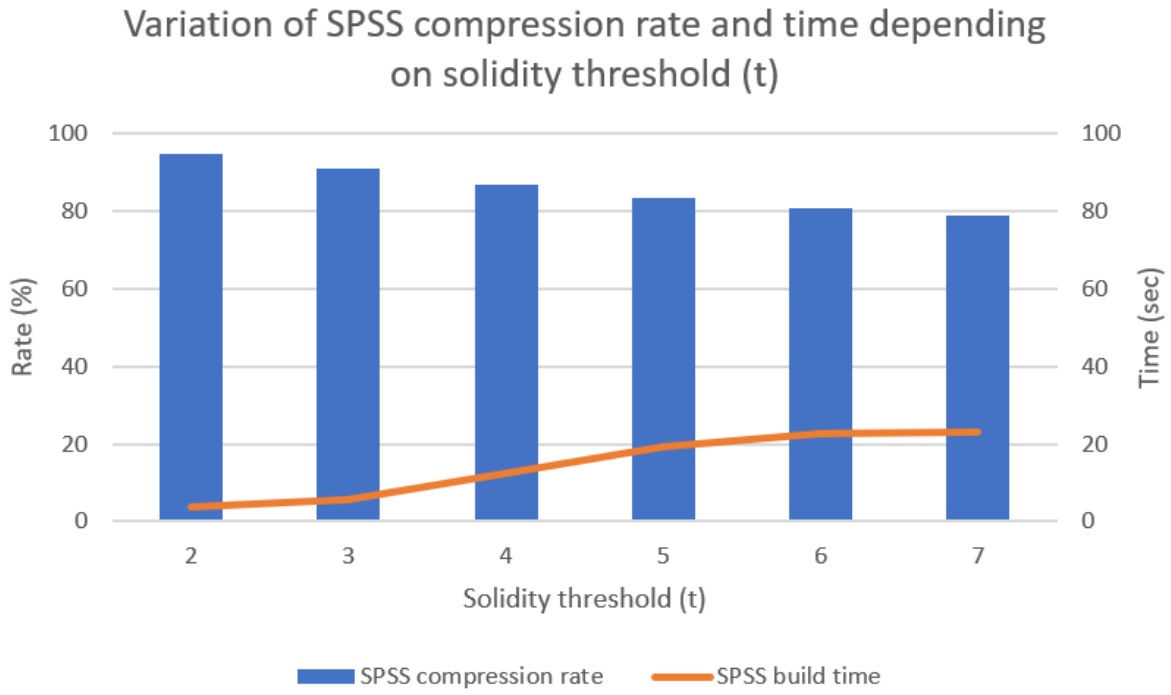


**Figure 3 : Variation de la compression et du temps de construction de la SPSS en fonction du seuil de solidité (t).** Elles suivent les mêmes tendances qu'en fonction de la taille des k-mers, mais de façon moins marquée. Le temps de compression de la SPSS semble se stabiliser à partir de t = 6.

Tout d'abord, on observe un meilleur taux de compression de la SPSS pour des valeurs de $k$ et de $t$ de plus en plus élevées. Cependant le seuil de solidité montre une influence moins importante que la taille des k-mers (figure 3, 80% en moyenne pour notre $t$ max, contre environ 70% pour notre $k$ max).

Le temps de construction de la SPSS, quant à lui, augmente sensiblement à partir de $k = 41$ (figure 2), et à partir de $t = 4$. En revanche, dans le cas du seuil de solidité, son augmentation semble cesser à partir de $t = 6$, alors qu'il continue de croître avec la taille des k-mers. Nous l'expliquons parce qu'à hautes valeurs de $t$, le nombre de k-mers canoniques solides diminue de plus en plus, et ce, d'autant plus que $k$ augmente également.

En résumé, le taux de compression augmente avec taille des k-mers et avec la solidité, mais cela se fait au détriment du temps de construction. Nous avons atteint un taux de compression maximal de 65% pour $k = 61$ et $t = 5$. En pratique, concernant la taille des k-mers, on observerait sans doute un gain plus "rentable" de taux de compression (c'est-à-dire sans trop pénaliser le temps de construction) avec des tailles lectures plus longues (seulement de 100pb dans notre étude, contre 150pb et plus en pratique). Ceci est néanmoins à nuancer en fonction des applications, car la taille optimale des k-mers n'augmente pas forcément de façon linéaire avec la taille des lectures, par exemple dans un contexte d'assemblage.

### 3.3 Temps de requête avec et sans FM-index

On observe une réduction drastique (pratiquement d'un facteur 10) du temps de requête avec l'utilisation du FM-index (voir figure 4 ci-après). On observe :

- en moyenne 5,5 secondes pour les faux positifs, contre 43 secondes sans index ;
- en moyenne 6.4 secondes pour les faux négatifs, contre 62 secondes sans index.

## 4 Améliorations

Premièrement, notons que la taille de la SPSS varie quelque peu pour des valeurs de $k$ et de $t$ identiques, du fait de sa construction de façon gloutonne, et n'est donc pas toujours la plus petite possible d'après les fonctions implémentées. En outre, nous sommes certainement bien loin de la taille minimale absolue possible pour cette SPSS. Nous avons essayé des algorithmes pour l'atteindre, mais n'avions pas les ressources nécessaires pour aller au bout de leur exécution : notre IDE "crashe" tout simplement en raison d'un manque de mémoire vive (pourtant 16Go de

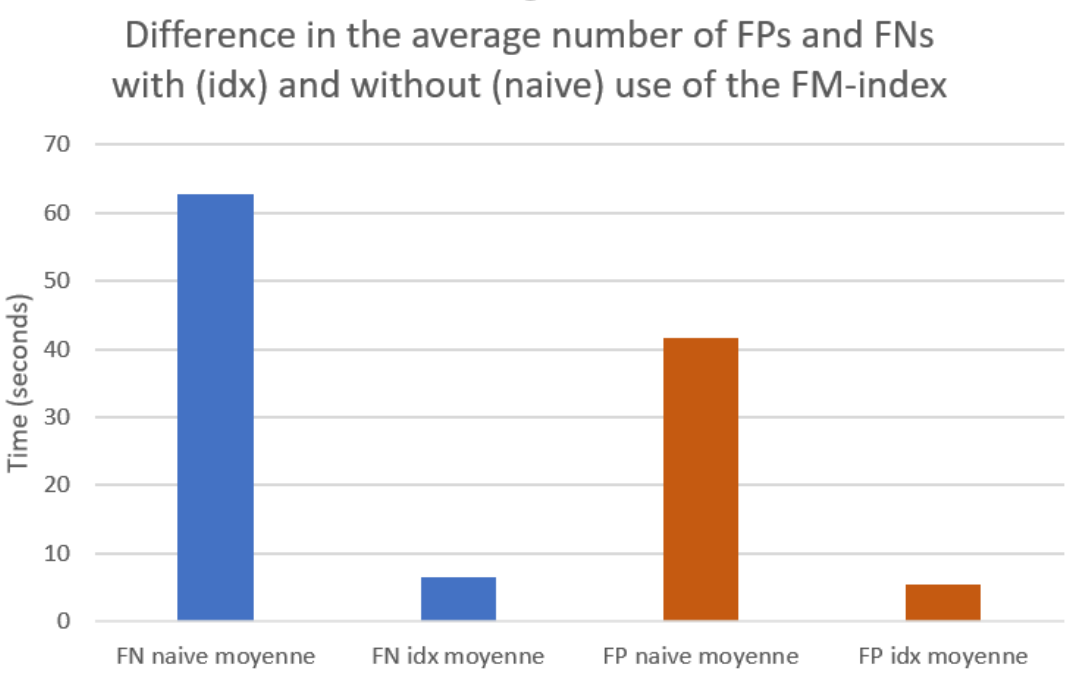


**Figure 4 : Variation du nombre moyen de faux positifs et de faux négatifs avec et sans utilisation du FM-index.**

RAM, avec un processeur Ryzen 5 6 coeurs, 3.80 GHz). Du reste, cette procédure, même sans se terminer, semblait déjà prendre beaucoup plus de temps que notre approche gloutonne.

Ensuite, bien qu'ayant implémenté une méthode de compression "maximale" (dans la limite des capacités de nos algorithmes gloutons), qui permet d'atteindre un taux de compression de 61% à $k = 61$ et $t = 5$ (contre 65% pour ces valeurs, avec le mode de compression initial), nous n'avons pas pu utiliser la SPSS ainsi produite, ce en raison de l'introduction de caractères numériques dans la séquence. Cela entraîne une impossibilité de requête des k-mers, notamment via le FM-index. Pour y remédier, nous aurions pu soit imaginer appliquer ce mode de compression aux k-mers eux-mêmes avant les requêtes, soit réussir à implémenter la décompression de cette SPSS minimale avant la requête. La première idée entraîne cependant un nombre d'opérations supplémentaires égale aux nombres de k-mers ; et nous avons tout simplement échoué à la seconde, faute de temps.

# 5 Conclusion

En conclusion, nous avons pu mettre en évidence l'importance de l'utilisation d'un index pour améliorer le temps de requête des k-mers (gain de l'ordre d'un facteur 10). Nous avons par ailleurs réussi à mettre en place des méthodes de compression relativement efficaces (gain de 10 à 30%) sans trop perdre en temps de calcul (20% de compression gagnés pour $k = 51$ et $t = 4$ en perdant 10 secondes par rapport à 2% gagnés pour les valeurs par défaut de $k = 31$ et $t = 2$).