Mémoire de stage
Bioinformatique, découverte de motifs entre des ensembles de fragments d'ADN

Gabriel Toublanc, Master IGIS ITA, 2^{eme} année

30 Juin 2017

Université de Rouen, U.F.R des Sciences et Techniques de Saint-Etienne-du-Rouvray, LITIS EquipeTIBS

Encadrants : Thierry Lecroq et Arnaud Lefebvre

Table des matières

Introduction

Cadre

Ce mémoire a été rédigé dans le cadre de mon stage de fin de Master IGIS ITA, dans le but de réaliser un rapport complet sur l'ensemble du travail effectué. Ce stage s'est déroulé du 1^{er} avril au 30 juin 2017, au sein de l'équipe TIBS du laboratoire LITIS de l'Université de Rouen, dans le bâtiment Monod puis CuriB sur le campus de Mont-Saint-Aignan.

Les travaux de cette équipe portent principalement sur la recherche, l'indexation et l'extraction d'informations pertinentes dans des données biologiques et des systèmes d'information en santé, et offrent donc de nombreuses applications dans ces domaines.

Ce stage a été financé par le LITIS, grâce à des fonds alloués par l'Université de Rouen, destinés à permettre l'accueil de stagiaires de seconde année de Master Recherche.

Contexte

Les séquenceurs ADN ont connu un développement important depuis le milieu des années 2000, ce qui a permis à la bioinformatique de développer de nouvelles techniques d'analyse.

Le séquençage ADN est constitué de deux principales familles de lectures :

— **Les lectures courtes** (Illumina, Roche, ...)
— **Les lectures longues** (Nanopore, PacBio, ...)

Les lectures courtes ont le net avantage d'être très précises, avec un taux d'erreur de moins de 1 %, la plupart étant des erreurs de substitution. Leur principal inconvénient est que de nombreuses lectures courtes sont nécessaires afin de reconstruire le génome, ce qui est coûteux et peu pratique.

Les lectures longues, en revanche, ont l'avantage de ne nécessiter que d'une petite quantité d'entre elles pour couvrir le génome d'un être vivant, mais le gros désavantage d'être hautement imprécises (15 à 30 % d'erreurs), principalement des erreurs d'insertions et de délétion.

Dans les faits, le génome peut être reconstruit grâce à un nombre déraisonnable de lectures courtes, qui produiront un génome trop fragmenté, ou en assemblant les lectures longues corrigées par les lectures courtes. Plusieurs méthodes de corrections ont été développées à ce sujet [1].

L'étude des k-mers des lectures longues a pour but de pouvoir corriger les lectures longues elles-mêmes, et ainsi de pouvoir se passer des lectures courtes.

Chapitre 1

Extraction des k-mers

1.1 K-mers

Les k-mers sont les facteurs de longueur **k** d'une séquence ADN donnée. Pour une lecture de longueur **L**, nous avons donc $\mathbf{L - k + 1}$ k-mers possibles.

Ex : Avec une séquence $x = AACCGGTT$, nous obtenons les k-mers de longueur 6 (6-mers) suivants :

$\mathbf{k_1}$: $A\ A\ C\ C\ G\ G\ T\ T$

$\mathbf{k_2}$: $A\ A\ C\ C\ G\ G\ T\ T$

$\mathbf{k_3}$: $A\ A\ C\ C\ G\ G\ T\ T$

6-mers : {AACCGG, ACCGGT, CCGGTT}

Afin d'énumérer les k-mers de grands jeux de données, divers outils logiciels ont été développés. L'objectif de ces outils est d'extraire les k-mers de façon à grouper les doublons, d'avoir un temps d'exécution viable et d'utiliser le moins d'espace mémoire possible.

Le logiciel **Jellyfish** [2] surpasse ses concurrents dans ce domaine, avec des performances bien supérieures, tout facteur confondu.

Les k-mers ainsi extraits sont principalement utilisés pour l'alignement et l'assemblage de lectures.

Dans le cas présent, l'utilisation des k-mers a pour but de trouver des répétitions au sein des lectures longues afin de pouvoir identifier les nucléotides erronées, afin de pouvoir les corriger.

1.2 K-mers espacés

Les k-mers espacés sont utilisés afin de simuler des corrections aux erreurs d'insertions et de délétions sur les lectures longues.
Au lieu de prendre les facteurs de longueur k d'une séquence, nous allons sélectionner les k-mers espacés selon un motif précis.

1.2.1 K-mers espacés à délétion

Avec les k-mers espacés à délétion, chaque **0** du motif correspond à une nucléotide à supprimer.
Par ex, un motif $\mathbf{m = 111011}$ créera tous les 5-mers espacés à partir des 6-mers en supprimant le 4^{eme} nucléotide.

Ex : Avec une séquence $x = AACCGGTT$ et le motif $m = 111011$, nous obtenons les 5-mers espacés suivants :

$\mathbf{k_1}$: $A\ A\ C$ ~~C~~ $G\ G\ T\ T$

$\mathbf{k_2}$: $A\ A\ C\ C$ ~~G~~ $G\ T\ T$

$\mathbf{k_3}$: $A\ A\ C\ C\ G$ ~~T~~ $T\ T$

5-mers : {AACGG, ACCGT, CCGTT}

1.2.2 K-mers espacés à insertion

Avec les k-mers espacés à insertion, chaque **0** du motif correspond à une nucléotide à insérer.
Toutes les nucléotides possibles pour chaque **0** sont extraites, ce qui produit $(\mathbf{L - k + 1}) * \mathbf{4^t}$ k-mers espacés possibles, avec t = nombre

de zéros dans le motif.

Ex : Avec une séquence $\mathbf{x} = \mathbf{AACCGGTT}$ et le motif $\mathbf{m} = \mathbf{111011}$, nous obtenons les 6-mers espacés suivants :

$$
\begin{array}{lcccccccc}
\mathbf{k_1}: & A & A & C & \overbrace{A,C,G,T} & C & G & G & T & T \\
\mathbf{k_2}: & A & A & C & C & \overbrace{A,C,G,T} & G & G & T & T \\
\mathbf{k_3}: & A & A & C & C & G & \overbrace{A,C,G,T} & G & T & T \\
\mathbf{k_4}: & A & A & C & C & G & G & \overbrace{A,C,G,T} & T & T
\end{array}
$$

$$
\textbf{6-mers} : \left\{ \begin{array}{llll}
\text{AAC}\mathbf{A}\text{CG}, & \text{ACC}\mathbf{A}\text{GG}, & \text{CCG}\mathbf{A}\text{GT}, & \text{CGG}\mathbf{A}\text{TT}, \\
\text{AAC}\mathbf{C}\text{CG}, & \text{ACC}\mathbf{C}\text{GG}, & \text{CCG}\mathbf{C}\text{GT}, & \text{CGG}\mathbf{C}\text{TT}, \\
\text{AAC}\mathbf{G}\text{CG}, & \text{ACC}\mathbf{G}\text{GG}, & \text{CCG}\mathbf{G}\text{GT}, & \text{CGG}\mathbf{G}\text{TT}, \\
\text{AAC}\mathbf{T}\text{CG}, & \text{ACC}\mathbf{T}\text{GG}, & \text{CCG}\mathbf{T}\text{GT}, & \text{CGG}\mathbf{T}\text{TT}
\end{array} \right\}
$$

1.3 État de la recherche sur les k-mers

Parmi les différents logiciels qui permettent l'extraction des k-mers simples, nous avons cité l'outil **Jellyfish** [2] qui est de très loin le plus efficace (500Mo en moins de 15s, avec 8 threads), et qui est spécialisé dans l'approche *multi-threading*.

En revanche, en ce qui concerne les k-mers espacés, les outils sont encore peu nombreux et inefficaces. Dans le cadre de ce stage de recherche, nous avons pris contact avec une équipe de chercheurs de l'Université de Montpellier. Cette équipe développe actuellement un outil nommé **GkAmpi**[3], permettant de faire de l'extraction de k-mers simples ou espacés à délétion.

Cependant au moment de l'écriture de ce rapport, l'outil est encore au stade experimental, et souffre de performances en vitesse peu satisfaisantes, même s'il profite d'une utilisation RAM très optimale. De plus, le logiciel permet d'extraire les k-mers espacés à délétion, mais pas ceux à insertion.

C'est à partir de cet état de fait que j'ai entrepris de développer mes propres logiciels, **kmersDel** et **kmersExpand**.

Chapitre 2

Implémentation

2.1 K-mers espacés à délétion : kmersDel

kmersDel a pour but d'extraire les k-mers espacés à délétion, en utilisant le principe d'une table de hachage afin de grouper les doublons.

Algorithme 1 : kmersDel

```
   Entrées : table_hachage table, mots lectures, mot motif, entier k
 1 pour chaque lecture de lectures faire
 2 |   pour i = 0; i + k ≤ |lecture|; i+ = 1 faire
 3 |   |   kmerEntier = 0 ;
 4 |   |   kmer = "" ;
 5 |   |   pour j = 0; j < k; j+ = 1 faire
 6 |   |   |   si motif[j] ≠ 0 alors kmer = kmer + lecture[i + j] ;
 7 |   |   fin
 8 |   |   pour chaque nucleotide de kmer faire
 9 |   |   |   kmerEntier∗ = 4 ;
10 |   |   |   suivant valeur de nucleotide faire
11 |   |   |   |   cas où A faire ;
12 |   |   |   |   cas où C faire kmerEntier+ = 1 ;
13 |   |   |   |   cas où G faire kmerEntier+ = 2 ;
14 |   |   |   |   cas où T faire kmerEntier+ = 3 ;
15 |   |   |   fin
16 |   |   fin
17 |   |   table[kmerEntier]+ = 1 ;
18 |   fin
19 fin
```

Pour chaque lecture traitée, nous faisons passer le motif en fe-

nêtre glissante afin de récolter tous les k-mers, puis nous ajoutons le k-mer à la table s'il n'est pas déjà présent. Afin de gagner en espace mémoire, le k-mer est converti en base 4 avant d'être stocké.

La table de hachage a subi diverses améliorations au fil du temps :

— C'était au début un type **map** de la librairie STL du C++, et le programme n'utilisait qu'un seul thread.

— Le développement du *multi-thread* permet de traiter l'information plus efficacement en parallélisant le traitement. Au lieu d'avoir une seule table, le programme créé **t** tables (où **t** est le nombre de threads), qui sont ensuite fusionnées ensemble. La duplication du nombre de tables est dû au fait que les conteneurs de la STL ne sont pas *thread-safe*, c'est à dire que la modification d'un même élément au sein de différents threads donne lieu à des comportements non-définis.

— La table de hachage est devenue une **unordered_map** (c++11), qui a l'avantage d'être quatre fois plus rapide en temps d'exécution que **map**, en contrepartie du fait qu'elle ne trie plus ses éléments par ordre croissant.

— Enfin après diverses recherches, la table est devenue une **cuckoohash_map** qui est un type issu de la librairie indépendante **libcuckoo** [4] [5]. Cette librairie, en plus d'être bien plus performante que les conteneurs de la STL, a pour avantage d'être *thread-safe*, ce qui a permis de ne conserver qu'une seule table pour tous les threads, décuplant alors les performances du programme.

Au niveau de l'affichage, une amélioration en vitesse importante a été effectuée en remplaçant les affichages du programme par un *buffer* qui stocke les lignes en mémoire avant de les afficher.

Avec ce programme simple mais efficace, nous avons réussi à se rapprocher des performances du ténor du genre, Jellyfish. En effet, pour un fichier de 500Mo de lectures, Jellyfish met environ 13 secondes contre 17 secondes pour kmersDel. L'avantage de kmersDel est qu'il permet, contrairement à Jellyfish, de compter les k-mers

espacés à délétion. En contrepartie, kmersDel utilise beaucoup plus de RAM que Jellyfish dès que l'on augmente la longueur de k, ce qui le rend moins utilisable en pratique.

Une version de kmersDel a cependant été développée avec les mêmes priorités que kmersExpand : diminuer l'utilisation mémoire en faveur d'une plus large usabilité. Cette version ne stocke pas les k-mers extraits dans une table de hachage, mais les affiche directement sur la sortie standard, ce qui implique qu'il n'y a pas de traitement des doublons.

Couplé avec Jellyfish, cela permet néanmoins de pouvoir traiter des k-mers de longueur arbitraire.

2.2 K-mers espacés à insertion : kmersExpand

Algorithme 2 : kmersExpand

Entrées : mots *lectures*, mot $motif$, entier k

pour chaque *lecture de lectures* **faire**
  pour $i = 0;\ i + k \leq |lecture|;\ i+ = 1$ **faire**
    $kmer = lecture[i : k]$ $kmersExpandRec(kmer, motif, 0)$
  fin
fin

Algorithme 3 : kmersExpandRec

Entrées : mot $kmer$, mot $nvKmer$, mot $motif$, entier $posMotif$, entier $posKmer$

si $posSeed == |motif|$ **alors** $affiche(nvKmer)$;
sinon si $motif[posMotif] \neq 0$ **alors**
  $kmersExpandRec(kmer, nvKmer + kmer[posKmer], posMotif + 1, posKmer + 1)$;
sinon
  $kmersExpandRec(kmer, nvKmer + A, posMotif + 1, posKmer)$;
  $kmersExpandRec(kmer, nvKmer + C, posMotif + 1, posKmer)$;
  $kmersExpandRec(kmer, nvKmer + G, posMotif + 1, posKmer)$;
  $kmersExpandRec(kmer, nvKmer + T, posMotif + 1, posKmer)$;
fin

L'approche adoptée pour kmersExpand a été différente, en effet, du à la manière dont les k-mers sont construits et à cause de l'explosion de RAM du précédent programme, la technique employée par

kmersDel ne pouvait pas fonctionner ici.

Etant donné qu'il y a quatre nucléotides possible par 0 du motif, nous avons $(L - k + 1) * 4^t$ k-mers possibles, ce qui développe de façon exponentielle les résultats. Afin de ne pas saturer la RAM, l'approche actuelle, malgré son côté récursif, est très peu gourmande en mémoire car elle ne stocke aucun des k-mers obtenus mais les affiche immédiatement. De plus, l'algorithme utilise des *mutex* afin de séparer l'affichage des différents threads (ce qui ralentit davantage l'exécution).

En conséquence, l'algorithme est bien plus lent que kmersDel, cependant, la longueur du fichier n'influe pas sur la quantité de RAM utilisée, ce qui en fait un outil utilisable en pratique.

Chapitre 3

Autres pistes de recherche

3.1 Mot Minimaux Absents (MAW)

[fragile]Mots Minimaux Absents (MAW) Toujours dans l'optique d'identifier les erreurs sur les lectures longues, nous avons étudié les Mots Minimaux Absents.

Définition 1 *Un mot minimal absent d'une séquence est un mot absent dont les facteurs propres (plus long suffixe et plus long prefixe) sont tous présents dans la séquence.*

Ex : Avec la séquence $s = AACACACC$, nous obtenons les mots minimaux absents $\{AAA, AACACC, AACC, CAA, CACACA, CCA, CCC\}$.

Le but de l'extraction des mots minimaux absents est d'observer leur fréquence d'apparition au sein des lectures longues, afin de pouvoir identifier les séquences qui auraient du apparaître et qui sont absentes.

Il y a plus de 13 millions de MAW dans les lectures longues.

Ceux avec une haute fréquence d'apparition (1k-10k) sont assez courts, et sont des séquences apparaissant correctement dans les lectures courtes et le génome de référence, mais n'apparaissant pas dans de nombreuses lectures longues, probablement à cause erreurs d'insertions/délétion.

Ceux avec une faible fréquence d'apparition ($<$ 1k) sont plus

longs et sont, pour beaucoup, des séquences n'apparaissant pas dans les lectures courtes ni dans le génome de référence

3.2 Plus long sous-mot commun (PLSC)

L'étude des sous-mots est analogue à l'étude des insertions/délétion dans les lectures longues. En effet :

Définition 2 *Un mot x est un **sous-mot** d'un mot y s'il existe une factorisation $y = z_0x_1z_1x_2 \cdots x_nz_n$ telle que $x = x_1x_2 \cdots x_n$.*

Par exemple, $x = AAAAC$ est un sous-mot de $y = \mathbf{AA}C\mathbf{A}C\mathbf{AC}C$.

Cela implique qu'on peut obtenir x à partir de la séquence y en supprimant certaines nucléotides, mais également que l'on peut obtenir y à partir de x en insérant certaines nucléotides.

À partir de ce constat, nous allons à présent à chercher à comparer deux lectures longues entre elles grâce à leur PLSC, afin de déterminer quels sont les insertions ou délétion nécessaire afin que ces deux séquences soient similaires.

Définition 3 *Le Plus Long Sous-mot Commun (PLSC) à deux séquences x et y est le mot z tel que z soit le plus long sous-mot à la fois dans x et dans y.*

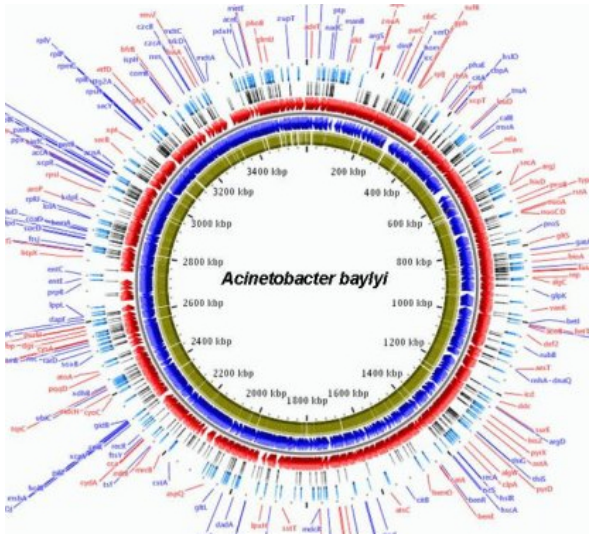
Par exemple, le plus long sous-mot commun aux séquences $x = \mathbf{ACCA}A\mathbf{C}$ et $y = A\mathbf{AC}A\mathbf{CAC}C$ est $ACCAC$
.

En cherchant le PLSC entre une lecture longue brute et une lecture longue corrigée, nous pourrions établir les modifications opérées afin d'avoir pu corriger la lecture initiale.

Chapitre 4

Résultats obtenus

4.1 Données étudiées



Le génome sur lequel se sont portés nos expériences est l'espèce de l'*Acinetobacter baylyi*. Cet organisme possède un génome plutôt petit, d'environ 3 600 000 nucléotides.

Les lectures longues utilisées sont celles issues du séquenceur MinION de Oxford Nanopore. Ce séquenceur a pour intérêt d'être très peu cher comparé aux normes du marché ($1 000 au lieu de plusieurs dizaines de milliers de dollards) et d'être compacte, ne nécessitant qu'un port USB pour fonctionner.

Les lectures longues ainsi extraites ont cependant près de 30% d'erreurs par rapport au génome de référence. En tout, elles ont produit 89 011 lectures longues avec une longueur moyenne de 4300 nucléotides.

4.2 kmersDel et kmersExpand

Ici nous avons utilisé kmersDel et kmersExpand en les groupant avec les k-mers contigus afin de simuler des corrections aux erreurs d'insertion et de suppression des lectures longues.

Il est important de noter qu'énormément des k-mers obtenus dans les lectures longues n'apparaissent pas dans le génome de référence. En effet le génome de référence ne contient, par exemple, que $\simeq$ 3 500 000 16-mers, alors que l'on obtient plus de 1 000 000 000 16-mers dans les lectures obtenues par le séquenceur MinION (0,20% - 0,40%). Cela implique que beaucoup de k-mers ne sont pas utiles à la correction des lectures longues.

Sachant cela, nous avons étudié différents paramètres :

— 16-mers, $freq = 5$, un trou de longueur 1 $\rightarrow$ 87% des bons k-mers trouvés

— 16-mers, $freq = 5$, un trou de longueur 1 à 2 $\rightarrow$ 98% des bons k-mers trouvés

Un fort pourcentage des bons k-mers est donc trouvé, mais ils sont noyés dans la quantité importante de k-mers inutiles, et il n'a pas été possible de les filtrer.

Des résultats similaires ont été observés sur d'autre longueur de k-mers, notamment sur les 20-mers et 11-mers. Enfin, afin de vérifier si l'information extraite serait plus intéressante si les lectures fournies en entrée étaient plus précises, nous avons également remplacé nos données Oxford Nanopore par des lectures longues de Pacific Biosciences, possédant uniquement 15% d'erreur. Cela n'a cependant pas amélioré nos résultats.

4.3 Mots Minimaux Absents (MAW)

Les Mots Minimaux Absents avec une haute fréquence d'apparition dans les lectures longues (1k-10k) sont assez courts, et sont des séquences apparaissant correctement dans le génome de référence. Ils n'apparaissent pas dans un grand nombre de lectures longues, probablement à cause des nombreuses erreurs d'insertion/délétion.

Les Mots Minimaux Absents avec une faible fréquence d'apparition ($<$ 1k) sont plus longs et sont, pour beaucoup, des séquences n'apparaissant pas dans le génome de référence.

Pour les tests suivants, nous éliminons les MAW inclus dans d'autres. La longueur moyenne d'un MAW est de 8,96 nucléotide, et nous récupérons 202 373 MAW.

En ne conservant que les k-mers contenant au moins un MAW, nous n'obtenons pas de bons résultats, presque tous les k-mers contiennent au moins un MAW. Le constat est similaire en filtrant les MAW trop courts (avec une longueur < 9) : seuls 3000 mauvais k-mers des lectures longues ne contiennent aucun MAW. À l'inverse, en augmentant la longueur minimale d'un MAW (même à 10), nous n'obtenons que trop peu de MAW, donc trop de k-mers ne contiennent aucun MAW, et nous passons à côté de beaucoup de bons k-mers.

En regardant le nombre de MAW présent dans chaque k-mer, après filtration des MAW de longueur < 9, nous obtenons une moyenne presque constante de 7 MAW par 16-mer, que les MAW soient recherchés sur des lectures longues, courtes, erronées ou correctes.

En regardant la couverture des k-mers par les MAW, cela n'est pas plus concluant, bons et mauvais k-mers couvrent le génome de référence à peu près de la même façon. Cela s'explique car presque tous les 8-9-10-11-mers sont présents dans le génome de référence.

Nous en avons conclu qu'il faut pouvoir essayer des MAW de plus grandes tailles, ce qui nous tourne donc vers les MAW non-fréquents ou rares. Malheureusement, les MAW rares ne sont pas assez nombreux et sont présent de façon trop homogène dans les bons et les mauvais k-mers, ce qui les rend difficilement utilisable.

4.4 Résultats : CompaReads et Plus Long Sous-mot Commun

Nous avons utilisé un outil nommé CompaReads[6] qui permet de tenter d'aligner des lectures longues avec le génome de référence.

Le but de l'utilisation de cet outil est d'identifier les lectures longues provenant d'une même région du génome de référence.

Afin d'atteindre cet objectif, nous avons suivi la procédure de test suivante :

1. Sélection d'une lecture longue
2. Récupération des lectures longues similaires
3. Recherche de la lecture longue corrigée correspondante
4. Vérifier qu'elles s'alignent bien sur la même région du génome

En suivant cette procédure, nous avons sélectionné deux 20-mers apparaissant fréquemment dans les lectures longues.

Avec cela, nous avons obtenu 25 lectures longues similaires aux deux choisies.

Enfin, nous avons récupéré les lectures longues corrigées correspondantes (corrigées par HG-CoLoR[1]), qui sont au nombre de 11.

Parmi ces 11 lectures corrigées, 8 d'entre elles s'alignent correctement sur la même région du génome.

Nous avons couplé cet outil avec la méthode du Plus Long Sous-Mot Commun. Après avoir calculé le PLSC entre deux séquences, nous pouvons chercher à l'aligner pour tenter de voir si la correction d'une lecture longue pouvait s'apparenter à la création du PLSC.

Nous avons effectué les différents tests suivants :

— Deux lectures longues similaires entre elles : pas d'alignement sur le génome

— Entre 1 lecture brute et son équivalente corrigée : très mauvais alignement

— Entre 2 lectures corrigées similaires : l'alignement s'effectue très bien

Ces résultats montrent que le PLSC entre deux lectures corrigées s'aligne très bien, mais qu'il ne permet pas, à lui seul, d'identifier les erreurs d'insertions/délétion, vu que le PLSC ne fonctionne pas avec les lectures non-corrigées.

Si nous testons avec le PLSC des 32-mers ou des 64-mers des lectures longues similaires, les résultats ne sont pas convaincant non plus, car aucun des PLSC ne parvient à s'aligner.

Chapitre 5

Conclusion

Sur l'ensemble des pistes de corrections des lectures longues elles-mêmes, une grande partie d'entre elles n'ont pas donné les résultats escomptés.

Cependant, étant donné le succès de kmersDel et kmersExpand à trouver presque tous les k-mers du génome, quelques pistes peuvent être encore explorées :

— Marquer les k-mers de leur méthode d'obtention d'origine (ex :5ème nucléotide supprimée, 3ème ajoutée...) afin d'obtenir une 'carte d'identité' du k-mer et d'ainsi pouvoir retracer son origine

— Dans la même veine d'idée, en retraçant l'origine des k-mers, nous pourrions mettre en place un lot de règles précises sur la position moyenne des nucléotides supprimées, ou un motif récurrent dans l'insertion de nucléotide

Les progrès de la technologie MinION de Oxford Nanopore et sa plus grande accessibilité en terme de coûts en font une technologie d'avenir. Oxford Nanopore est toujours en train d'améliorer son produit et fourni même des résultats sur les lectures longues à hauteur de 90% de nucléotides correctes.

Les lectures longues sont également un système de fragmentation de l'ADN qui sera très important dans les années à venir, de par la reconstruction plus aisée du génome, permit par les plus grands chevauchements de ces lectures.

Bibliographie

[1] Pierre Morisse, Thierry Lecroq, and Arnaud Lefebvre. HG-CoLoR : Hybrid-Graph for the error Correction of Long Reads. In *Actes des Journées Ouvertes Biologie Informatique et Mathématiques*, 2017.

[2] Guillaume Marcais and Carl Kingsford. A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics*, 27(6) :764–770, 2011.

[3] Alban Mancheron. GkAmpi. *Personal Communication*, 2017.

[4] Bin Fan, David G. Andersen, and Michael Kaminsky. Memc3 : Compact and concurrent memcache with dumber caching and smarter hashing. In *Networked Systems Design and Implementation*, apr 2013.

[5] Xiaozhou Li, David G. Andersen, Michael Kaminsky, and Michael J. Freedman. Algorithmic improvements for fast concurrent cuckoo hashing. In *EuroSys*, apr 2014.

[6] Nicolas Maillet, Claire Lemaitre, Rayan Chikhi, Dominique Lavenier, and Pierre Peterlongo. Compareads : comparing huge metagenomic experiments. *BMC Bioinformatics 13(Suppl 19)*, 2012.