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 batiment Monod puis CurieB 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 nouvelle technique 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 nécessaire 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 par 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 taille **k** d'une séquence ADN donnée. Pour une lecture de taille **L**, on a donc $\mathbf{L - k + 1}$ k-mers possibles.

Ex : Avec une séquence $x = AACCGGTT$, on a les k-mers de taille 6 (6-mers) suivants :

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

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 taille k d'une séquence, on va 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 $m = 111011$, on a 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 $\mathbf{m} = \mathbf{111011}$, on a 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, chaînes lectures, chaîne 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, on fait passer le motif en fenêtre glis-

sante afin de récolter tous les k-mers, puis on ajoute 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 parrallélisant 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 devenu 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 devenu 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 à 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 performance 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 taille de k, ce qui le rend moins utilisable en pratique.

Une version de kmersDel à 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 taille arbitraire.

2.2 K-mers espacés à insertion : kmersExpand

Algorithme 2 : kmersExpand

Entrées : chaînes *lectures*, chaîne $motif$, entier k

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

Algorithme 3 : kmersExpandRec

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

```
si posSeed == |motif| alors affiche(nvKmer) ;
sinon si motif[posMotif] ≠ 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, on a $(L - k + 1) * 4^t$ 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 kmers obtenus mais les affiche immédiatement. De plus, l'algorithme utilise des *mutex* afin de séparer l'affichage des différents threads (ce qui ralenti davantage l'exécution).

En conséquence, l'algorithme est bien plus lent que kmersDel, cependant, la taille 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 (MMAs)

[fragile]Mots Minimaux Absents (MAWs) Toujours dans l'optique d'identifier les erreurs sur les lectures longues, nous avons étudiés 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 $s = AACACACC$, on obtient 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 aurait dûe apparaître et qui sont absentes.

Il y a plus de 13 millions de MAWs 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

Explication wikipedia + pierre
Resultats pas au rdv

3.2 Plus long sous-mot commun (PLS)

Explication cours de lecroq.
Schema matrice lcs Resultats pas au rdv

Chapitre 4

Résultats obtenus

4.1 kmersDel et kmersExpand

Ici on a utilisé kmersDel et kmersExpand en les groupant afin de reproduire les erreurs d'insertion et de suppression des lectures longues. Différents paramètres ont été essayé :

— 16

1.1 16-mers, un trou d'emplacement variable, freq = 5 => En ne gardant qu'un trou, mais en le déplaçant (ex : 1011, 1101), on obtient des k-mers différents En faisant l'union des 16-mers étendus, et des 16-mers espacés des LR, avec un trou de taille fixe et d'emplacement variable : Trou de longueur 1 : Environ 800 millions de 16-mers espacés / étendus dans les LR 3 092 709 16-mers trouvés dans les SR 3 536 267 16-mers contigus dans les SR Trou de longueur 1 à 2 : Environ 1 million 500 mille 16-mers espacés / étendus dans les LR 3 461 162 16-mers trouvés dans les SR 3 536 267 16-mers contigus dans les SR => Tout les k-mers sont quasiment trouvés, mais impossible de déterminer les bons / mauvais en jouant sur les seuils de fréquence

Chapitre 5

Conclusion

Peu de bon resultats, a part kmersExpand, beaucoup de choses à revoir.

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*.

[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.