Mémoire de stage
TITRE DE L'INTITULE DE STAGE

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

25 Juin 2017

Table des matières

Chapitre 1

Introduction

1.1 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.

1.2 Contexte

L'analyse de séquençage ADN appliqué à la Bio-Informatique a connu un développement important dans les années 2000.
Le séquençage ADN est effectué de deux façons différentes, par le biais de lectures sur des brins d'ADN :

$\longrightarrow$ **Les lectures longues** (Nanopore, PacBio, ...)
$\longrightarrow$ **Les lectures courtes** (Illumina, Roche, ...)

Les **lectures longues** ont l'avantage de nécessiter que d'**une petite quantité** d'entre elles pour quadriller le génome d'un être vivant, mais le gros désavantage d'être **hautement imprécises** (15 à 30 % d'erreurs)

Les **lectures courtes**, en revanche, ont le net avantage d'être **très précises**, avec un taux d'erreur de moins de 1 %. Leur inconvénient principal est qu'il faut **de nombreuses lectures** courtes afin de reconstruire le génome, ce

qui est coûteux et peu pratique.

Dans les faits, le génome peut être reconstruit grâce à un nombre déraisonnable de lectures courtes, ou en corrigeant les lectures longues grâce aux lectures courtes. Plusieurs **méthodes de corrections** ont été développées à ce sujet [1].

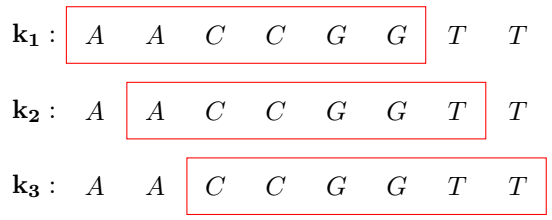
Afin de pouvoir se passer des lectures courtes, et d'ainsi corriger les lectures longues d'elles-mêmes, une branche de la bioinformatique s'intéresse à l'étude des **k-mers**.
PLUS DINFO SUR lE PROBLEME RENCONTRE ICI

Chapitre 2

Comptage des k-mers

2.1 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 :



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 meilleures 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.

2.2 K-mers espacés

Les **k-mers espacés** sont utilisés afin de simuler les 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.

2.2.1 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 :

$\mathbf{k_1}$: $A\ A\ C\ \not{G}\ G\ T\ T$

$\mathbf{k_2}$: $A\ A\ C\ C\ \not{G}\ G\ T\ T$

$\mathbf{k_3}$: $A\ A\ C\ C\ G\ \not{T}\ T$

5-mers : {AACGG, ACCGT, CCGTT}

2.2.2 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 $(\mathbf{L - k + 1 + t}) * \mathbf{4^t}$ k-mers espacés possibles, avec $\mathbf{t}$ = nombre de zéros dans le motif.
Ex : Avec une séquence $\mathbf{x = AACCGGTT}$ et le motif $\mathbf{m = 111011}$, on a les 6-mers espacés suivants :

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

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

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

$\mathbf{k_4}$: $A\ A\ C\ C\ G\ G\ \overbrace{A,C,G,T}\ T\ T$

6-mers : { AAC**A**CG, AAC**C**CG, AAC**G**CG, AAC**T**CG, ACC**A**GG, ACC**C**GG, ACC**G**GG, ACC**T**GG, CCG**A**GT, CCG**C**GT, CCG**G**GT, CCG**T**GT, CGG**A**TT, CGG**C**TT, CGG**G**TT, CGG**T**TT }

2.3 Etat de l'art

Parmi les différents logiciels qui permettent le comptage des k-mers simple, nous avons l'outil Jellyfish [2] qui est de très loin de plus efficace (500Mo en $<$ 10s avec 8 coeurs), 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 innéficace. Une equipe de chercheur de l'université de montpellier sont en train de developper un outil nommé gkampi permettant de faire de l'extraction de k-mers normaux ou espacés.
Cependant à l'heure de l'écriture de ce rapport, les developpements de l'outils sont encore au stade experimental, et souffre de performance en vitesse peu satisfaisante, même si elle profite d'une utilisation RAM très optimale.
C'est à partir de cet etat de fait que j'ai commencé à developper mon propre logiciel, kmerDel.

Chapitre 3

K-mers espacés à délétion : kmerDel

3.1 Programme

Algorithme 1 : kmerDel

```
   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 fenetre glissante afin de recolter tous les k-mers, puis on ajoute le kmer à la table s'il n'existe pas déjà. Afin de gagner en espace mémoire, le kmer est converti en base 4 avant d'être stocké.

La table de hachage a subit divers améliorations au fil du temps :

- Tout d'abord, c'était un type **map** de la librairie STL du C++, et le programme n'avait n'utilisait qu'un seul thread.
- Ensuite, avec le développement du multi-thread, le fichier de lectures ADN a été séparé en t ensemble de lignes, qui sont passés indépendament à une table de hachage par thread. Puis les tables étaient refusionnées ensemble. L'utilisation de t table était du au fait que les conteneurs de la STL ne sont pas *thread-safe*, c'est à dire que l'utilisation de threads sur les conteneurs de la STL donne lieu à un comportement non-défini.
- La table de hachage est devenu une **unordered_map** (c++11) qui a l'avantage d'être prêt de quatre fois plus rapide en temps d'execution que **map**, en contrepartie du fait qu'elle e trie plus par ordre croissant ses éléments.
- Enfin après diverses recherches, la table est devenu une **cuckoohash_map** qui est un type issu de la librairie indépendante libcuckoo(ref). Ce type 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 replacant les affichage du programme par un **buffer** qui stocke les lignes avant de les afficher.

3.2 Résultats

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 pour kmerDel. L'avantage de kmerDel est qu'il permet, contrairement à Jellyfish, de compter les k-mers espacés à délétions. En contrepartie, kmerDel utilise beaucoup plus de RAM que Jellyfish dès que l'on augmente la taille de k, ce qui le rend moins utilisable en pratique.

RESULTAT PIERRE EN TABLEAU

Chapitre 4

K-mers espacés à insertion : kmerExpand

4.1 Programme

Algorithme 2 : kmerExpand

Entrées : chaînes *lectures*, chaîne $motif$, 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 : kmerExpandRec

Entrées : chaîne $kmer$, chaîne $nvKmer$, chaîne $motif$, 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 kmerExpand 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 kmerDel ne pouvait pas fonctionner ici.

Etant donné qu'il y a quatre nucléotides possible par 0 du motif, on a $(L - k + 1 + t) * 4^t$ k-mers de taille k possible, ce qui développe de façon exponentielle les resultats. 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
En conséquence, l'algorithme est bien plus lent que kmerDel, cependant, la taille du fichier n'influe pas sur la quantité de RAM utilisée, ce qui en fait un outil utilisable en pratique.

4.2 Résultats

Resultats pierre plutot bons, identification du probleme toujours incertain

Chapitre 5

Autres structures de données recherchées

5.1 Mot Minimaux Absents (MMAs)

Explication wikipedia + pierre
Resultats pas au rdv

5.2 Plus long sous-mot commun (PLS)

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

Chapitre 6

Conclusion

Peu de bon resultats, a part kmerExpand, 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.