Kmer signature in mitochondrial genomes for metagenome taxonomic assignation

Jérémy Fontaine and Samuel Blanquart

19 août 2014

1 Semaine 1 : Mardi 1 Juillet

1.1 Réduction de la BDD

La base de donnée complète totalise **30Go**, l'objectif était de réduire la taille de cette BDD et n'ayant les données qu'aux feuilles, pour les autres noeuds de l'arbre les données sont sous forme de lien symbolique. On obtient au final une BDD avec une taille de **1,1GO**.

Pour cela :

- Modification du fichier chemin-liste d'accession (voir figure 1)
- Script de remplissage des noeuds internes

tax1/tax2/.../taxonFeuilleX : accession_1,...,accession_N
tax1/tax2/.../taxonFeuilleY : accession_1,...,accession_M

FIGURE 1 – Exemple de fichier "chemin : liste d'accession"

L'idée pour générer les liens symboliques est de faire remonter chaque données feuilles vers la racine de la BDD (figure 2).

```
      //
Pour chaque feuille l de la BBD
{
    /* On se deplace au niveau de la feuille */
    cd l;
    Pour chaque fichier f du dossier courant;
    {
      /* on se deplace dans la parent de cette feuille */
      courant = pwd // commande unix
      tant que courant != racine
      {
        ln -s X
        /* on remonte d'un niveau */
        cd .. ;
        courant = pwd
      }
      /* on se replace a la feuille pour traiter le nouveau fichier */
      cd l;
    }
}
```

FIGURE 2 – Algo lien symbolique

1.2 Krona

krona est un utilitaire qui permet de visualiser des hiérarchies sous forme de camembert "zoomables". Pour cela cette hiérarchies doit être écrites sous formes de fichier xml. Krona construit ensuite un fichier html local, un navigateur web permet alors de visualiser le camembert. (figure 3)

```
<node name="Alveolata">
    <genomes><val>1009</val></genomes>

    <node name="Apicomplexa">
        <genomes><val>993</val></genomes>
        ...
    </node>
</node>
```

FIGURE 3 – Exemple simplifié d'un fichier pour krona

1.3 Scripts dmp et krona

En fin de semaine, deux scripts ont été développés pour :

- Récupérer/mettre à jour les fichiers dmp pour la construction de la bdd.
- Récupérer et installer krona

2 Semaine 2 : Lundi 7 Juillet

2.1 Générer le fichier weka à partir d'un dossier

L'idée première était de générer le fichier de comptage au format weka à partir d'un dossier. Pour se faire pour un dossier D il faut effectuer le comptage pour ses sous dossier $d_1, ..., d_n$ en prenant soins de récupérer chaque taxid pour chaque d_i afin de lancer le programme de comptage *count_ kmer* avec les bons arguments. Le programme a était réalisé en Perl.
(trunk/generate_learn/generate_learn.pl)

2.1.1 Parallélisme

Au lieu de lancer le programme réalisé auparavant il est possible à partir des options d'utiliser plusieurs processeurs afin d'effectuer plusieurs comptage. Cependant même si le programme offre cette fonctionnalité lors de l'appel system de Perl pour appeler le programme C de comptage, Perl rend immédiatement la main et lance l'appel système suivant, le parallélisme se fait alors par défaut.

2.2 Générer les fréquences de kmer aux feuilles

Le but de cette partie est de générer les fréquences uniquement aux feuilles. Ainsi pour construire le fichier weka à un noeud donnée d il suffit d'aller récupérer les fréquences aux feuilles du sous arbre ayant pour pour racine d. (trunk/generate_learn/generate_count.pl)

3 Semaine 3 : Mardi 15 Juillet

3.1 Gestion d'erreur en C

Une partie de cette semaine a été consacrée au débogage du comptage écrit en C. En effet le comptage à la racine avec une fenêtre de 50, provoquait des erreurs d'allocation mémoire. Ce problème fut régler grâce à la re-allocation de la mémoire.

3.2 Taille du jeu de fréquences à la racine

Avec une taille de fenêtre glissante constante, la taille du fichier de fréquence est proportionnel à la somme des tailles de séquences (nombre de nucléotides totale). On a pu évaluer ainsi la taille du fichier de fréquence au plus haut niveau avec une fenêtre de taille 50 et le pattern ####. On obtiendrai au niveau de la racine, une taille de **123Go** pour le fichier de fréquence et un temps de **120 min**. Ainsi grâce aux graphes ci dessous on a pu évaluer le temps (fig. 4) pour générer le fichier de fréquence et la taille de celui-ci (fig. 5) sur l'ensemble des séquences (251 371 097 nucléotides).

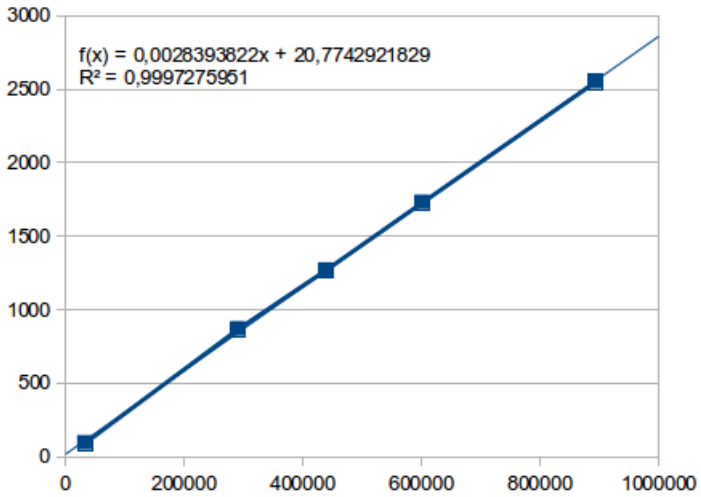

FIGURE 4 – Temps en seconde en fonction de la taille de la séquence avec une fenêtre de taille 50

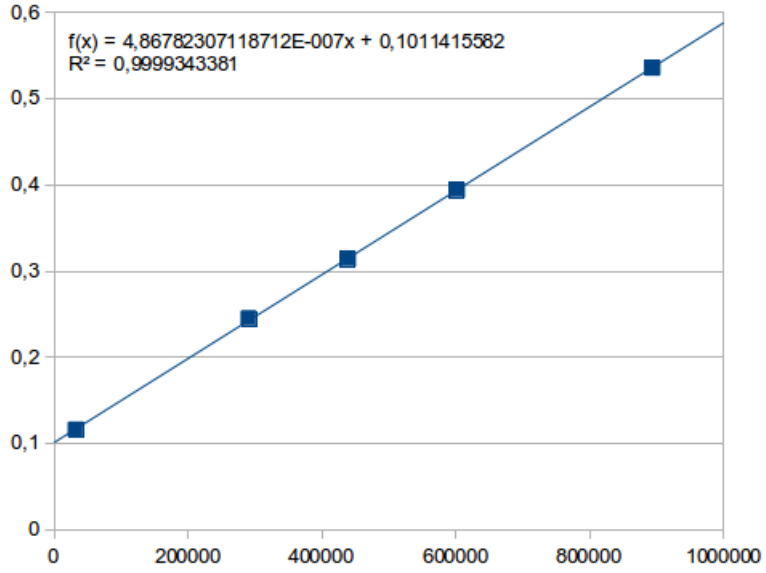

FIGURE 5 – Temps en seconde en fonction de la taille de la séquence avec une fenêtre de taille 50

3.3 Passage aux objets

Pour la suite du projet on décide de passer au c++ afin de tout gérer en tant qu'objet, on pourra ainsi plus aisément manipuler la table de fréquence en kmers. La génération de données pour weka, la validation croisée (avec les cartes...) ...

3.3.1 Les classes

Pour le moment on ne manipule que trois classes :

- classData : pour gérer les données (séquences,taille, nombre de jeux de données, responsable du comptage)
- classPattern : gestion d'un kmer
- FreqKmer : classe principale gérant principalement le tableau de fréquence.

L'objet à la possibilité de s'instancier à partir

. D'un fichier fasta

. D'un fichier contenant une liste de chemins vers des fichiers fasta.

De ce fait lorsqu'on souhaite obtenir la fréquence à un niveau, c'est au niveau des feuilles du sous arbre définit par ce niveau où l'on va effectuer le comptage.

En fin de semaine des tests de validation ont été rajoutés au programme. Le programme est fonctionnel et les tests le confirment.

4 Semaine 4 : Lundi 21 Juillet

4.1 Nouvelle façon de compter

Suite à la réunion du 18 Juillet, on a décide de ne plus compter avec une fenêtre glissante. En effet cette façon générer des redondances dans la matrice de fréquences en kmers. Pour cela l'utilisateur à le choix de fournir son propre décalage (en nucléotide) sinon le programme utilise un décalage égale à 20% de la taille de la fenêtre.

4.2 Optimisation

Dans un premier temps le programme à été revu pour prendre en compte ce décalage mais sans optimisation. L'optimisation consisterais à ne compter que les nouveau kmers lors du décalage de la fenêtre. Cette astuce était utilisée dans la version précédente. Mais étant que le décalage était égale à 1 nucléotides, une variable était suffisante pour mémoriser le kmer qui "sortait" de la nouvelle fenêtre comparé à la fenêtre précédente. On copiait ainsi la ligne de fréquence de la fenêtre précédente, on décrémenter la fréquence du kmer qui "sortait" et on calculait le nouveau kmer qui "entrait".

Avec un décalage égale à n, on a n kmer qui "sort" et n nouveaux kmer à compter (entrant). Pour se faire on se propose d'utiliser deux tampons :

. *courant* : correspondant à la ligne de comptage actuel, pour une fenêtre donnée.
. *precedent* : correspondant à la ligne de comptage pour la fenêtre précédente.

Ces tampons ont la même taille que le nombre de kmer présent dans la fenêtre, soit $f - k + 1$ si f est la taille de la fenêtre et k la taille du kmer. Ainsi lors qu'on compte une toute première fois cela permet d'initialiser le tampon courant et de mémoriser tous les kmers rencontrés. Lorsqu'on passe à la fenêtre suivante si on effectue un décalage de d nucléotides, il suffit donc :

. D'intervertir les deux tampons
. De copier les bonnes parties dans le nouveau tampon, courant
. Calculer les nouveaux kmers entrant et sauvegarder dans le tampon courant.

On obtient donc les kmers rencontrés, et on augmente dans le tableau de fréquence par rapport au tampon courant. Voir figure 6 pour illustration, avec un kmer=###, une fenêtre de taille 7 et un décalage $d = 2$.

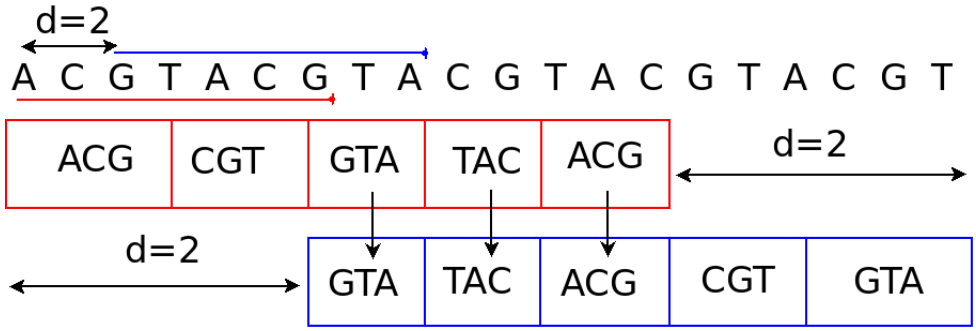

FIGURE 6 – Principe du décalage avec une fenêtre de taille 7 et un tri-mers

Même si les tests sont ok et les tableaux de fréquences corrects. Il y a des cas où valgrind détecte des erreurs au niveau des ces buffers. D'autre erreurs par ailleurs sont également détecté par valgrind. L'objectif principal avant de continuer et d'éliminer toutes ces erreurs afin de continuer sur un programme "propre".

5 Semaine 5 : Lundi 28 Juillet

5.1 Débogage

Au fil de l'implémentation , des erreurs d'accès mémoire se sont accumulés. Ce fut une tâche longue de trouver toutes les sources d'erreur. A présent au fil du développement avant de passer une étape, valgrind et les tests doivent le programme.

5.2 Les cartes du tableau fréquences

5.2.1 D'une ligne à son jeu de donné, sa séquence

Les semaines précédentes une carte a été mise en place, pour obtenir à partir d'une séquence d'un jeu de donnée l'indice de la ligne correspondante dans la table des fréquences. Il a été jugé utile également d'avoir la carte inverse permettant à partir d'une ligne de retrouvé la séquence et le jeu de donnée associé. Cependant ces cartes sont insuffisantes ce qui est réellement important c'est de savoir pour une ligne à quelle taxon correspond la fréquence. La carte a donc été enrichie avec en amont le taxon associé par exemple avec la structure en figure 7 :

```
+taxon_alpha
|
|___+others
|   |__genomes.fasta (3 seqs)
|
|___+taxon__A
|   |__genomes1.fasta (3 seqs)
|   |__genomes2.fasta (1 seqs)
|   |__genomes3.fasta (3 seqs)
|   |__genomes4.fasta (2 seqs)
|
|___+taxon__B
|   |__genomes1.fasta (7 seqs)
|
|___+taxon__C
|   |__genomes1.fasta (4 seqs)
|   |__genomes2.fasta (1 seqs)
|
```

FIGURE 7 – Exemple de structure de l'arborescence au niveau du taxon alpha

Avec la structure donnée en figure 7 la carte associé serait celle présentée en figure 8.

Taxon	Others			A								B							C				
Data	0			1			2	3				4		5							6			7
SeqInData	0	1	2	0	1	2	0	0	1	2	0	1	0	1	2	3	4	5	6	0	1	2	3	0

FIGURE 8 – Map dans le cas de la structure donnée en figure 7, avec comme racine taxon__alpha

Il est possible à présent, à partir d'une ligne, de savoir à quelle séquence de quel jeu de données et surtout à quel taxon correspond la ligne de fréquence. C'est un bon point pour plus tard dans le développement des outils, en particulier pour l'écriture du fichier d'apprentissage puisque le taxid d'un taxon pour une ligne de fréquence donnée est connu.

Lors du test du programme sur une autre machine, il y a eu des erreurs avec les tests 15 à 18. Ceci est dû fait que lors de la lecture des sous dossier par l'appel système readdir, c++ ne garantie pas d'ordre. Un trie est alors effectue sur les chemins. C'est une étape qui permet au bon déroulement des tests 15 à 18 sur n'importe quel machine. Cependant il est possible de se passer du trie, dans ce cas le tableau de fréquence sera dans un ordre différent mais avec les cartes cela ne pose pas de soucis.

6 Semaine 6 : Lundi 04 Août

6.1 Échantillonnage

L'objectif de cette semaine est de pouvoir échantillonner les données. C'est primordial pour la suite puisque comme nous l'avons vu à la figure 5 la taille du fichier de fréquence peut atteindre plus de 120Go, une taille trop importante pour l'outil d'apprentissage *Weka*.

6.1.1 Principe d'échantillonnage

Pour échantillonner les données à un niveau taxonomique *alpha*, on se propose de considérer SZ (Sample Size) séquences tirées aléatoirement pour chaque sous taxon de *alpha*.

Si on reprend l'exemple de la figure 7 avec $SZ = 5$, on a par exemple suite à un tirage aléatoire sans remise des séquences :

```
+taxon_alpha
|
|___+others
|   |__genomes.fasta (3 seqs)
|
|___+taxon__A
|   |__genomes1.fasta (2 seqs)
|   |__genomes3.fasta (1 seqs)
|   |__genomes4.fasta (2 seqs)
|
|___+taxon__B
|   |__genomes1.fasta (5 seqs)
|
|___+taxon__C
|   |__genomes1.fasta (4 seqs)
|   |__genomes2.fasta (1 seqs)
|
```

FIGURE 9 – Exemple de structure de l'arborescence au niveau du taxon alpha après l'échantillonnage pour $SZ = 5$

6.1.2 Méthode d'échantillonnage

Pour échantillonner les données on va enrichir notre objet $FreqKmer$ par l'ajout d'un tableau de booléens à deux dimensions pour savoir quelles séquences seront considérées dans le comptage de kmers.

```
29
30 /* Pour chaque sous taxon */
31 for(int i=0;i<nbChildTaxa;i++)
32 {
33     nbSeqTaxa = getNSeqInTaxa(i);
34
35     /* si on doit tirer plus de qu'il y a de seq alors on tire tout */
36     if(sampleSize>=nbSeqTaxa)
37     {
38       d = obtainStartLineTaxaInFastaList(i); /* indice debut data pour ↩
           taxon courant i */
39       f = obtainEndLineTaxaInFastaList(i); /* indice fin data pour taxon ↩
           courant i */
40
41       /* Pour chaque data du taxon courant */
42       for(int j=d; j<=f;j++)
43       {
44          /* Pour chaque sequence */
45         for(int k=0;k<data[j]->getNtaxa();k++)
46         {
47           /* On prend la k-eme sequence du j-eme data */
48           mask_tmp[j][k]=true;
49         }
50       }
51     }
52     else
53     {
54         /* on tire sampleSize parmi les nbSeqTaxa sequences
55        * dans le taxon courant
56        * (tirage sans remise pour les seq.!)
57        */
58       randomTab(&candidates,nbSeqTaxa,sampleSize);
59       /* je mets a jour mon mask selon les candidats trouves */
60       maskTab(&candidates,mask_tmp,i);
61     }
```

FIGURE 10 – Algo d'échantillonnage

```
63
64 /* tirage sans remise de sampleSize sequences
65 parmi tabSize sequences */
66 void randomTab(vector<int> *result,int tabSize,int sampleSize)
67 {
68
69   int *tmp = new int[tabSize];
70   int r = -1;
71
72   /* Borne du tirage au depart on peut tirer toutes les sequences */
73   int sup = tabSize;
74   int val_tmp;
75
76   /* on cree un tableau contenant
77    * initialement les indices, on va ainsi tirer les indices des
78    * sequences a considerer pour l'echantillon */
79   for(int i=0;i<tabSize;i++)
80   {
81     tmp[i]=i;
82   }
83
84   /* On tire sampleSize indice */
85   for(int j=0;j<sampleSize;j++)
86   {
87
88     r = rand() % sup;
89     /* on met la valeur tiree a la fin du tableau
90      * et on n'y touche plus */
91     val_tmp = tmp[sup-1];
92
93     /* on met l'indice dans le resultat */
94     result->push_back(tmp[r]);
95     tmp[sup-1] = tmp[r];
96     tmp[r] = val_tmp;
97
98     /* la borne du tirage diminue */
99     sup--;
100
101  }
102
103 }
```

FIGURE 11 – Algo de tirage sans remise

7 Semaine 7 : Lundi 11 Août

L'objectif de cette semaine est de pouvoir générer un jeu de données pour la validation croisée sous weka.

7.1 Classer les données

Avant de poursuivre, la base de donnée à été modifiée afin des données classée. Ceci permet de mieux se retrouver entre séquences d'acides aminées, nucléotidiques, fichier genbank, les enzymes (cox1,cox2...) ...

Chaque dossier comporte en plus des dossiers des sous taxons :

. Un dossier *frequencies* où sera rangées les fréquences de kmers

. un dossier *data* correspondant aux données génétiques et étant constitué

– Un dossier *fasta* où sera rangées les séquences, constitué de deux sous dossiers

* *aminoAcids* contenant les séquences d'acides aminés, chaque cox, cytb,...sont également rangé dans un dossier de même nom.

* *nucletotides* contenant les séquences nucléotidiques, de même ici chaque cox, cytb,...sont également rangé dans un dossier de même nom. Ce dossier contient en plus un dossier *genomes* contenant tous le génome complet (voir figure 12).

– un dossier *genbank* contenant les fichiers genbank associés au taxon courant

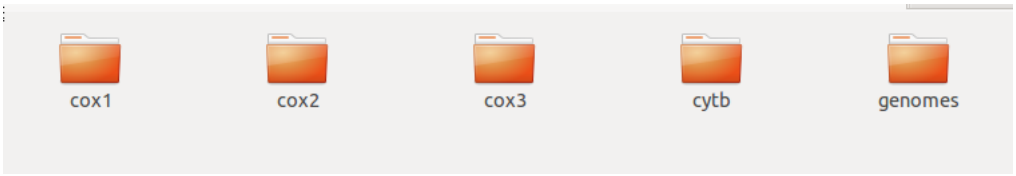

FIGURE 12 – Dossiers lorsqu'on se trouve dans le dossier data/fasta/nucleotides d'un taxon.

Cette façon de classer les données permet à l'utilisateur de pouvoir spécifier sur quelles séquences il souhaites travailler en indiquant les mots clés *cox1*,*cox2*,...ou *genomes*.

8 Semaine 7 : Lundi 18 Août

9 Semaine 8 : Lundi 25 Août