

Kmer signature in mitochondrial genomes for metagenome taxonomic assignation

Jérmy Fontaine and Samuel Blanquart

28 juillet 2014

1 Semaine 1 : Mardi 2 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'accension (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'accension"

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

```

1      //
2  Pour chaque feuille l de la BBD
3  {
4      /* On se deplace au niveau de la feuille */
5      cd l;
6      Pour chaque fichier f du dossier courant;
7      {
8          /* on se deplace dans la parent de cette feuille */
9          courant = pwd // commande unix
10         tant que courant != racine
11         {
12             ln -s X
13             /* on remonte d'un niveau */
14             cd .. ;
15             courant = pwd
16         }
17         /* on se replace a la feuille pour traiter le nouveau fichier */
18         cd l;
19     }
20 }

```

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)

```

21 <node name="Alveolata">
22     <genomes><val>1009</val></genomes>
23
24     <node name="Apicomplexa">
25         <genomes><val>993</val></genomes>
26         ...
27     </node>
28 </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, \dots, 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 été 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 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 réglé 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 obtiendra 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).

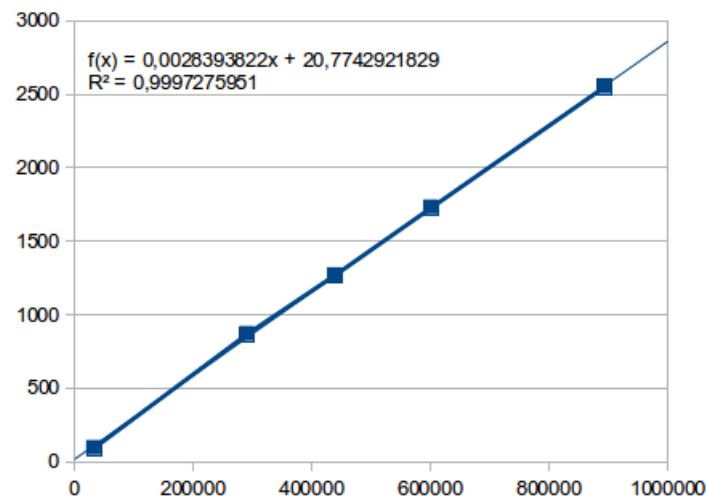


FIGURE 4 – Temps en seconde en fonction de la taille de la séquence avec une fenêtre de taille 50

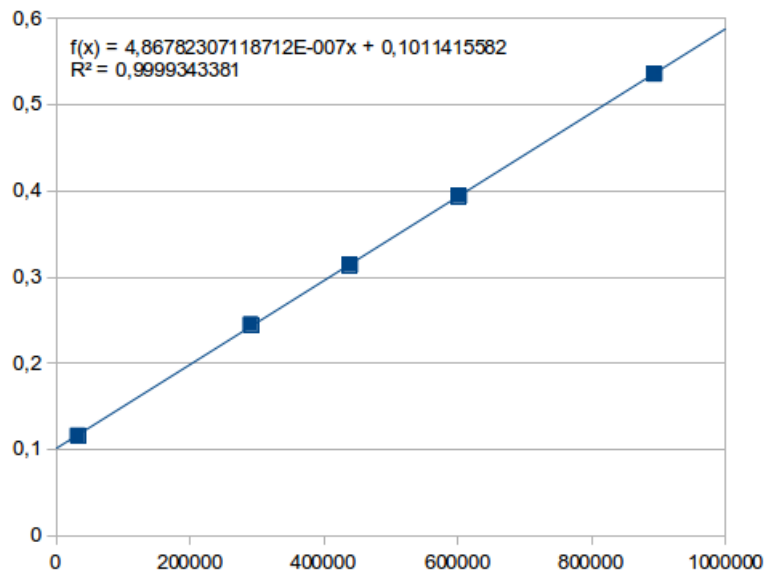


FIGURE 5 – Temps en seconde en fonction de la taille de la séquence avec une fenêtre de taille 50

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...

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 rajouté au programme.