Bioinformatique

Rapport Développeur – Thomas Esseul – IL1.1

Introduction

Le projet consistait à développer un programme permettant la compression et décompression de fichiers fasta. Dans ce but, il a fallu développer premièrement une structure de données en suivant les indications fournies, et secondement les fonctions de traitement des fichiers à compresser et à décompresser.

Structure du projet

Le programme se compose de 6 fichiers :

- **Compressor.py** : Le fichier nécessaire à la compression, utilisable par ligne de commande
- **Decompressor.py** : Le fichier nécessaire à la décompression, utilisable par ligne de commande
- **Reader.py** : Le fichier contenant la fonction de lecture d'un fichier fasta avec remplissage d'un filtre de Bloom
- **BloomFilter.py** : Définition de la structure de données du filtre de Bloom
- **BitArray.py** : Définition de la structure de données du tableau de bits
- **utils.py** : Fonctions « utilitaires » (calcul des voisins, de la forme canonique, etc..)

Plusieurs fichiers dépendent d'autres fichiers du module :

BloomFilter utilise BitArray comme structure de base.

Reader, Compressor et Decompressor utilisent tous les trois le BloomFilter pour stocker et interroger les données. Ceux-ci utilisent également les fonctions utilitaires.

Points clés de l'implémentation

Reader

Le reader est résolument simple, il se contente de parcourir le fichier fasta d'entrée (en ne regardant que les lignes contenant un read de génome), récupère chaque kmer de la taille indiquée en paramètre de l'appel (par défaut 20) et en stocke la forme canonique dans le filtre de Bloom. Il finit par renvoyer le Filtre de Bloom remplit pour permettre son utilisation par les autres fonctions.

Compressor

Le compresseur, requérant en entrée le fichier fasta et le filtre de Bloom rempli, fonctionne sur un principe similaire, en plus de sérialiser et compresser le filtre de Bloom. Il parcourt les reads du fichier fasta, inscrit le premier kmer de celui-ci sur le fichier compressé et commence le traitement du reste du read.

Partant du premier jusqu'au dernier kmer de la ligne, le traitement se base sur la recherche "d'ambiguïté", c'est à dire une situation où le kmer traité aurait plusieurs voisins possibles dans le filtre de Bloom (pour être exact, on recherche la forme canonique des voisins dans le filtre).

Pour détecter ces ambiguïtés, on calcule les voisins possibles et les parcourt à la recherche d'un voisin présent dans le filtre qui serait différent du kmer écrit dans le read, si au moins un de ceux-ci est trouvé, la situation est considérée comme ambigüe. Cela est symptôme d'une ambiguïté à venir au moment de la décompression, pour parer à cela, on inscrit le caractère correspondant au kmer présent à la suite du fichier traité, dans ce qu'on appelle les "indications d'embranchements", et passe au kmer suivant.

Decompressor

Le décompresseur commence par restaurer le filtre de Bloom sérialisé et compressé par le compresseur, il parcourt ensuite le fichier compressé ligne par ligne. Le kmer écrit en début de ligne est récupéré puis commence la restauration du read. Comme pour le compresseur, on utilise le filtre de Bloom pour chercher les voisins existants (ou plutôt leur forme canonique).

Dans la recherche de voisin on appelle "candidat" un voisin présent dans le filtre de Bloom. Ainsi, une fois les potentiels voisins évalués, si un seul candidat est trouvé, il s'agit de la suite du read en cours de restauration, on peut alors l'inscrire dans le fichier décompressé. Si plusieurs candidats existent, cela signifie qu'on est face à une situation "ambigüe" du compresseur, il faut alors se référer aux indications d'embranchement inscrite dans le fichier compressé, à la suite du premier kmer.

Dans les deux cas, on a restauré un caractère de plus dans la chaîne et on peut alors recommencer l'évaluation des voisins du kmer nouvellement reconstitué.

Bibliothèques utilisées

pickle

Lors de la compression d'un fichier, il est également nécessaire de conserver le filtre de Bloom contenant les données des reads du fichier originel, pour ce faire, la bibliothèque pickle a été utilisée pour sérialiser le filtre de Bloom dans un fichier et permettre de le restaurer par la suite dans la fonction de décompression.

gzip

Pour gagner en espace mémoire, la bibliothèque de compression gzip a été utilisée pour compresser (et décompresser) le fichier contenant le filtre de Bloom sérialisé.

argparse

Dans le but de gérer l'appel des fichiers et de leurs fonctions en prenant en compte des paramètres (par exemple la taille de kmer utilisée, les noms des fichiers d'entrée et sortie, etc ..) la bibliothèque argparse s'est avérée être un prompt renfort. Elle permet une gestion simple des paramètres positionnels et optionnels et génère automatiquement une aide à l'utilisateur en cas d'erreur de saisie, ce qui a été son rôle dans ce cas.

re

La bibliothèque re (pour « regular expression ») a été utilisée pour retrouver simplement le premier kmer du fichier compressé, quand on procède à la décompression de celui-ci. Re a également servie pour sécuriser les paramètres d'appels des fonctions de compression et de décompression, permettant de faire fi des éventuelles extensions données, assurant ainsi l'utilisation de fichier correctement formatés.

Par exemple si la fonction de compression était appelée avec le paramètre « 15xreads_istic_bif.fost » comme fichier d'entrée, le programme chercherait un fichier nommé « 15xreads_istic_bif.fasta ».