Projet ALG 2020

Mappeur sans gap, détection de SNPs

Clara Delahaye, Claire Lemaitre, Pierre Peterlongo

Généralités

Ce projet est à effectuer en binômes.

Il devra être rendu au plus tard le **Jeudi 14 janvier 2021**, par mail aux l'adresses : claire.lemaitre@inria.fr et pierre.peterlongo@inria.fr avec comme sujet [UE alg]Projet suivi de vos deux noms de familles.

Pitch

Vous allez implémenter un mappeur-SNPcaller de reads sur génome de référence. Durant le mapping, des substitutions seront autorisées (pas d'insertion ou délétion). Ce mapping permettra d'identifier les SNPs contenus dans les reads et de les fournir dans un fichier de type *vcf*.

Pour un read donné, son alignement ne sera testé que sur les régions du génome qui partagent au moins un k-mer avec ce read (ancrage).

Le génome sera indexé sous forme d'un FM-Index.

Précisions

Indexation

Un premier programme `index.py` permettra d'indexer un génome de référence, fourni sous la forme d'un fichier fasta. Pour simplifier on ne considère que des génome représentés sous la forme d'une unique séquence.

Ce premier programme prendra en entrée les arguments suivants:

```
python index.py --ref [genome_file.fa] --out [dumped_index.dp]
```

Pour le génome pris en entrée, il créera un FM-Index (FMI): transformée de Burrows-Wheeler + structures données additionnelles pour effectuer le pattern matching. Il stockera ce FMI dans le fichier `dumped_index.dp` (cf fin de projet pour un exemple d'écrire / lecture sur disque d'un tel fichier). Vous pourrez utiliser le programme `tools_karkkainen_sanders.py` proposé en TP pour construire le tableau des suffixes.

Mapping

Un second programme `map.py` permettra de mapper un ensemble reads, fournis sous la forme d'un fichier fasta décompressé (eg. `reads.fa`) sur un génome de référence (eg. `genome_file.fa`), et de son FMI (eg. `dumped_index.dp`) précédemment calculée via `index.py`.

Ce programme prendra en entrée les arguments suivants:

```
python map.py --ref [genome_file.fa] --index [dumped_index.dp] --reads
[reads.fa] -k [k_value] --max_hamming [h_value] --min_abundance [m_value] --out
snps.vcf
```

Pour chaque read r : les potentielles positions de mapping de r sur le génome de référence seront détectées via l'ancrage de k-mers (k étant un paramètre de ce programme). On fera attention à ne pas tester plusieurs fois la même position de mapping de r sur le génome.

Si le mapping de r échoue (trop de substitutions, le nombre maximal de substitutions étant également un paramètre du programme `--max_hamming`) rien n'est stocké.

En cas de multi-mapping (plusieurs positions possibles) on conservera la position montrant le moins de substitutions. S'il en existe plusieurs, on conservera la position de mapping la plus *à gauche* sur le génome.

S'il réussit,

- **une sortie intermédiaire** consistera à simplement afficher l'identification de ce read avec sa meilleure position de mapping et les substitutions détectées lors du mapping.
- une fois cette sortie intermédiaire validée (entre vous et avec nous): vous pourrez implémenter la partie SNP calling, en stockant les positions de substitution.

Fichier VCF

Une fois tous les reads mappés, vous avez à votre disposition un ensemble de positions du génomes mutées, avec les abondances des allèles alternatifs. Ainsi vous avez à votre disposition toute l'information nécessaire pour produire un fichier VCF simplifié qui suivra le format suivant:

```
POS REF ALT ABUNDANCE
```

Par exemple la ligne

```
129836  A T 26
```

Indique qu'à la position `129836`, le génome est constitué d'un `A` mais que 26 reads ont mappé à cette position avec un `T`. Notez qu'on ne demande pas d'afficher l'abondance de l'allèle référence.

Vous n'indiquerez que les variants dont l'abondance est supérieure ou égale à `m_value` définie par l'option `-min_abundance`.

Les lignes seront ordonnées par positions sur le génome. La position est `0-based`, c'est-à-dire que la première lettre du génome est considérée comme la position 0.

Votre fichier débutera avec les lignes suivantes:

```
#REF: nom de fichier contentant le génome de référence
#READS: nom de fichier contentant les reads
#K: valeur de k utilisée pour effectuer l'ancrage
#MAX_SUBST: la variable h_value
#MIN_ABUNDANCE: la variable m_value
```

Voici un exemple d'un tel fichier, `result.vcf`:

```
#REF: reference.fa
#READS: my_reads.fa
#K: 14
#MAX_SUBST: 5
#MIN_ABUNDANCE: 10
129836  A T 26
145831  C G 25
```

Bonus

Une fois le reste du projet effectué, vous pourrez en bonus proposer une ou plusieurs approches permettant de limiter les temps de calculs de votre mappeur.

Si vous effectuez ce bonus, nous attendons de vous que vous étudiez l'impact de vos propositions sur les temps de calculs mais aussi sur la qualité des résultats obtenus. De même il faudra décrire dans les rapports ces propositions et leurs impacts.

Données de test

3 jeux de données vous seront fournis (sur Moodle) sur lesquels vous pourrez tester et appliquer votre outil :

- smallMappingTest : tout petit jeu de données syntéhtiques pour tester la partie mapping
- coli : jeu de données synthétiques pour tester les performances de votre outil ainsi que l'impact des paramètres
- covid : vraies données d'un prélèvement pulmonaire d'un patient covid.

Rapports

Rapport développeur

Ce rapport explicite la structure du programme (diagrammes bienvenus) et précise le fonctionnement des fonctions clefs.

Il doit être court (2 à 3 pages).

Rapport utilisateur

Il se compose de deux parties distinctes:

1. Une partie permettra d'indiquer en quelques paragraphes les principes de base de votre assembleur, et présentera les commandes à utiliser (à considérer comme un fichier *README*).
2. Une partie présentera les résultats obtenus :
   - Temps de calculs
   - Mémoire utilisée
   - qualité des résultats

   en **fonction des valeurs des paramètres**.

   Pour évaluer la qualité des résultats, nous vous fournirons un *vcf* de référence et un programme qui permettra de connaitre le nombre de SNPs Faux Positifs et le nombre de SNPs Faux Négatifs de votre résultat.

   Cette partie inclura une discussion et des conclusions quant aux qualités et défauts de l'approche proposée, et guidera l'utilisateur sur le choix des paramètres.
3. Enfin, un paragraphe portera sur l'anayse des données *covid* (voir section données de test) avec votre outil.

Précisions

- Votre programme ne sera pas interactif. Une fois lancé avec ses arguments, il termine sans intervention de l'utilisateur.
- Respectez scrupuleusement les noms des programmes et options précisées dans ce document.
- Votre code sera largement commenté et les noms de classes, de variables et de fonctions devront avoir un sens.
- Le code sera en python3 et respectera la norme PEP8 (utilisation de `pylint` bienvenue)
- Les bibliothèques extérieures de type BioPython sont interdites.
- L'utilisation d'un package type `getopt` pour gérer les options du programme est vivement conseillée.

Notation

La notation de vos projets (rendus à temps et respectant le format du mail de rendu) prendra en compte les aspects suivants :

- Choix algorithmiques et succès de l'implémentation,
- Organisation, lisibilité et commentaires du code,
- Qualité des rapports et des analyses effectuées.

Ne sous-estimez pas l'importance des rapports et la forme du code. Un projet "qui marche" n'est pas nécessairement synonyme de bonne note.

Sérialisation

En python il est facile d'écrire le contenu d'un objet entier dans un fichier :

```
import pickle

O = MY_OBJECT() # Exemple my_fmi = Fmi("reference.fa") : création du FMI de la séquence contenue dans `reference.fa`
pickle.dump(O, open("myobjectO.dumped","wb")) # Exemple pickle.dump(my_fmi, open("dumped_index.dp","wb"))

#Plus tard (lors d'un autre appel du programme ou d'un autre programme, vous pouvez lire 'O' à partir du fichier myobjectO.dumped :
O==pickle.load(open("myobjectO.dumped","rb")) # Exemple pickle.load(my_fmi, open("dumped_index.dp","rb"))
```