SORBONNE UNIVERSITÉ

Explication du code

PLAN

Division des séquences en [kmer, indice]

Comparaison des [kmer, indice]

Etude sur les fichiers

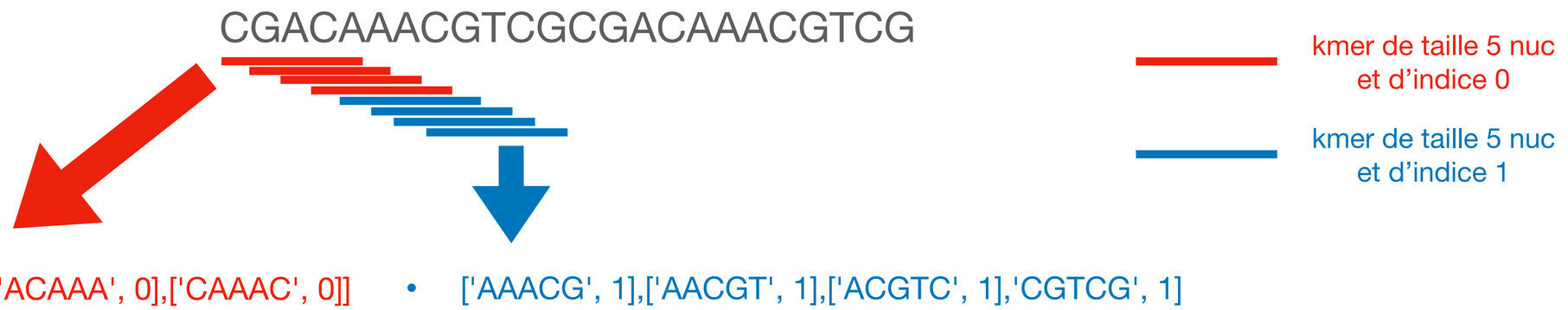
Division des séquences en [kmer, indice]

def kmers_split(seq, k=5, taille_V = 280, nb_indice = 70):

- Grâce à une fonction test, on a pu déduire que 4 kmers de taille 5 avec le même indice donne de bons résultats. Exemple d'une liste de [kmer, indice] :[[‘CGACA’, 0],['GACAA', 0],['ACAAA', 0],['CAAAC', 0],['AAACG', 1],['AACGT', 1],['ACGTC', 1],'CGTCG', 1]].

- Le but des indices est de ne pas comparer un kmer au debut de le sequence ref avec un kmer en dernier dans la sequence query. C'est inutile et rend le résultat faux. Exemple: On veut pas comparer un [‘CGACA’, 0] dans la seq ref avec un [‘CGACA’, 27] dans la seq query.

- Exemple sur une sequence:



[[‘CGACA', 0],[‘GACAA', 0],['ACAAA', 0],['CAAAC', 0]]

- ['AAACG', 1],['AACGT', 1],['ACGTC', 1],'CGTCG', 1]

Comparaison des listes de kmer de la forme [[kmer,indice]]

def comparaison(query_kmer, ref_kmer, bloc_max = 40):

- Dans cette fonction, on parcourt toute la liste kmer reference et on voit si il y a le même [kmer, indice] dans la liste kmer query. S'il y a, on incrémente de 1 le count, sinon on passe.

- Afin d'éviter des comparaisons inutiles, on a mis une variable bloc_max. Cette variable a pour but d'arrêter l'algorithme dans le cas où le count n'a pas été incrementé pendant bloc_max comparaison, c-a-d, qu'on a pas pu trouver un [kmer, indice] identique dans les deux listes lors de bloc_max iteration. On a utilisé une fonction test() afin de trouver le bloc_max le plus optimal.

Etude sur nos fichiers

def main_code(nb_indice = 60, bloc_max = 45, i =0, while_i = 150):

- Etape 1: On ouvre le fichier des sequences ref grace à la fonction read_db(seq) deja prédéfinis.
- Etap 2: On ouvre notre fichier query.
- Etape 3: on divise notre seq query en [kmer, indice] en précisant la taille du kmer = 5 et le max de la division qui est de 280 nuc.

```
q_kmer = kmers_split(query, 5, 280, nb_indice)
```

Etude sur nos fichiers

def main_code(nb_indice = 60, bloc_max = 45, i =0, while_i = 150):

- Etape 4: On parcourt tous les keys du dictionnaire du fichier ref. Exemple: on prend les keys = [IGV1-18*01, IGV1-18*02, IGV1-18*03]

```
>IGHV1-18*01
caggttcagctggtgcagtctgg
tgggtgcgacaggcccctggaca
accacagacacatccacgagcac
>IGHV1-18*02
caggttcagctggtgcagtctgg
tgggtgcgacaggcccctggaca
accacagacacatccacgagcac
>IGHV1-18*03
caggttcagctggtgcagtctgg
tgggtgcgacaggcccctggaca
accacagacacatccacgagcac
```

- Etape 5: On fait une boucle for afin de parcourir toutes les seq et de comparer leurs [kmers, indice] avec les [kmers, indice] de la seq query deja fait

```
# on commence l'algorithme
for j in range(len(dic_db_keys)):
    r_kmer = kmers_split(dic_db_values[j], 5, 280, nb_indice) #
    comp = comparaison(r_kmer, q_kmer, bloc_max ) # on realise
    dic[dic_db_keys[j]] = comp # on ajoute la valeur du resulta
                               # ayant comme key le kmer de la
```

- On place tous nos résultat dans un nouveau dictionnaire qui prend comme valeur le nom de la seq ref ainsi que son score. Exemple: dic = {IGV1-18*01: 128, IGV1-18*02:50, IGV1-18*03:110}

Etude sur nos fichiers

def main_code(nb_indice = 60, bloc_max = 45, i =0, while_i = 150):

- Etape 6: on prend le score max du dictionnaire et on l'ajoute dans une liste

- Etape 7:

-> si la liste est de taille 1, c-a-d, qu'il y a seulement une et une seule seq trouvée, alors on renvoie cette seq.

-> si la liste est de taille > 1, c-a-d, que l'algorithme a trouvé plusieurs seq de même score, on essaie de faire une etude supplémentaire sur ces seq afin de trouver la seq qu'on veut. Dans notre cas, on a utiliser la fonction levenshtein_distance(seq1, seq2) afin de comparer le score des different seq trouvée avec la seq query en etude.

- Etape 8: la dernière étape consiste à voir si on a trouvé la bonne seq ou pas. On compare le nom de la seq ref qu'on a trouvé avec celle de la query en etude. Si c'est bien la seq désirée alors on incrémente tot_vrai de 1 sinon on passe.