IFT3295 - TP3

21 novembre 2024

Ce TP est à faire en équipe de deux. Vous devez le rendre au plus tard le jeudi 5 décembre 2024 avant la démo (13h30). Le TP ne comprend qu'un seul exercice qui requiert l'implémentation d'un programme imitant 'blast' pour la recherche de séquences biologiques. Les autres consignes sont les mêmes que pour les TPs 1 et 2.

Heuristique de recherche

BLAST (Basic Local Alignment Search Tool) est un algorithme heuristique de recherche qui permet de trouver **très rapidement** des régions similaires entre des séquences. Il s'agit d'un des outils les plus utilisés en bioinformatique et biologie comparative. Les différentes étapes de l'algorithme BLAST peuvent se résumer comme suit :

1. Construire tous les kmers présents dans la séquence input pour un k donné.
2. Rechercher les kmers dans les séquences de la base de données en utilisant un algorithme de recherche exact. BLAST utilise des graines qui requièrent des matchs sur chacune des k positions. Les positions où les kmers match représentent les *hot spot* dénotés HSP (High Scoring Pairs) qui serviront à retrouver l'alignement.
3. Étendre l'alignement, dans les deux directions, aux alentours des HSPs.
4. Évaluer les alignements pour ne garder que ceux pertinents.

Au cours de ce TP, nous allons développer et implémenter **PLAST (Primitive Local Alignment Search Tool)** une heuristique librement basée sur la version originale, "sans gap", de blast pour des séquences de nucléotides.

Votre programme doit **obligatoirement** prendre en entrée : la séquence nucléotidique à rechercher, la banque de données contenant des séquences nucléotidiques et les arguments optionnels suivant : les deux seuils ainsi que le format des graines (explication plus tard). Par exemple :

```
$ python plast.py -i \
>       CGTAGTCGGCTAACGCATACGCTTGATAAGCGTAAGAGCCC \
>       -db tRNAs.fasta -E 5 -ss 10 -seed '11111111111'
```

Votre programme doit retourner les séquences de la banque de données qui sont similaires à la séquence input, ainsi que les informations suivantes : le meilleur HSP identifié et ses statistiques (score, bitscore, position, alignement, evalue). Un exemple d'output pour la commande précédente :

```
>I|gat|Mesostigma_viride
# Best HSP score:78.00, bitscore:24.00, evalue: 7.67e-02
13  CGCATACGCTTGATAAGCGTA  33
23  AGTATACGCCTGATAAGCGTA  43

----------------------------------------
Total :  1
```

1 Implémentation (55 points)

Les différentes sous-sections suivantes vous guideront pour l'implémentation de PLAST.

1.1 Banque de données

Vous disposez d'une banque de données de séquence d'ARNt mitochondriaux de plantes, sous format fasta (voir fichier tRNAs.fasta). Les recherches doivent se faire dans cette banque de données.

1.2 Graines (seeds) et Kmer (10 points)

La stratégie de la recherche consiste à repérer tous les HSPs (fragments similaires) entre la séquence recherchée et les séquences de la banque de données. Pour déterminer un HSP, des mots de longueur fixe sont identifiés

dans un premier temps entre la séquence recherchée et chaque séquence de la banque. L'algorithme originale de BLAST utilise des graines de tailles 11 (avec des match à chaque position, soit "11111111111") pour les nucléotides. Nous utiliserons donc la même graine. **Bonus** : Étendre votre algorithme afin qu'il puisse utiliser d'autres types de graines.

Étant donnée une séquence S en input, et une graine g avec $|g| = k$, vous devez donc, en premier lieu, énumérer tous les mots de tailles k qui se trouvent au sein de S (notez que vous aurez également besoin de la position de chaque k-mer au sein de la séquence input).

1.3 Recherche exacte des mots de taille k dans la banque de donnée (10 points)

Pour chacune des séquences de la banque de données, vous devez trouvez les positions des occurences de chaque kmer. Ces positions (et l'alignement correspondant) représentent les *HSPs initiaux* qu'il faudrait ensuite étendre dans les deux sens.

1.4 Heuristique gloutonne pour l'extension des HSPs (10 points)

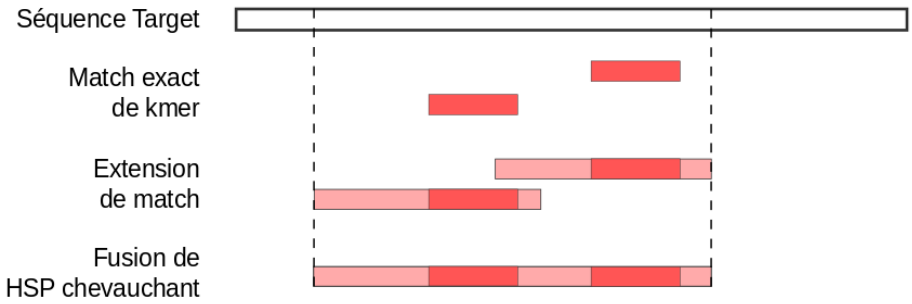
Il faut ensuite étendre la similitude dans les deux directions (en ignorant la possibilité de gaps) le long de chaque séquence, à partir du HSP initial, de manière à ce que le score cumulé puisse être amélioré. Pour ce faire, vous devez considérer simultanément l'extension d'un seul caractère à gauche ou d'un seul caractère à droite. Seule l'extension produisant le meilleur score pour le HSP étendu sera choisie. En cas d'égalité, vous pouvez choisir d'étendre vers la gauche. Considérez un score de $+5$ pour les match de nucléotides et une pénalité de -4 pour les mismatch. Vous devez arrêter l'extension dans une direction, lorsque l'une des conditions suivantes est obtenue :

— La fin de l'une des séquences est atteinte.

— Le score cumulé descend d'une quantité x donnée par rapport à la valeur maximale qu'il avait atteint. **Cette quantité constitue le premier seuil (-E)** et sa valeur par défaut est de **4**. Dans ce dernier cas l'extension dans les deux directions est totalement arrêtée et on ne conserve pas la dernière extension.

1.5 Fusion des HSPs chevauchants (10 points)

Vous devez ensuite fusionner tous les HSPs se chevauchant et impliquant les mêmes séquences. Pensez aux coordonnées de début et de fin des HSPs dans les deux séquences. Le schéma suivant illustre les différentes étapes.



1.6 Statistique sur les HSPs et cutoff (10 points)

Afin de ne conserver que les HSPs pertinents, il est important de les comparer puis filtrer selon un système de scorage robuste. Malheureusement, il n'est pas possible de comparer les HSPs en fonction de leurs scores bruts, puisque ces derniers sont biaisés par la taille des alignements. Nous utiliserons à la place un type de score appelé **bitscore** qui est défini comme suit :

$$B = round\Big(\frac{\lambda S - \ln(K)}{ln(2)}\Big)$$

où S désigne le score brut du HSP, λ et K sont des constantes avec pour valeurs respectives : 0.192 et 0.176.

À partir de ce score vous pouvez calculer la *e-value* qui représente le nombre attendu, par chance, de HSP avec un bitscore d'au moins B. Plus la e-value d'un HSP est petite, plus ce dernier est pertinent.

$$e = mn2^{-B}$$

avec m représentant la taille totale des séquences de la banque de données, n la taille de la séquence à rechercher et B le bitscore du HSP.

Le score de signifiance des HSPs est déterminé par leurs e-values. **Le second seuil (-ss) correspondant donc au seuil de signification et sa valeur par défaut est de 1e-3**. Vous devez donc retourner uniquement

les HSPs qui sont significatifs et un seul HSPs significatif par séquence (celui avec le meilleur bitscore).

1.7 Output de votre programme (5 points)

Pour chacune des séquences de la banque de données qui est similaire à la séquence en entrée, vous devez retourner le meilleur HSP pour la séquence (voir exemple plus haut). Votre output doit être trié par pertinence des résultats.

2 Questions sur la mise en pratique (45 points)

Vous devez ensuite répondre aux questions suivantes en vous servant de votre programme.

1. Donnez l'output de votre programme pour chacune des séquences du fichier *unknown.fasta* qui contient des séquences d'ARNts dont la nature est inconnue. On vous demande d'utiliser les paramètres par défaut (-ss = 0.001, -E = 4, seed = '11111111111').
2. En déduire, **si possible** la nature (acide aminé et anticodon) de chacune des séquences. Notez que l'identifiant des séquences de la banque de données est sous la forme : Acide Aminé|Anticodon|Espèce. Vous pouvez au besoin jouer avec le seuil de signification (-ss) pour affiner votre recherche.
3. Vérifiez vos résultats en vous servant du véritable outil BLAST pour nucléotide (BLASTN) disponible sur NCBI ==> `https://blast.ncbi.nlm.nih.gov/Blast.cgi` . On vous demande de comparez les deux résultats.
4. **Bonus :** Qu'arrive t'il lorsque vous utilisez des graines plus longues ? plus courtes ? (on vous demande l'impact sur la vitesse, la précision et la sensibilité de PLAST)
5. **Bonus :** PatternHunter, un autre algorithme de recherche, utilise des graines espacés par des positions "*don't care*" comme "111010010100110111". Adaptez votre algorithme aux graines de PatternHunter et comparez ses perfomances à celui de l'algorithme original (utiliser un seuil de signifiance (-ss) moins stringent tel que 0.01)