université de BORDEAUX

OBI

Projet Python 2023

Lucien PIAT Sara SPIELER

Superviseurs:

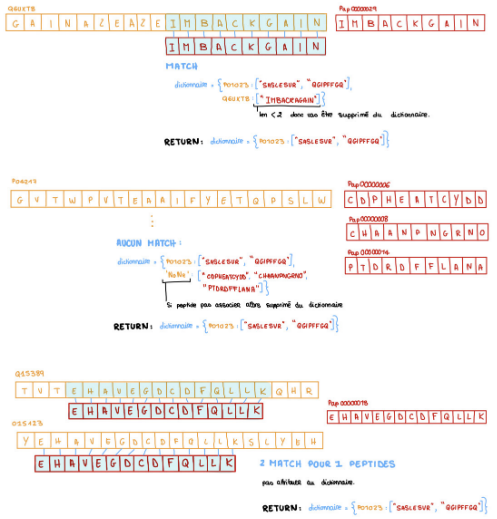
R. Uricaru S. Karkar

0.1 Observation des fonctions

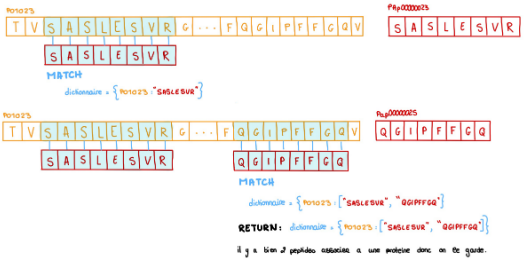
0.1.1 Exemple concret

Give toy examples to show on which cases your identifier finds the original set of proteins

Notre jeu de données étant très volmineux, des jeux "test" ont été produits pour observer le fonctionnement du programme. Ci dessous, sont illustrés quelques exemples. Notez que les jeux de test son disponible avec les scripts.



(a) Ajout d'une séquence valide avec deux peptides



(b) Exemple de refus des séquences invalides

Figure 1: Exemples d'exécution de la fonction unique_match_set sur des données réelles

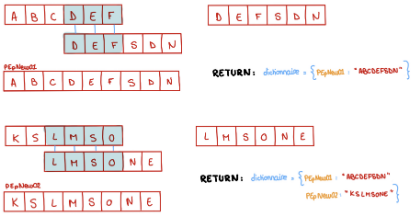


Figure 2: Exemples d'exécution de la fonction assembely_peptides sur des données fictives

0.1.2 Similarité entre les mesures et le jeu de données

How close is your solution to the human_blood protein set ?

Notre banque de donnés contient un total de 772 protéines. En appliquant la fonction unique_match_set 365 d'entre elles possèdent au moins deux peptides uniques qui leurs sont associés. Cela représente 47,2 % des protéines totales, ce qui pour des données biologiques semble être un très bon début.

Malheureusement, après avoir ajouté les peptides assemblés en sortie de la fonction assembely_peptides, les 173 protéines identifiées faisaient toutes parties des protéines déjà reconnues. Il serait intéressant de refaire le pipeline en modifiant l'overlap min (ici a 15) pour obtenir plus de peptides d'assemblage.

0.1.3 Solution en cas concret

May your solutions be used on real world cases ?

Afin de poser une base statistique sur nos affirmations, on réalise un test comparant la fraction des protéines identifiées au nombre total de protéines.

> On observe que 47,2 % des protéines du jeu de donné sont identifiées. La proportion des protéines identifiées est significativement différente de la prorportion théorique ($p < 0.0001$, $X2 = 545$, $ddl = 1$, test de comparaison de proportion à proportion, unilatéral). On conclut que l'échantillon étudié devrait être raffiné avant d'être utilisé en situation réelle.

0.1.4 Amélioration biologique

Can your identifier deal with all the biological cases ?

La solution que nous avons produite semble prometteuse, en revanche, comme présentée dans la partie 2, la lenteur du script rend son utilisation en dur relativement complexe. Dans le cas où les mesures sont répètes et que l'analyse doit être menée plusieurs fois, le temps de calcul pourrait limiter les activités du laboratoire.
Malgré tout, les scripts produisent des résultats lisibles et ergonomiques, facilitant leur utilisation future.

D'autre part, nos fonctions sont écrites de façon a limiter la plupart des erreurs liées aux défauts du jeu de données. En effet, chaque peptide doit être unique. De plus, un manque de précision dans les mesures sera comblé par la redondance de l'analyse, les protéines sont discriminées par au moins deux peptides et, le chevauchement minimal lors de l'assemblage de ces derniers permet d'assurer une bonne fiabilité des résultats.

0.2 Amélioration possible

0.2.1 Test de vitesse

Notre fonction unique_match_set est relativement rapide, 12 minutes de calcul environ pour le jeu de donné entier. Par contre, la fonction assembly_peptides est très chronophage. L'assemblage du fichier complet pour un overlap minimal à 15 a pris 8h40 en tout pour un total de 3410 peptides assemblés soit environ 7 peptides par minute.

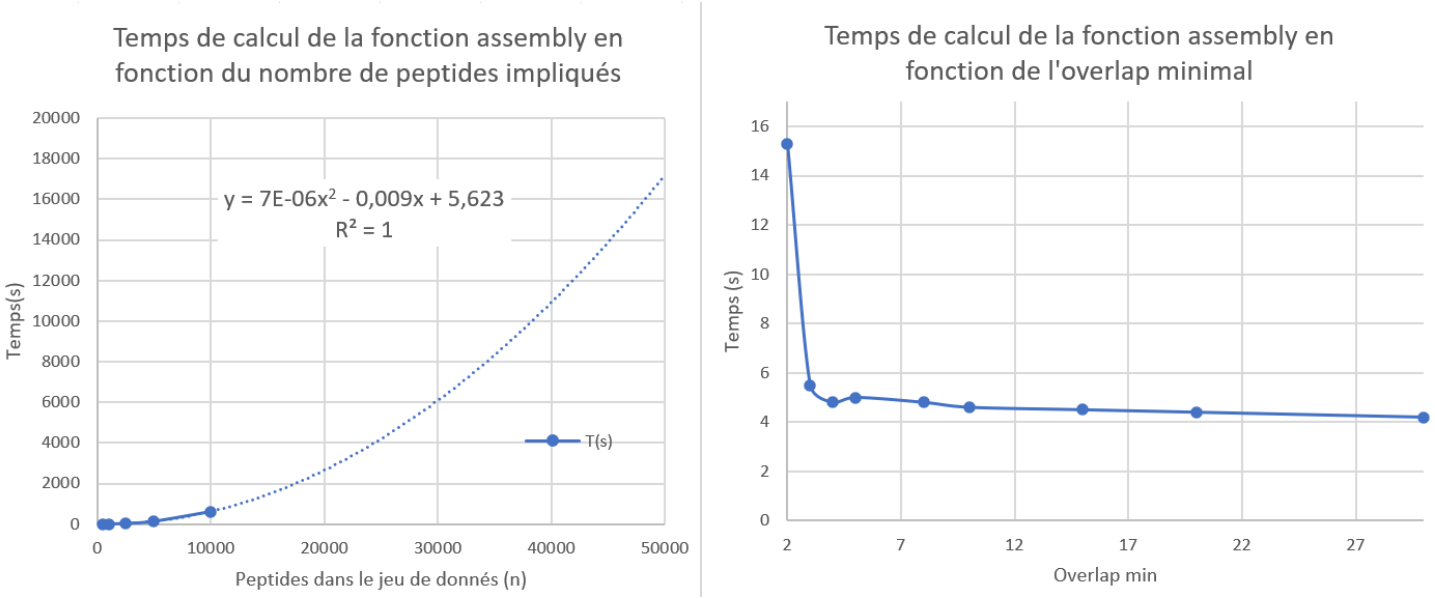


Figure 3: A gauche, temps de calcul de assembly_peptides en fonction du nombre de peptides impliqués, la croissance du temps semble être polynomiale. A droite, le temps de calcul en fonction de l'overlap minimal, ce dernier ne semble pas avoir d'influence au-dessus de 2

0.2.2 Perspectives d'amélioration

Nos fonctions, ont déjà par le passé été codées et optimisées par de nombreux bioinformaticiens. En lisant les publications à ce sujet, on peut facilement identifier deux perspectives d'améliorations majeures pour notre code.

Cock, P.& de Hoon, M. (2009). Biopython: freely available Python tools for computational molecular biology and bioinformatic. in Oxford University Press (OUP). https://doi.org/10.1093/bioinformatics/btp163

Comme cité dans la publication sur le package "Biopython" le language C offre de bien meilleures performances pour les traitements répétitifs que nous faisons ici. De plus, l'ajout de "Multithreading" dans nos programmes, nous permettrait d'utiliser la puissance de nos machines à leurs pleins potentiels divisant le temps de calcul par le nombre de coeurs de notre CPU.

Touts les scripts et fichiers sont sur GitHub