Debrief du vendredi 27/01

1 Récapitulatif de la journée

Pour être sûr que certains baitModels n'ont pas la même masse et pour avoir plus d'informations sur le fichier de données utilisé, G. Fertin et G. Jean m'ont demandé :

- de vérifier que les baitModels ont tous la même masse
- de noter le nombre minimum, maximum et moyen de baitModels
- pour chaque baitModel :
  - ◇ la longueur du tronçon le plus long d'acides aminées (longest stretch)
  - ◇ le nombre de masses dans le baitModels (dont le nombre de masses ne correspondant à aucune combinaison d'acides aminées dans la table des masses *mass_table_8.csv*, le nombre de masses correspondant à une seule combinaison d'acides aminées et le nombre de masses correspondant à plusieurs combinaisons d'acides aminées dans la table)
- toute autre information d'intérêt

De plus, après observation des résultats obtenus par la deuxième méthode en ignorant les masses, il faudrait tester la méthode en remplaçant les masses par des tirets ("-") pour que les algorithmes utilisés (ClustalW ou MUSCLE) alignent les séquences en prennant en compte les écarts. La première suggestion est de remplacer chaque masse m par k tirets où k est la longueur de la combinaison la plus longue ayant une masse m dans la table des masses.

2 Statistiques sur le fichier de données

En plus des informations demandées, les statistiques suivantes ont aussi été relevées :

- pour chaque bait :
  - ◇ le nombre de baitModels associés au bait
  - ◇ la moyenne des masses des baitModels associés au bait
  - ◇ l'écart-type des masses des baitModels
- pour chaque baitModel :
  - ◇ sa masse

J'ai implémenté un script *checkMass.py* (se trouvant dans le répertoire "instances" sur le dossier UNcloud) qui vérifie qu'il n'y a pas de baitModels avec des masses différentes (la différence de masse étant d'au moins 1 Da). Il y a bien des baitModels différant d'au moins 1 Da dans le fichier *bait10000.txt* (3 350 baits sur les 10 000).

Le script *convert.py* a été modifié pour qu'il génère un fichier texte contenant les statistiques susmentionnées (ce fichier *stats_bait10000.txt*) se trouve dans le répertoire "instances/stats"). De plus, un script *getStats.py* a été ajouté dans le même répertoire pour simplifier

la lecture des données (il permet d'obtenir les statistiques pour un bait donné ou pour l'ensemble des baits contenus dans le fichier *stats_ bait10000.txt* ou tout autre fichier du même format). Un fichier README se trouve dans le répertoire détaillant le format du fichier *stats_ bait10000.txt*. **EDIT :** une version ultérieure fournie également le nombre de pic en communs ("Shared Peak Count") et le score SpecGlob pour chaque baitModels.

3 Première version de la méthode linéaire

3.1 Idée

J'ai implémenté la méthode de résolution linéaire en Python (script *linearBaitFusion.py*). Pour lancer ce script, il faut lui passer le chemin vers le fichier de statistiques mentionné ci-dessus et une table de masses (e.g. `python3 linearBaitFusion.py stats_bait10000.txt mass_table_8.csv`).

Avec cette première version j'ai retrouvé 3 886 baits sur les 7 663 baits contenant au moins deux baitModels. Pour fusionner les baitModels, l'algorithme utilise autant de curseurs que de baitModels. Ces curseurs peuvent être vus comme une tête de lecture pour chaque baitModel pouvant lire un acide aminé à la fois. De plus, ces curseurs peuvent se déplacer à des rythmes différents selon la validité de leur baitModel (voir suite). L'algorithme est itératif et se décline en deux phases à chaque itération :

1. l'acide aminé sous chaque curseur est lu et est considéré comme un candidat. Chaque curseur ayant lu un acide aminé propose un candidat (si plusieurs curseurs ont lu le même acide aminé alors on considère que cet acide aminé, ou candidat, a autant de votes que de curseurs l'ayant lu). Une élection à la majorité est alors effectuée pour choisir un seul candidat (e.g. si 4 curseurs ont lu l'acide aminé Y et 2 autres ont lu l'acide aminé G alors le candidat élu sera Y). Nous présenterons par la suite la stratégie utilisée en cas d'ex aequo. Il est aussi possible pour un baitModel de proposé une masse comme candidat au lieu d'un acide aminé.
2. une fois l'élection terminée, les baitModels "ayant voté" pour l'élu sont considérés comme valides. Ceux ayant voté pour un autre acide aminé sont considérés comme étant en probation, si lors d'une prochaine élection ils proposent un candidat ne correspondant pas au candidat élu alors ils seront considérés comme invalides. D'autre part, pour chaque masse proposée par des baitModels on soustrait la masse du candidat élu à la masse proposée. Le baitModel proposera la nouvelle masse obtenue au prochain tour. Ce mécanisme est repris à chaque itération jusqu'à ce la masse corresponde à la masse d'un acide aminé (si la masse proposée par un baitModel correspond à la masse d'un acide aminé on considère que le candidat proposé par le baitModel est l'acide aminé en question). Si la masse obtenue après la soustraction de la masse du candidat est inférieure à 1 Da en valeur absolue alors on considère qu'elle est nulle. Cependant, si la masse obtenue est négative (strictement inférieure à -1 Da) alors on considère que le baitModel est invalide indéfiniment.

Après l'étape de validation des baitModels, les curseurs des baitModels n'étant pas invalides sont avancés d'un cran. **La séquence en sortie est la concaténation des élus.**

L'algorithme s'arrête lorsque l'acide aminé K ou R est retrouvé, si tous les curseurs des baitModels valides ou en probation arrivent à la fin de leur baitModel respectif, ou si aucun baitModel ne propose de candidat. Tous les élus obtenus jusqu'à interruption de l'algorithme sont pris en compte. Lorsque l'algorithme est interrompu parce qu'aucun candidat n'est proposé ou qu'il est impossible de départager deux candidats on considère que la résolution est incomplète.

En plus de ces deux phases, certains mécanismes ont été ajoutés pour ne pas trop pénaliser les baitModels ou pour gérer les situations d'ex aequo :

- Les baitModels sont pondérés : un baitModel valide se verra attribué un score de 2, un baitModel en probation se verra attribué un score de 1 et un baitModel invalide un score de 0. Lors d'un ex aequo entre plusieurs candidats un premier score est utilisé pour départager les candidats. Ce score est noté $score_A$ et correspond à la somme des pondérations attribuées aux baitModels votant pour un candidat donné. Le groupe de baitModels ayant le $score_A$ le plus grand est celui dont le candidat sera retenue. Le but de ce score est de favoriser les candidats proposés par des baitModels valides. Par exemple, trois baitModels respectivement valides (le $groupe_1$) votent pour l'acide aminé I et trois autres baitModels en probation (le $groupe_2$) votent pour l'acide aminé K alors ce sera l'acide aminé I qui sera élu étant donné que $score_A(groupe_1) = 6$ alors que $score_A(groupe_2) = 3$.

  Dans le cas où les scores $score_A$ sont égaux pour chaque groupe de baitModels alors on attribue à chaque baitModel d'un groupe un score calculé à partir du longest stretch (LS) du baitModel, au nombre de masses du baitModel ne correspondant pas à une combinaison d'acides aminés (GUM), au nombre de masses du baitModel correspondant à une et une seule combinaison d'acides aminés (GSC) et au nombre de masses du baitModel correspondant à plusieurs combinaisons possibles d'acides aminés (GMC). Le score attribué à chaque baitModel est calculé comme suit :

  $$e^{\frac{LS}{10+10*GSC+100*GMC+1000*GUM}}$$

  On définit alors le $score_B$ d'un groupe comme étant la somme des scores attribués à chaque baitModel compris dans le groupe multiplié par la pondération du baitModel (valide, en probation, invalide). Le but de ce score est de favoriser les baitModels ayant le plus de masses "simplifiables" et les plus longs tronçons d'acides aminés. Ainsi, si un baitModel valide propose le candidat I et un autre baitModel valide propose le candidat K et que les deux baitModels ont un longest stretch de 5 mais le premier a une masse ne correspondant à aucune combinaison d'acides aminés tandis que le deuxième a une masse correspondant à une et une seule combinaison d'acides aminés dans la table des masses. Alors, les scores respectifs des deux baitModels seront 1.00 et 1.28. Le candidat du deuxième baitModel est alors choisie parce qu'il a une masse correspondant à une combinaison d'acides aminés contrairement au premier baitModel ayant une masse "inexplicable". K est alors élu.

  Enfin, si le $score_B$ ne peut départager les candidats l'algorithme est interrompu et la résolution est incomplète.
- Une stratégie de réinsertion (ou "comeback") est utilisée pour poursuivre l'utilisation de baitModels précedemment invalidés. Lors de la phase de validation, si le curseur d'un baitModel invalide est sur un acide aminé correspondant à l'élu actuel alors

on considère que le baitModel est en probation. L'intérêt est qu'un baitModel peut reproposé des candidats après qu'il ait été invalidé peut-être "injustement".

3.2 Illustration

Voici une illustration de la méthode sur le bait GGSGATIIMVVQR avec les baitModels suivant :

- GGSQTI[570.32]R
- GGSGATII[457.24]R
- NSG[44.07]VVMII[128.01]QR
- GGSQTIIMVVQR
- [114.04]SGATI[343.19]VQR