

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 aminés (longest stretch)
 - ◊ le nombre de masses dans le baitModels (dont le nombre de masses ne correspondant à aucune combinaison d'acides aminés dans la table des masses *mass_table_8.csv*, le nombre de masses correspondant à une seule combinaison d'acides aminés et le nombre de masses correspondant à plusieurs combinaisons d'acides aminés 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 prenant 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 fournit 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 proposer 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 précédemment 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 repropose 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

La position des curseurs est représentée par la couleur rouge. La séquence obtenue itérativement est représentée en bleu. Nous ne représenterons pas explicitement les états des baitModels ici mais nous préciserons lorsqu'ils passent d'un état à un autre. Nous numérotions les baitModels de 1 à 5 en partant du haut vers le bas (le baitModel 4 est donc le baitModel GGSQTIIMVVQR).

```

G G S Q T I [570.32]
G G S G A T I I [457.24] R
N S G [44.07] V V M I I [128.01] Q R
G G S Q T I I M V V Q R
[114.04] S G A T I [343.19] V Q R


---


G

```

Initialement, tous les curseurs sont placés sur le premier élément de leur baitModel respectif. Les baitModels 1, 2 et 4 votent pour l'acide aminé G. Le baitModel 3 vote pour l'acide aminé N tandis que le baitModel 5 propose une masse de 114.04 Da. L'acide aminé G est élu à la majorité, la masse de G est donc soustraite à la masse proposée par le baitModel 5 et le baitModel 3 sera maintenant placé en probation parce qu'il propose un acide aminé différent de G. Les curseurs sont avancés d'un cran.

```

G G S Q T I [570.32]
G G S G A T I I [457.24] R
N S G [44.07] V V M I I [128.01] Q R
G G S Q T I I M V V Q R
[57.02] S G A T I [343.19] V Q R


---


G G

```

La masse proposée par le baitModel 5 correspond exactement à la masse de l'acide aminé G. Les baitModels 1, 2, 4 et 5 votent donc pour G. Le baitModel 3 précédemment en probation est maintenant invalidé car il propose à nouveau un acide aminé différent de l'élu. Un deuxième G est rajouté à la séquence de sortie. Les curseurs avancent d'un cran sauf celui du baitModel 3 qui est invalidé.

```

G G S Q T I [570.32]
G G S G A T I I [457.24] R
N S G [44.07] V V M I I [128.01] Q R
G G S Q T I I M V V Q R
      S G A T I [343.19] V Q R


---


G G S

```

Les baitModels valides (1, 2, 4 et 5) proposent tous S. S est élu. Le curseur du baitModel 3, précédemment invalidé, est sur l'acide aminé S donc à travers la stratégie "comeback" le baitModel 3 est remplacé en probation. Les curseurs sont avancés.

```

G G S Q T I [570.32]
G G S G A T I I [457.24] R
N S G [44.07] V V M I I [128.01] Q R
G G S Q T I I M V V Q R
      S G A T I [343.19] V Q R


---


G G S G

```

Les baitModels 1 et 4 votent pour l'acide aminé Q, les baitModels 2, 3 et 5 votent pour G. G est élu. Les baitModels 1 et 4 sont placés en probation. Jusqu'ici les baitModels 1, 3 et 4 sont en probation tandis que 2 et 5 sont valides. Les curseurs sont avancés.

```

G G S Q T I [570.32]
G G S G A T I I [457.24] R
N S G [44.07] V V M I I [128.01] Q R
G G S Q T I I M V V Q R
      S G A T I [343.19] V Q R


---


G G S G A

```

Les baitModels 1 et 4 proposent l'acide aminé T, les baitModels 2 et 5 proposent l'acide aminé A. T et A sont ex aequo lors de l'élection à la majorité. Le $score_A$ est utilisé pour les départager. Les baitModels 1 et 4 ($groupe_1$) sont en probation donc $score_A(groupe_1) = 2$ tandis que les baitModels 2 et 5 ($groupe_2$) sont valides d'où $score_A(groupe_2) = 4$. Le candidat du $groupe_2$ est alors élu, c'est-à-dire A. Les baitModels 1 et 4 sont alors éliminés parce qu'ils étaient en probation et sont à nouveau en désaccord. Le baitModel3 propose une masse de 44.07 Da. Or, $44.07 - m(A) < -1$ donc la masse de 44.07 Da ne correspond à aucun acide aminé. Le baitModel 3 est définitivement invalidé. Les curseurs du baitModel 2 et 5 sont avancés.

```

G G S Q T I [570.32]
G G S G A T I I [457.24] R
N S G [44.07] V V M I I [128.01] Q R
G G S Q T I I M V V Q R
      S G A T I [343.19] V Q R


---


G G S G A T

```

Les baitModels 2 et 5 proposent le candidat T. C'est le seul candidat donc il est élu. Les baitModels 1 et 4, précédemment invalidés, sont aussi sur l'acide aminé T donc il sont remis en probation. Tous les curseurs sont avancés sauf celui du baitModel 3 qui est maintenant détruit parce que le baitModel 3 est invalidé indéfiniment.

```

G G S Q T I [570.32]
G G S G A T I I [457.24] R
N S G [44.07] V V M I I [128.01] Q R
G G S Q T I I M V V Q R
          S G A T I [343.19] V Q R


---


G G S G A T I

```

Tous les baitModels toujours valides ou en probation votent pour I. I est donc choisie à l'unanimité. Les curseurs sont avancés.

```

G G S Q T I [570.32]
G G S G A T I I [457.24] R
N S G [44.07] V V M I I [128.01] Q R
G G S Q T I I M V V Q R
          S G A T I [343.19] V Q R


---


G G S G A T I I

```

Les baitModels 2 et 4 votent pour l'acide aminé I. Les baitModels 1 et 5 proposent respectivement les masses 570.32 Da et 343.19 Da. La masse $m(I)$ de I est soustraite à ces masses. Les curseurs sont avancés.

```

G G S Q T I [457.24]
G G S G A T I I [457.24] R
N S G [44.07] V V M I I [128.01] Q R
G G S Q T I I M V V Q R
          S G A T I [230.11] V Q R


---


G G S G A T I I M

```

Les baitModels 1, 2 et 5 proposent respectivement les masses 457.24 Da, 457.24 Da et 230.11 Da. Le baitModel 4 vote pour M. M est élu et la masse de M est soustraite aux masses proposées par 1, 2 et 5. Les curseurs sont avancés.

```

G G S Q T I [326.2]
G G S G A T I I [326.2] R
N S G [44.07] V V M I I [128.01] Q R
G G S Q T I I M V V Q R
          S G A T I [99.07] V Q R


---


G G S G A T I I M V

```

Le baitModel 4 vote pour V. La masse proposée par le baitModel 5 correspond à la masse de V donc le baitModel 5 vote aussi pour V. V est élu et la masse de V est soustraite des masses proposées par le baitModel 1 et le baitModel 2. Les curseurs sont avancés.

```

G G S Q T I [227.13]
G G S G A T I I [227.13] R
N S G [44.07] V V M I I [128.01] Q R
G G S Q T I I M V V Q R
          S G A T I          V Q R
-----
G G S G A T I I M V V

```

Les baitModels 4 et 5 votent pour V. V est élu. La masse de V est soustraite des baitModels 1 et 2. Les curseurs sont avancés.

```

G G S Q T I [128.06]
G G S G A T I I [128.06] R
N S G [44.07] V V M I I [128.01] Q R
G G S Q T I I M V V Q R
          S G A T I          V Q R
-----
G G S G A T I I M V V Q

```

Les baitModels 4 et 5 votent pour Q. Les masses proposées par les baitModels 1 et 2 correspondent respectivement à la masse de Q. Q est élu. Les curseurs des baitModels 2, 4 et 5 sont avancés. Le curseur du baitModel 1 est arrivé en bout de chaîne et n'est plus pris en compte.

```

G G S Q T I
G G S G A T I I          R
N S G [44.07] V V M I I [128.01] Q R
G G S Q T I I M V V Q R
          S G A T I          V Q R
-----
G G S G A T I I M V V Q R

```

Les baitModels 2, 4 et 5 votent pour l'acide aminé R. R est élu. L'algorithme est terminé. La séquence reconstituée est GGS GATIIMVVQR. Elle correspond exactement à la séquence du bait.

3.3 Algorithme de la première méthode

D'un point de vue algorithmique, on peut représenter les curseurs avec deux tableaux. Un tableau est utilisé pour stocker la position de chaque curseur tandis que l'autre tableau est utilisé pour stocker les masses éventuellement proposées par les baitModels. Enfin, un troisième tableau est utilisé pour stocker l'état de chaque baitModel. Les états possibles sont 2 (le baitModel est valide), 1 (le baitModel est en probation) et 0 (le baitModel est invalide). Tous les baitModels sont initialement valides. Pour compter le nombre de baitModels ayant voté pour un candidat nous utilisons une table de hachage qui à un acide aminé associe un compteur. De la même façon, les scores A et B attribués à un candidat (un acide aminé) sont stockés dans des tables de hachage (représentées par des $\langle \rangle$).

Algorithme 1 Algorithme méthode linéaire

fonction LINEARBAITFUSION(baitModels) :n \leftarrow length(baitModels)candidate \leftarrow $\overline{}$ keepgoing \leftarrow *True*indices[n] \leftarrow {0}

▷ Initialisation des curseurs

masses[n] \leftarrow {0.0}validation[n] \leftarrow {2}fusion \leftarrow empty_string()

▷ Boucle principale

Tant que candidate $\neq K$ **et** candidate $\neq R$ **et** keepgoing, **faire** :candidate \leftarrow DoElection(n, baitModels, indices, masses, validation)**Si pas** isAminoAcid(candidate) **alors**keepgoing \leftarrow *False***exit loop****Sinon**fusion \leftarrow fusion + candidate**Fin si****Pour** i de 1 à n, **faire** :

▷ Étape de validation des baitModels

Si $0 \leq \text{indices}[i] < \text{length}(\text{baitModels}[i])$ **alors**c \leftarrow baitModels[i][indices[i]]**Si** c = candidate **et** absoluteValue(masses[i]) ≤ 1 **alors****Si** validation[i] = 0 **alors**validation[i] \leftarrow 1

▷ Comeback

Fin si**Sinon****Si** masses[i] ≥ 1 **alors****Si** masses[i] - getAminoAcidMass(candidate) ≥ -1 **alors**masses[i] \leftarrow masses[i] - getAminoAcidMass(candidate)**Sinon**indices[i] \leftarrow -1validation[i] \leftarrow 0**Fin si****Sinon, si** isAminoAcid(c) **et** validation[i] = 2

validation[i] = 1

Sinon

validation[i] = 0

Fin si**Fin si****Fin si****Fin pour****Fin tant que**

renvoyer fusion

Fin fonction

Algorithme 2 Fonction DoElection

```
fonction DOELECTION(n, baitModels, indices, masses, validation) :  
    candidate  $\leftarrow$  _  
    scoresA  $\leftarrow$   $\langle \rangle$   
    scoresB  $\leftarrow$   $\langle \rangle$   
    candidates  $\leftarrow$   $\langle \rangle$   
  
    Pour i de 1 à n, faire :  
        Si  $0 \leq \text{indices}[i] < \text{length}(\text{baitModels}[i])$  alors  
            c  $\leftarrow$  baitModels[i][indices[i]]  
            Si isMass(c) alors  
                masses[i]  $\leftarrow$  masses[i] + readMass(c)  
            Fin si  
  
            Si masses[i] < -1.0 alors  
                validation[i]  $\leftarrow$  0  
                skip iteration  
            Sinon, si absoluteValue(masses[i])  $\geq 1$   
                Si isAminoAcid(masses[i]) alors  
                    c  $\leftarrow$  getAminoAcidWithMass(masses[i])  
                Fin si  
            Sinon  
                masses[i]  $\leftarrow$  0.0  
            Fin si  
        Fin si  
  
        Si c  $\neq$  _ alors  
            Si c  $\notin$  candidates alors  
                candidates[c]  $\leftarrow$  0  
                scoresA[c]  $\leftarrow$  0  
                scoresB[c]  $\leftarrow$  0  
            Fin si  
            candidates[c]  $\leftarrow$  candidates[c] + 1  
            scoresA[c]  $\leftarrow$  scoresA[c] + validation[i]  
            scoresB[c]  $\leftarrow$  scoresB[c] + validation[i] * getBaitModelScore(baitModel[i])  
        Fin si  
    Fin pour  
  
    candidate  $\leftarrow$  selectCandidateWithBestScore(candidates, scoresA, scoresB)  
    renvoyer candidate  
Fin fonction
```
