INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

4BIM

Réarrangements dans le génome de la bactérie Yersinia Pestis

Alice Genestier, Hanâ LBATH, Emilie MATHIAN

2019

Contents

1 Introduction **2**

2 Implémentation d'une heuristique optimisant le nombre d'inversions **2**
2.1 Contexte historique et biologique 2
2.2 Implémentation d'une heuristique 3
2.2.1 Calcul d'un score d'organisation 3
2.2.2 Énumération des permutations 3
2.2.3 Construction de l'arbre 5
2.3 Génération d'échantillons aléatoires 6

3 Comparaisons de deux génomes bactériens complets **7**
3.1 Comparaison des génomes médiéval et actuel 7
3.2 Dot matrix et filtrage des orthologies 7
3.3 Réarrangements 10

1 Introduction

Ce projet s'intéresse à la comparaison de génome à l'échelle chromosomique. Cette comparaison repose sur l'hypothèse que l'ordre des gènes sur les chromosomes d'organismes apparentés est plus ou moins conservé. Le nombre d'inversions nécessaires pour retrouver l'ordre des gènes homologues entre un organisme B supposé apparenté à un organisme A, est ainsi l'objet d'étude. Deux approches seront proposées :

- Une heuristique sera tout d'abord implémentée pour optimiser le nombre minimal d'inversions nécessaires pour retrouver l'ordre d'une séquence de gènes. Cette méthode permettra d'estimer la distance évolutive séparant deux espèces, mais son utilisation sera limitée par la taille de la comparaison à effectuer. Ainsi seulement un petit nombre de gènes ne pourra être pris en compte.
- Face à cette limite un modèle reposant sur un parcours de graphe sera implémenté. Cette seconde méthode sera appliquée pour comparer les chromosomes bactériens de *Yersinia pestis CO92* et de la souche médiévale responsable de la peste noire.

2 Implémentation d'une heuristique optimisant le nombre d'inversions

2.1 Contexte historique et biologique

Dans les années 1910, alors que l'observation directe de l'ordonnancement des gènes sur les chromosomes est inaccessible, la communauté scientifique discute d'une possible organisation linéaire des gènes sur les chromosomes. Cette organisation fera finalement consensus dans les années 1940, notamment grâce aux travaux de Sturtevant qui a introduit la notion de probabilité de liaison. Le chercheur découvre également les mécanismes d'inversion et introduit cette notion pour la comparaison des génomes à l'échelle des chromosomes. En 1937, il tente ainsi d'expliquer le lien de parenté entre deux espèces de drosophiles *Pseudoobscura* et *D. Melanogaster* par reconstitution des évènements d'inversion nécessaires pour retrouver l'ordre des gènes sur les chromosomes [1]. Il ne sera démontré que bien plus tard que ce problème est NP complet.
Dans l'exposé qui suit nous allons tenter de modéliser le problème proposé par Sturtevant par une heuristique. Pour ce faire, nous prendrons en référence l'une des deux espèces. Puis nous tenterons de calculer le nombre minimal d'inversions nécessaires pour retrouver l'ordre des gènes de l'espèce de référence à partir de l'organisation observée chez la seconde espèce.

2.2 Implémentation d'une heuristique

2.2.1 Calcul d'un score d'organisation

Q1 : Afin de connaître à tout moment l'état d'organisation de la séquence en cours d'analyse nous avons mis en place un score. Ce score traduit le nombre de lettres bien placées, selon l'ordre alphabétique. Pour le calculer nous avons dénombré le nombre d'éléments adjacents, soit le nombre de lettres consécutives dans l'ordre alphabétique, grâce à la fonction `adjacent()`. Le score correspondant à une séquence est égale à la longueur de la liste retournée par `adjacent()` plus un si la première lettre est bien placé et plus un si la dernière lettre est bien placée. Ce score est calculé par la fonction `Score()`. Pour simplifier notre exposé nous parlerons de séquences de lettres, bien que pour faciliter l'implémentation de la méthode les séquences de lettres sont généralement converties en liste d'entiers correspondant aux valeurs ascii des caractères, via la fonction `ConvertAsci()`.

```
CALCUL DU SCORE  :

  Mot : LHFEBADCKIJGM
Score  =  1 +1 +1  +1  +1 = 5
```

Figure 1: Exemple de calcul du score

2.2.2 Énumération des permutations

Q2 : Pour résoudre le problème de Sturtevant, chaque inversion doit permettre d'augmenter la taille de la séquence triée, soit le score. Ainsi d'après l'exemple mentionné ci-dessus (figure 1), nous pourrions inverser la séquence pour placer le "A" en première position, le "B" serait alors bien placé. Puis réaliser une nouvelle inversion pour placer le "C" à la suite du "B, ce qui permettrait de bien positionner la lettre "D" etc. Cette logique permettrait d'obtenir la solution mais le nombre d'inversions nécessaire serait sous optimal. En effet en réalisant un tel processus, 8 inversions sont nécessaires.
Pour améliorer ce résultat il faut retenir toutes les inversions permettant d'augmenter le score, et pas nécessairement celles permettant de trier le début de la séquence. Pour ce faire nous raisonnerons en terme d'arbre, en mémorisant tout les chemins menant à la solution nous pourront retrouver la solution optimale.
Nous avons implémenté deux versions d'inversion, la première est nommée `permutation()`, elle énumère **certaines** permutations possibles à chaque évènement. Les inversions doivent permettre de conserver ou d'augmenter le score. Reprenons l'exemple 1 :

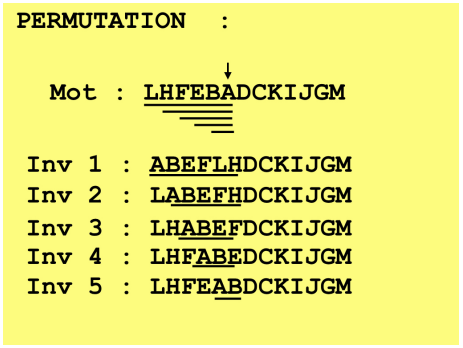

Figure 2: Énumération des permutations

La fonction `permutation()` prend en argument le score courant de la séquence et fait appel à la fonction `end_ord()` qui retourne l'indice de la dernière lettre bien placée. Puis elle appelle la fonction `inversion()` qui permet de permuter l'ordre de la séquence. Ces étapes sont répétées itérativement pour `i` allant de 2 à la taille de la séquence à inverser. La fonction `permutation()` retourne une liste des inversions ayant permis d'augmenter ou de conserver le score, chaque élément de la liste est donc associé à la séquence permutée, le score associé à celle-ci, et la distance à l'origine de cette séquence (voir la section suivante).

Une seconde version de permutation, nommée `permutation_v2()` a été écrite pour prendre en compte les inversions possibles en dehors des éléments compris entre la fin de la séquence triée et la première lettre minimale. Elle permet ainsi de tester d'avantage de combinaisons, toutefois le temps de calcul est fortement augmenté. Seules les inversions permettant d'augmenter ou de maintenir le score seront conservées, de plus si la distance à l'origine est supérieure à `taille de seq + 1`, cette possibilité est rejetée, car quelque soit le scénario cette solution sera sous optimale. Reprenons l'exemple 1 :

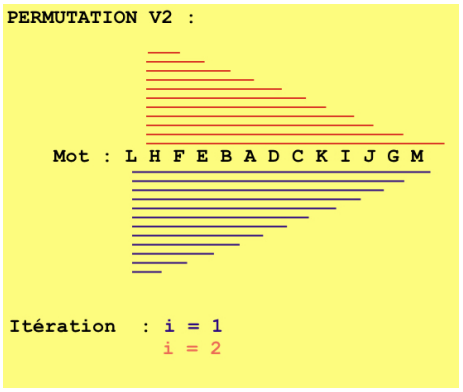

Figure 3: Combinaisons testées par permutation_v2()

Les avantages de la fonction `permutation_v2()` peuvent être explicitement montrés dans l'exemple suivant :

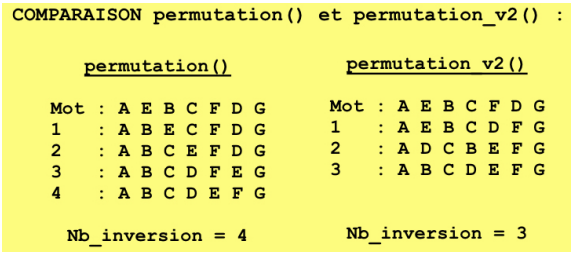

Figure 4: Comparaison des scénari de permutation() et de permutation_v2()

2.2.3 Construction de l'arbre

Q3 : Afin de trier la séquence, nous devons générer des séries d'inversions. Pour cela nous créons une fonction `nb_inversion()`, qui fait appel à la fonction `permutation()` ou `permutation_v2()`, selon le choix de l'utilisateur . La fonction `nb_inversion()` prend en argument la liste à trier. Elle calcul le score initial et appelle une première fois l'une des versions de permutation, les résultats retournés sont mémorisés dans une pile. Tant que cette pile n'est pas vide, `nb_inversion()` fait appel à permutation en prenant le premier élément de la pile en argument, soit une des séquence à retenir par permutation associée à son score courant et à sa distance à l'origine, qui est incrémentée d'une unité à chaque appel de la fonction permutation. Le premier élément de la pile, soit la séquence qui vient d'être calculée est supprimé et les résultats retournés par ce nouvel appel à permutation sont placés en tête. Ceci permet de continuer le trie de la séquence à partir d'une des proposition retenue. Le parcours d'une branche est terminé lorsque le score est équivalent à celui de la séquence triée. La distance à l'origine qui correspond au nombre d'inversion nécessaires pour trier une branche est retenue. À la fin du parcours de l'arbre on obtient une liste des distances. La fonction `nb_inversion()` retourne finalement la distance minimale.
Rappelons que si l'utilisateur choisi la seconde version de permutation le temps calcul sera fortement augmenté. Toutefois pour que celui-ci reste raisonnable le calcul d'une branche sera interrompu si la distance à l'orgine devient supérieure à `taille de seq + 1`.

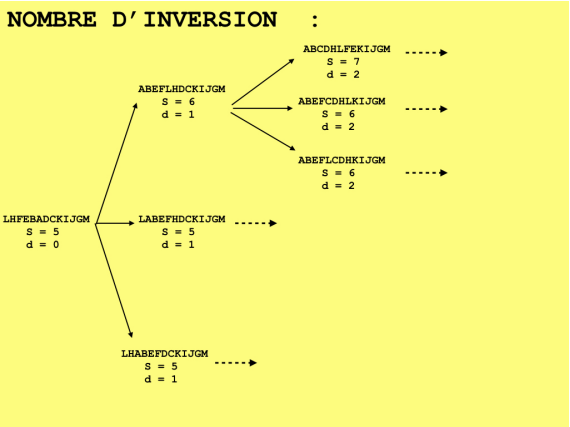

Figure 5: Schématisation de l'arbre des inversion

2.3 Génération d'échantillons aléatoires

Q4 : Afin d'évaluer si les nombres minimaux d'inversions observés, pour des chromosomes à 6, 7 et 13 gènes, pourraient être dus au hasard, nous avons répété le processus sur des séquences aléatoires. Pour ce faire nous avons créé une fonction nommée `scenario_aleatoire()`. Elle prend en entrée une liste, et un nombre d'itérations. Pour chaque itération, elle génère une version aléatoire de la liste donnée en entrée (cf :`seq_aleatoire()`) [1], puis elle fait appel à la fonction `nb_inversion()`, pour déterminer le nombre minimal d'inversions nécessaires pour trier cette séquence aléatoire. Les résultats de chaque itération sont conservés dans une liste. Cette liste peut être donnée en argument à la fonction `stat_parente()`, qui selon le nombre minimal d'inversions observées pour la séquence d'intérêt, retourne la probabilité d'observer un tel résultat sous l'hypothèse du hasard. La statistique générée correspond à la proportion de résultats aléatoires qui sont inférieurs au résultat de la séquence d'intérêt.

Q5 : Nous obtenons les résultats suivants:

1. Pour le chromosome III_R de taille 7, l'ordre des gènes de *Melanosgater* est `A E B C F D G`. Le nombre d'inversion minimal est de 3. La proportion de séquences aléatoires de longueur 7 obtenant un score inférieur est de 0.316, pour 500 tirages, sachant que la distance moyenne pour des séquences de longueur 7 est de 3.924.

2. Pour le chromosome III_L dont l'ordre des gènes est `C F E B A D`, le nombre minimal d'inversions obtenu est de 3. La proportion de séquences aléatoires de taille 6 ayant un score inférieur est de 0.17 (pour 500 tirages), sachant que la distance moyenne pour des séquences de longueur 6 est de 3.184

3. Pour le chromosome II_R dont l'ordre des gènes est `A C E B F D`, le nombre

[1] La fonction scenario_aleatoire() peut écrire les résultats dans un fichier texte, si l'option est activée. Ceci a été implémenté en raison du temps de calcul nécessaire pour les séquences de grandes tailles (> 13).

minimal d'inversions obtenu est de 4. La proportion de séquences aléatoires de taille 6 ayant un score inférieur est de 0.602 (pour 500 tirages).

4. Pour le chromosome II_L dont l'ordre des gènes est `D E F A C B`, le nombre minimal d'inversions obtenu est de 2. La proportion de séquences aléatoires de taille 6 ayant un score inférieur est de 0.048 (pour 500 tirages).

5. Pour le chromosome X dont l'ordre des gènes est `L H F E B A D C K I J G M`, le nombre minimal d'inversions obtenu est de 6. La proportion de séquences aléatoires de taille 13 ayant un score inférieur est de 0.015 (pour 66 tirages), sachant que la distance moyenne pour des séquences de longueur 13 est de 8.242.

Les nombres d'inversions minimaux ont été calculés grâce à l'utilisation de `permutation_v2()`, pour les séquences biologiques. Cependant en raison des temps de calcul, la fonction `permutation()` a été utilisée pour générer les résultats sur les séquences aléatoires. Il est ainsi probable que les proportions calculées soient quelque peu biaisées.
D'après ces résultats pour les chromosomes X et II_L la probabilité d'obtenir par hasard les nombres minimaux d'inversions observés, de respectivement 6 et 2, est faible < 0.05. Ainsi on peut affirmer qu'il existe un lien de parenté entre les deux espèces de drosophile *Pseudoobscura* et *D. Melanogaster*. Pour les chromosomes II_R, III_L et III_R, les nombres nombres minimaux d'inversions obtenus pourraient être dus au hasard, la structure de ces chromosomes est donc moins conservées.

3 Comparaisons de deux génomes bactériens complets

3.1 Comparaison des génomes médiéval et actuel

Nous avons comparé la taille des génomes de la souche actuelle (identifiant NC_003143.1 sur NCBI) et de la souche médiévale.
Le génome actuel possède 3551058 bases et elles sont toutes définies alors que le génome médiéval possède 3533579 bases dont 25769 ne sont pas définies. Le génome actuel est donc plus grand que le génome médiéval.

3.2 Dot matrix et filtrage des orthologies

Nous avons aligné les deux séquences sur BLAST et nous avons ainsi pu reproduire sur R la dot matrix correspondante (cf dot_matrix_vf.R) :

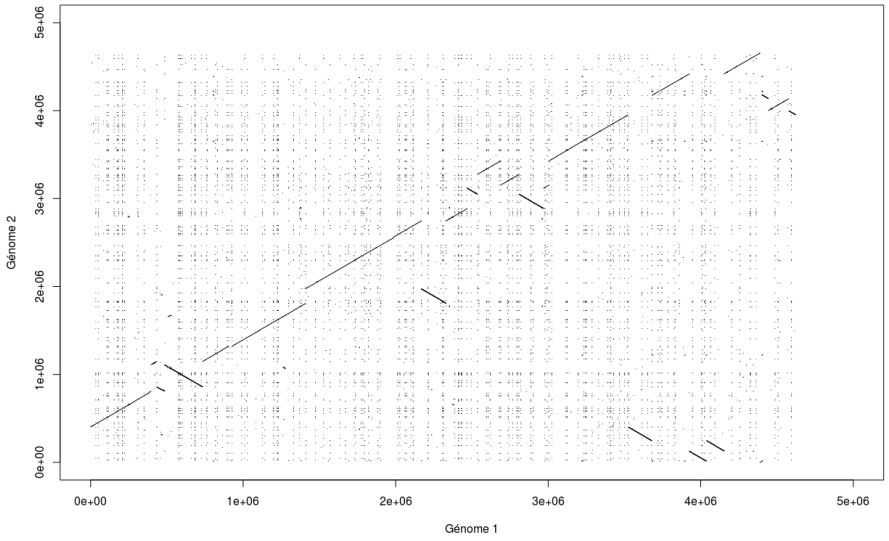

Figure 6: dot matrix des alignements des génomes médiéval et actuel

Nous avons ensuite **filtré** cette dot matrix pour conserver uniquement les lignes diagonales, correspondant aux blocs orthologues, et ainsi supprimer tous les points organisés en grille, qui correspondent aux répétitions. Pour ce faire, nous avons procédé par étapes:

1. Nous avons **remis les colonnes de début et de fin de fragments dans l'ordre** (*i.e.* nous nous sommes assurées que les coordonnées du début de chaque fragment étaient bien inférieures à celles de leur fin).

2. Nous avons effectué un **premier filtrage** de la dot matrix en supprimant tous les fragments de taille inférieure à 2000 nucléotides. Cette étape nous a permis à la fois de diminuer le temps de calcul dans les étapes de fusion et d'améliorer la précision des fusions (cf étape suivante).

3. Nous avons ensuite réalisé une **première fusion** des segments de la dot matrix. Pour cela, nous avons sélectionné, deux par deux, les fragments à la fois proches dans l'espace et qui, s'ils étaient fusionnés, ne modifiait pas la pente du segment non fusionné. Une fois sélectionnés, ces derniers sont fusionnés en formant un segment dont le début est celui du premier segment et la fin est celle du deuxième segment. Aussi, si nous n'avions pas effectué un premier filtrage des points correspondant aux répétitions, ils aurait pu fusionner entre eux ou avec des segments de la diagonale.

4. Nous avons ensuite **réitéré l'étape de fusion** 8 fois. Ce chiffre a été déterminé empiriquement en remarquant qu'effectuer plus d'étapes de fusions n'améliorait pas le résultat final.

5. Nous avons enfin effectué un **dernier filtrage** de la dot matrix en supprimant tous les fragments de taille inférieure à 20000 nucléotides. Cette valeur a été

déterminée empiriquement et représente un bon compromis entre lisibilité de la dot matrix et nombre de segments conservés.

Après toutes ces étapes, nous conservons **21 fragments** et nous avons ainsi obtenu la dot matrix suivante :

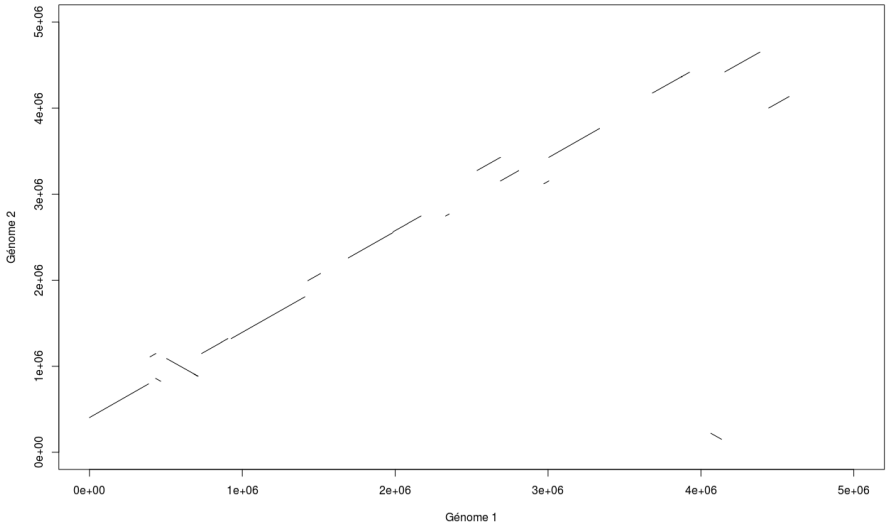

Figure 7: Dot matrix filtrée des alignements des génomes médiéval et actuel

Nous pouvons remarquer que, **par rapport à la dot matrix non filtrée (figure 6)**, comme attendu, les points correspondants aux répétitions ont été supprimés dans la **dot matrix filtrée (figure 7)**. En revanche, un certains nombre de segments à pente négative ont aussi été supprimés et certains segments à pente positive ont été écourtés. C'est le prix à payer pour avoir un schéma sans le "bruit" correspondant aux répétitions. En effet, bon nombre de ce qui apparaît comme des segments continus dans la dot matrix non filtrée sont en réalité une succession de segments très courts (et qui par conséquents vont être éliminés lors du filtrage) relativement proches les uns des autres.

Nous pouvons alors comparer le **taux de similarités** entre les alignements conservés (les segments diagonaux) et ceux non conservés (le "bruit" de fond) suite au filtrage, obtenus via la moyenne des taux de similarité de ces derniers. Pour ce faire, à chaque fusion de deux fragments, nous avons conservé la moyenne des taux de similarités de ces derniers. Nous obtenons ainsi un **taux de similarité** de 99.36 pour l'alignement conservé suite au filtrage et un taux de similarité de 97.75 pour l'alignement non conservé suite au filtrage. Le fait que le taux de similarité de l'alignement correspondant aux répétitions (non conservées lors du filtrage) est inférieur au taux de similarité des blocs orthologues (alignement conservé lors du filtarge) nous indique que **la dynamique de duplication dans le génome est plus ancienne que le moyen-âge**.

3.3 Réarrangements

Dans cette partie, nous cherchons à déterminer combien d'inversions séparent la souche médiévale de la souche actuelle. Pour ce faire, nous avons dans un premier temps déterminé les arcs, soit les gènes adjacents pour l'espèce de référence. Puis nous avons réitéré l'opération pour l'espèce d'intérêt.

Ces deux étapes nous ont permis de générer un graphe. Sur ce dernier, chaque noeud correspond à l'une des extrémité d'un segment orthologue, chaque arc relie des extrémités adjacentes. Ainsi chaque noeud possède deux voisins, l'un est associé à l'espèce de référence, l'autre celle d'intérêt. La fonction utilisée prend en argument la matrices des positions de début et de fin, de chaque segment orthologue. Le graphe obtenu à partir de nos données est celui ci-contre.

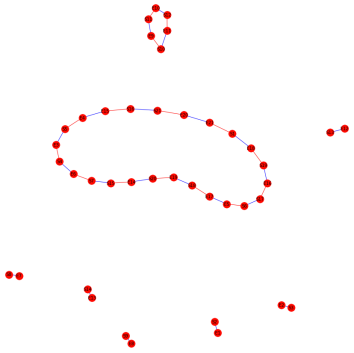

Figure 8: Graphe associé à notre jeu de données

Nous avons ensuite déterminé le nombre de cycles (c) (qui correspond au nombre de composantes connexes) et le nombre de marqueurs orthologues (n) de notre jeux de données. Cela nous a ensuite permis de déterminer le nombre d'inversions (d) associé grâce à la formule d=n-c. Pour notre jeu de donnée, nous obtenons d=13.

Enfin, nous voulions déterminer si le nombre d'inversions observé pourrait être dû au hasard.

Nous avons donc simulé des inversions aléatoires sur le même nombre de marqueurs, et nous avons comparé le 'd' observé dans le contexte biologiques et les 'd' obtenus par simulation.

Pour ce faire, nous avons utilisé une fonction qui permute les positions observés pour le génome de référence, et pour le génome d'intérêt. Cette fonction calcule ensuite le graphe correspondant et, à partir de celui-ci, infère le nombre d'inversions. Elle répète cette opération n fois et elle retourne la proportion de graphes aléatoirement générés qui permettent d'obtenir un nombre d'inversions inférieur à celui observé. En appliquant cette fonction à nos données et en donnant à n la valeur 500 (la fonction réalise donc 500 fois les étapes décrites ci-dessus), on obtient le résultat suivant :
La probabilité d'observer moins de 13 inversions pour 21 gènes orthologes est de 0.026 (après 500 simulations). Nous pouvons rejeter l'hypothèse du hasard au risque de 5%. Les deux organismes sont apparentés.

References

[1] AH Sturtevant and CC Tan. The comparative genetics ofdrosophila pseudoobscura andd. melanogaster. *Journal of genetics*, 34(3):415–432, 1937.