Recherche de motifs probabilistes : le cas des Matrices Poids Position dinucléotidiques (di-PWM)

Marie Mille




Rapport de stage

Recherche de motifs probabilistes : le cas des Matrices Poids Position dinucléotidiques (di-PWM)

Marie Mille

Master 1 « Sciences et Numériques pour la Santé »
Parcours « Bioinformatique, Connaissances, Données »

Tuteurs scientifiques : Bastien Cazaux, Julie Ripoll et Eric Rivals au LIRMM
(Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier)

Tuteur pédagogique : Sylvain Daudé

Juillet 2021

Table des matières

Remerciements

Je tiens à remercier sincèrement mes encadrants de stage Bastien Cazaux, Julie Ripoll et Eric Rivals de m'avoir offert la possibilité de faire ce stage avec eux. Je les remercie tout particulièrement pour leur disponibilité et leur adaptabilité durant cette période si particulière. Malgré l'alternance de présentiel et de distanciel, j'ai toujours reçu l'aide, le soutien mais aussi l'autonomie dont j'avais besoin pour progresser et je les remercie pour cela.

Merci à eux et à toute l'équipe MAB pour leur accueil durant ce stage.

Je remercie également le projet GEM Flagship financé par Labex NUMEV pour leur soutien.

1 Introduction

Avant de parler du contexte, des objectifs et des réalisations de mon stage, je vais présenter rapidement la structure d'accueil dans laquelle j'ai effectué celui-ci.

1.1 Structure d'accueil

Le stage de Master 1 du Master Sciences et Numérique pour la Santé, parcours Bioinformatique, Connaissances et Données est d'une durée d'un peu plus de trois mois. Il fait suite à un projet bibliographique mené au premier semestre. Mon stage s'est déroulé au Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM) au sein de l'équipe Méthodes et Algorithmes pour la Bioinformatique (MAB). Ses travaux portent sur différentes méthodes de calcul permettant de répondre à des questions biologiques. L'une des thématiques de recherche de l'équipe est l'algorithmique du texte et méthodes pour l'analyse du séquençage à haut débit. C'est dans ce groupe de travail que j'ai été intégrée pour mon stage. Ce groupe conçoit des algorithmes performants pour analyser les séquences biologiques à grande échelle. Ces algorithmes une fois implantés dans des logiciels sont diffusés au travers de la plateforme ATGC. La plateforme ATGC est une des plateformes fondatrices de l'Institut Français de Bioinformatique (IFB), auquel elle participe depuis son lancement en 2013.

Les chercheurs Bastien Cazaux, Julie Ripoll et Eric Rivals membres de ce groupe de travail, m'ont accompagnée durant l'étendue de mon rapport bibliographique et de mon stage. Ils ont partagé avec moi leur expérience et leur savoir en bioinformatique centrés sur l'analyse des séquences.

1.2 Contexte

La régulation de la transcription est un processus important dans la vie de la cellule. Des protéines spécialisées, appelées Facteurs de Transcription (FT), se lient sur de courtes et spécifiques séquences d'ADN pour réguler l'expression des gènes à proximité. Les séquences reconnues par un FT ne sont pas identiques, mais similaires. Afin de capturer la similarité de ces sites de liaison, différentes représentations existent et sont généralement appelées *motifs*. Pour les FT, les types de motif les plus répandus sont les Matrices Poids Position (*Position Weight Matrices : PWM*) aussi communément appelées Matrices de Poids Spécifiques à la Position (*Position-Specific Weight Matrices (PSWM)* ou Matrices de scores spécifiques à la position (*Position-Specific Scoring Matrices (PSSM)*).

Une PWM est construite à partir d'un alignement multiple de séquences observées et retranscrit la variation observée des nucléotides à différentes positions. Plusieurs bases de données (JASPAR, TRANSFAC, etc.) rassemblent les PWM pour les FT connus. Ces matrices sont utilisées pour scanner des séquences d'ADN et déterminer de possibles sites de liaison et éventuellement annoter ces séquences. Dans le cas de génomes complets, le temps de recherche de plusieurs PWM peut être important [5, 3].

Les PWM présument que les positions de la séquence de liaison sont indépendantes les unes des autres. Cependant, différentes études ont montré qu'une mutation

à une position donnée influence la probabilité de mutation des positions voisines. Pour dépasser cette limitation, Kulakovskiy et al. [2] ont proposé un type de motif plus complexe, appelé di-PWM, qui modélise la fréquence d'occurrences de dinucléotides du site de liaison, alors que les PWM modélisent des mononucléotides. Cela permet de modéliser la relation entre nucléotides voisins. Les positions de la séquence du site de liaison ne sont plus traitées comme indépendantes. Leurs études montrent que les di-PWM améliorent la sensibilité comparées aux PWM, et produisent ainsi moins de faux-négatifs lors du balayage d'une séquence. La base de données HOCOMOCO fournit des di-PWM pour les FT chez l'homme et la souris.

Même s'il existe d'autres types de motifs plus complexes tels que les Modèles de Markov Cachés ou les modèles de covariance comme nous l'avions vu lors du projet bibliographique, pour ce projet, nous nous concentrons sur les di-PWM.

1.3 Objectifs du stage

À l'heure actuelle, il existe un seul outil de recherche de di-PWM appelé SPRYSARUS (github), basé sur l'utilisation d'un super alphabet et implémenté en JAVA. Le principe d'un super alphabet est de transformer l'alphabet en q-mers, permettant ainsi une filtration plus rapide [4].

Notre objectif est de créer de nouveaux algorithmes permettant d'effectuer cette recherche, de les implémenter, de les tester et de les rendre disponibles pour tous sous forme d'un module Python3.

Différentes stratégies permettent de rechercher des motifs sur une séquence d'ADN. Nous avons choisi de développer des algorithmes de recherche par fenêtre glissante ainsi qu'une approche par énumération de mots suivie d'une recherche de cet ensemble de mots dans la séquence d'ADN. Pour tester nos implémentations, nous avons utilisé les di-PWM de FT humains de la base de données HOCOMOCO.

1.4 Réalisations

Durant ce stage, j'ai :

- adapté aux di-PWM une stratégie d'évaluation fréquemment utilisée grâce à l'utilisation d'une *LookAheadTable(LAT)*, cf. section 2.2.1.
- développé une nouvelle structure permettant l'évaluation du score des di-PWM, la *LookAheadMatrix(LAM)*, cf. section 2.2.2.
- décrit et prouvé les propriétés de la *LAT* et de la *LAM*, cf. section 2.2.4.
- développé et implémenté un algorithme de recherche des di-PWM par fenêtre glissante, cf. section 3.3.
- développé et implémenté une nouvelle stratégie de recherche des di-PWM basée sur l'énumération de mots que l'on recherche ensuite dans la séquence d'ADN, cf. section 4.
- étudié la complexité des deux types d'algorithmes aux sections 3.3 et 4.3 et analysé les résultats à la section 6.

L'ensemble du travail décrit dans ce mémoire est le résultat de mon travail de stage ; les algorithmes et résultats présentés sont issus de ce travail (même si je me suis

bien sûr appuyé sur des notions bien connues dans la littérature comme la recherche par fenêtre glissante, ou l'idée de la LookAheadTable qui est déjà mentionné dans l'article de Beckstette et al. 2006 comme faisant partie du « folklore »).

Dans le reste de ce mémoire, j'utilise le pronom nominatif « nous » plutôt que « je », car cela reflête mieux la réalité du travail de recherche.

2 Définition du problème et notations

Avant de définir formellement notre problème, nous déterminons les notations utilisées dans ce rapport.

2.1 Notations

Il est important de préciser que nous présentons dans ce rapport un alphabet quelconque, mais dans la pratique, nous travaillons sur de l'ADN et notre alphabet est de taille 4.

— Pour un alphabet Σ de taille σ, une di-PWM P représentant un motif de taille m est une matrice de taille $\sigma^2 \times (m-1)$. Le poids du dinucléotide db où d et b appartiennent à Σ aux positions respectives i et $i+1$ sera alors noté $P[db, i]$. Il correspond au *log* de la fréquence observée du dinucléotide à la position donnée rapportée à sa fréquence attendue(1/16). La figure 1 est une représentation d'un motif de taille 6 et ayant pour alphabet A, C, G, T.

	0	1	2	3	4
AA	0.77	-0.14	-0.14	-1.97	-1.77
AC	-0.78	-1.97	-2.58	-2.58	-2.23
AG	-0.65	0.50	1.08	-4.4	-1.45
AT	-1.77	-1.2	-4.4	1.27	-4.4
CA	0.52	0.26	-1.11	-3.18	0.64
CC	-1.77	-0.43	-3.12	-0.56	-1.6
CG	-1.32	0.94	0.31	-4.4	-0.14
CT	-3.12	-0.22	-4.4	-1.77	1.97
GA	1.48	0.16	0.82	-1.45	-3.12
GC	0.33	-0.43	-2.23	-1.97	-4.4
GG	1.28	0.85	1.83	-1.97	-1.97
GT	-1.02	-1.32	-2.23	2.42	0.16
TA	-1.21	-0.59	-1.11	-4.4	0.61
TC	-2.23	-1.11	-4.4	-4.4	2.14
TG	-1.45	0.78	0.19	-4.4	-0.54
TT	-2.23	-1.45	-4.4	-2.23	0.16

di-PWM

FIGURE 1 – Di-PWM représentant un motif de taille 6 et ayant pour alphabet A, T, G et C. Les colonnes correspondent aux positions dans le motif et les lignes aux dinucléotides. Une valeur représente le poids d'un dinucléotide à une position donnée. Cette di-PWM est une di-PWM factice inspirée de la di-PWM du FT ATF3 réduite en longueur et dont les positions ont été modifiées.

— Le seuil de score fixé comme paramètre de la recherche est noté θ.

— Le texte dans lequel on effectue la recherche est noté T et sa longueur n.

— Pour un mot $u = u_0 \dots u_l$ et $0 \leq i < j \leq l$, on note $u[i, j]$ la sous-chaîne de u commençant à la position i et se terminant à la position j.

— Pour un mot de longueur m, $u = u_0 \dots u_{m-1}$, un préfixe de u est une sous-chaîne de u ayant pour premier élément u_0 et de longueur strictement inférieur à m. Un suffixe de u est une sous-chaîne de u commençant à une position strictement supérieure à 0 et ayant pour dernière position u_{m-1}.

— On note Σ^p où $p \in \mathbb{N}$ l'ensemble des mots possibles de longueur p formés sur l'ensemble de l'alphabet Σ.

Avec un motif probabiliste de type PWM ou di-PWM, pour un mot u de longueur m on peut calculer le score de u selon P. Le score est déterminé par la somme des valeurs pour chaque dinucléotide de u en fonction de leur position.

Calcul du score Le score d'un mot u de taille m pour une di-PWM P de taille $m-1$, noté $score(u)$ est calculé de la façon suivante :

$$score(u) = \sum_{i \in \{0, m-2\}} P[\ u[i, i+1], i]$$

Le calcul de ce score est une somme et celle-ci étant commutative, il peut se faire dans un ordre différent. On peut le calculer indifféremment de gauche à droite ou de droite à gauche, ou en changeant l'ordre des positions.

Calcul du seuil θ en fonction du ratio seuil donné On note $score_{max}$ et $score_{min}$ les scores maximum et minimum d'une di-PWM et $ratio$ le ratio du seuil donné en entrée.

$$\theta = (score_{max} - score_{min}) * ratio + score_{min}$$

— On définit une occurrence comme une sous-chaîne de T dont le score est supérieur ou égal à θ.

Notre problème se définit de la manière suivante. Nous cherchons la position et le score des occurrences du motif P dans le texte T.

2.2 Calcul des majorants

Nous utilisons dans nos différents algorithmes une estimation d'un score maximum atteignable pour un mot de taille m. Ce score est calculé à partir du score du préfixe du mot qu'on nomme score partiel auquel on ajoute un majorant du suffixe de ce mot. Afin de calculer le score maximum atteignable pour un mot, nous pouvons utiliser l'une des deux structures distinctes que sont la *LookAheadTable(LAT)* et la *LookAheadMatrix(LAM)*. La figure 2 présente ce principe de manière générale.

Les deux structures se construisent de manière semblable. Les deux algorithmes de construction commencent par calculer le(s) dernier(s) élément(s) de la structure

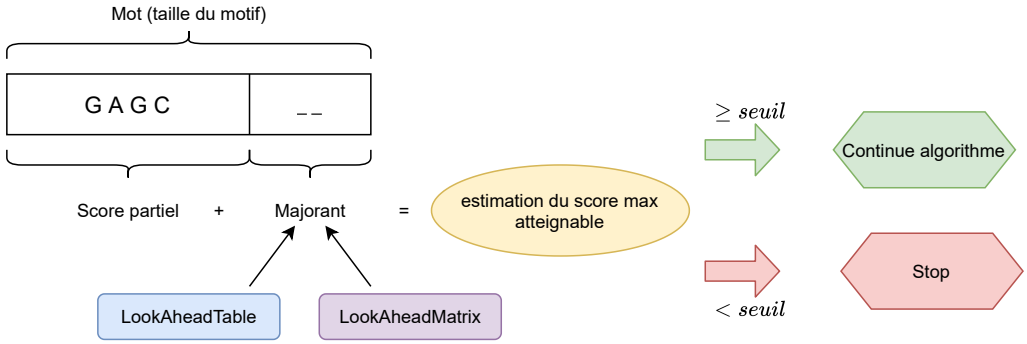


FIGURE 2 – Estimation du score maximum atteignable pour un mot de taille m à partir du score partiel de son préfixe. Le score du préfixe est calculé (score partiel) et sommé au majorant calculé à partir de la LAT ou de la LAM. Le score maximum atteignable pour un mot complet à partir de ce préfixe est évalué en comparaison au seuil fixé afin de déterminer si l'algorithme doit continuer ou non.

en partant de la fin du motif. Puis, ils avancent vers le début du motif en calculant les éléments de manière récursive. La dernière position est traitée indépendemment, par la suite, les positions suivantes dans l'ordre du traitement, dépendent du calcul précédent.

Nous détaillons ensuite la définition et l'algorithme de construction de la LAT puis ceux de la LAM.

2.2.1 La *LookAheadTable*

La LAT d'une di-PWM est une table qui va stocker le majorant que le score d'un suffixe peut atteindre à partir d'une position donnée. La définition se trouve en définition 2.1.

Definition 2.1 (*LookAheadTable* associée à une di-PWM)**.**
La LAT L d'une di-PWM P d'un motif de taille m est une table de taille $(m-1)$ où pour tout entier i tel que $0 \leq i \leq m-2$

$$L[i] := \begin{cases} \max\limits_{d,b \in \Sigma} P[db, i], & \text{si } i = m-2 \\ \max\limits_{d,b \in \Sigma} P[db, i] + L[i+1], & \text{si } 0 \leq i < m-2 \end{cases}$$

Ici db représente le dinucléotide constitué de la paire de symboles d et b.

$L[i]$ est un majorant du score d'un suffixe de longueur $(m-1-i)$. L'algorithme 1 calcule la LAT d'un di-PWM. Comme la calcul de $L[i]$ dépend de $L[i+1]$, le calcul de la LAT se fait de droite à gauche dans la di-PWM (cf. la boucle ligne 8 algorithme 1).

2.2.2 La *LookAheadMatrix*

La LAM d'une di-PWM est une matrice qui va stocker le score maximum qu'un suffixe peut atteindre à partir d'une position donnée et commençant par une lettre de l'alphabet spécifique, cf. définition 2.2.

Algorithme 1 : MakeLookAheadTable

Input : Alphabet Σ of size σ,di-PWM matrix P of size $\sigma^2 \times (m-1)$, for a motif of length m

Output : LookAheadTable L of size $(m-1)$

```
1  L ← initialized with −∞
2  max ← −∞
3  for d ∈ {0, ..., σ − 1} do
4      for b ∈ {0, ..., σ − 1} do
5          score ← P[db, m − 2]
6          if score > max then max ← score
7  L[m − 2] ← max
8  for i ∈ {m − 3, ..., 0} do
9      max ← −∞
10     for d ∈ {0, ..., σ − 1} do
11         for b ∈ {0, ..., σ − 1} do
12             score ← P[db, i] + L[i + 1]
13             if score > max then max ← score
14     L[i] ← max
15 return L
```

Definition 2.2 (*LookAheadMatrix* for di-PWM)**.**

La *LookAheadMatrix* M pour une di-PWM P d'un motif de taille m et un alphabet Σ de taille σ est une matrice de taille $\sigma \times (m-1)$ où pour tout i tel que $0 \leq i \leq m-2$ et pour tout $d \in \Sigma$:

$$M[d,i] := \begin{cases} \max\limits_{b \in \Sigma} P[db,i], & \text{if } i = m-2 \\ \max\limits_{b \in \Sigma}(P[db,i] + M[b,i+1]), & \text{if } 0 \leq i < m-2 \end{cases}$$

$M[d,i]$ est donc le score maximum d'un suffixe de longueur $(m-1-i)$ et commençant par la lettre d. L'algorithme 2 calcule la LAM d'un di-PWM.

2.2.3 Exemples

Un exemple de LAT et de LAM d'une di-PWM est présenté à la figure 3.

Si on se base sur cet exemple de di-PWM et sur la LAT et la LAM qui en découlent, on peut illustrer un exemple concret d'estimation du score maximum atteignable. Prenons le préfixe AGT, il a pour score :

$$score(AGT) = -0.75 - 1.32 = -2.07.$$

Pour un suffixe commençant à la position 3, la valeur du majorant est 4.56 avec la LAT et -0.09 avec la LAM.

$$score_{LAT} = -2.07 + 4.56 = 2.49$$

Algorithme 2 : MakeLookAheadMatrix

Input : Alphabet Σ of size σ,di-PWM matrix P of size $\sigma^2 \times (m-1)$, for a motif of length m

Output : LookAheadMatrix M of size $\sigma \times (m-1)$

$M \leftarrow$ initialized with $-\infty$
for $d \in \{0, \ldots, \sigma - 1\}$ **do**
    $max \leftarrow -\infty$
    for $b \in \{0, \ldots, \sigma - 1\}$ **do**
        $score \leftarrow P[db, m-2]$
        if $score > max$ **then** $max \leftarrow score$
    $M[d, m-2] \leftarrow max$
for $i \in \{m-3, \ldots, 0\}$ **do**
    for $d \in \{0, \ldots, \sigma - 1\}$ **do**
        $max \leftarrow -\infty$
        for $b \in \{0, \ldots, \sigma - 1\}$ **do**
            $score \leftarrow P[db, i] + M[b, i+1]$
            if $score > max$ **then** $max \leftarrow score$
        $M[d, i] \leftarrow max$
return M

	0	1	2	3	4
AA	0.77	-0.14	-0.14	-1.97	-1.77
AC	-0.78	-1.97	-2.58	-2.58	-2.23
AG	-0.65	0.50	1.08	-4.4	-1.45
AT	-1.77	-1.2	-4.4	1.27	-4.4
CA	0.52	0.26	-1.11	-3.18	0.64
CC	-1.77	-0.43	-3.12	-0.56	-1.6
CG	-1.32	0.94	0.31	-4.4	-0.14
CT	-3.12	-0.22	-4.4	-1.77	1.97
GA	1.48	0.16	0.82	-1.45	-3.12
GC	0.33	-0.43	-2.23	-1.97	-4.4
GG	1.28	0.85	1.83	-1.97	-1.97
GT	-1.02	-1.32	-2.23	2.42	0.16
TA	-1.21	-0.59	-1.11	-4.4	0.61
TC	-2.23	-1.11	-4.4	-4.4	2.14
TG	-1.45	0.78	0.19	-4.4	-0.54
TT	-2.23	-1.45	-4.4	-2.23	0.16

di-PWM

LookAheadTable

Position de départ du suffixe	0	1	2	3	4
Score max atteignable	8.81	7.33	6.39	4.56	2.14

LookAheadMatrix

Position de départ du suffixe commançant par	0	1	2	3	4
A	7.66	6.89	5.64	3.41	-1.45
C	7.41	7.33	4.87	1.41	1.97
G	8.52	7.24	6.39	4.56	0.16
T	5.79	7.17	4.75	-0.09	2.14

FIGURE 3 – Exemple de LAT et de LAM construites à partir d'une di-PWM donnée, qui est celle de la figure 1. Les deux structures sont de même longueur correspondant à la taille de la di-PWM. Pour la LAT, on a pour une colonne i une estimation du score maximum atteignable d'un suffixe commençant à la position i. Pour la LAM, on obtient pour une colonne i et une ligne r, l'estimation du score maximum atteignable pour un suffixe commençant à la position i par le nucléotide r.

et

$$score_{LAM} = -2.07 - 0.09 = -2.16.$$

L'estimation du score maximum atteignable est totalement différente selon si on utilise la LAT ou la LAM. Si $\theta = 2$, alors l'algorithme sera stoppé avec la LAM, mais pas avec la LAT.

2.2.4 Propriétés

LAT et LAM servent de majorants pour estimer le score complet à partir d'un score partiel d'une fenêtre. Si le score complet estimé est trop bas, plus besoin de continuer à évaluer le score de la fenêtre courante : on gagne du temps en passant à la fenêtre suivante. Il est donc intéressant de déterminer laquelle de LAT ou LAM donne une meilleure estimation, ou permet de gagner le plus de temps.

Propriété 2.1. Propriétés de la *LookAheadTable* L et de la *LookAheadMatrix* M :

1. $\forall i \in \{0, \dots, m-2\}$ et $\forall d \in \Sigma$
   $L[i] \geq M[d, i]$
2. $\forall i \in \{0, \dots, m-3\}$ et $\forall\ u = u_0 \dots u_{i+1} \dots u_{m-1} \in \Sigma^m$:
   $score(u_0 \dots u_{i+1}) + L[i+1] \geq score(u_0 \dots u_{i+1}) + M[u_{i+1}, i+1]$
3. Soit $i \in \{0, \dots, m-3\}$ et u un mot. On définit $A_L \subset \mathbb{N}$ l'ensemble des positions i d'une fenêtre telles que : $score(u_0 \dots u_{i+1}) + L[i+1] \geq \theta$. De même, on définit $A_M \subset \mathbb{N}$ l'ensemble des positions i d'une fenêtre telles que : $score(u_0 \dots u_{i+1}) + M[u_{i+1}, i+1] \geq \theta$. Alors on a : $A_M \subseteq A_L$
4. $\forall i \in \{0, \dots, m-2\}$ et $\forall\ u = u_0 \dots u_i \in \Sigma^{i+1}$: il existe $v = u_0 \dots u_i v_{i+1} \dots v_{m-1} \in \Sigma^m$, tel que $score(v) = score(u) + M[u_i, i]$

Démonstration. **Propriété 1 :** On veut montrer que $\forall i \in \{0, \dots, m-2\}$ et $\forall d \in \Sigma$, $L[i] \geq M[d, i]$. On va procéder par récurrence sur i :

• Pour $i = m-2$, sachant que

$$L[i] = \max_{\mathbf{d,b \in \Sigma}} P[db, i] \text{ et } M[d, i] = \max_{\mathbf{b \in \Sigma}} P[db, i],$$

on veut montrer que $\forall d \in \Sigma$, $\max\limits_{\mathbf{d,b \in \Sigma}} P[db, i] \geq \max\limits_{\mathbf{b \in \Sigma}} P[db, i]$.

Comme $\{P[db, i] \mid d, b \in \Sigma\} = \bigcup\limits_{d \in \Sigma} \{P[db, i] \mid b \in \Sigma\}$, on a

$$\forall d \in \Sigma, \max\{P[db, i] \mid d, b \in \Sigma\} \geq \max\{P[db, i] \mid b \in \Sigma\}.$$

• Pour $i < m-2$, on suppose que $\forall d \in \Sigma, M[d, i+1] \leq L[i+1]$. Soit $d \in \Sigma$, on veut montrer que $M[d, i] \leq L[i]$, i.e.

$$\max_{b \in \Sigma}(P[db, i] + M[b, i+1]) \leq \max_{b,d \in \Sigma}(P[db, i] + L[i+1])$$

Par la définition du maximum, on a

$$\max_{b \in \Sigma}(P[db, i] + M[b, i+1]) \leq \max_{b \in \Sigma} P[db, i] + \max_{b \in \Sigma} M[b, i+1].$$

De plus, similairement à la preuve pour $i = m-2$, $\max_{b\in\Sigma}(P[db, i]) \leq \max_{d,b\in\Sigma}(P[db, i])$.

Enfin, selon notre hypothèse de récurrence, $\forall d \in \Sigma : M[d, i+1] \leq L[i+1]$ et comme $\max_{b\in\Sigma} M[b, i+1] \in \{M[d, i+1] \mid d \in \Sigma\}$, on obtient alors $\max_{b\in\Sigma} M[b, i+1] \leq L[i+1]$.

On a donc : $\max_{b\in\Sigma}(P[db, i] + M[b, i+1]) \leq \max_{b,d\in\Sigma}(P[db, i] + L[i+1])$.

Propriété 2 : On va montrer que $\forall i \in \{0, \ldots, m-3\}$ et $\forall u = u_0 \ldots u_{i+1} \ldots u_{m-1} \in \Sigma^m$:

$$score(u_0 \ldots u_{i+1}) + L[i+1] \geq score(u_0 \ldots u_{i+1}) + M[u_{i+1}, i+1].$$

Sachant que $\forall i \in \{0, \ldots, m-2\}$ et $d \in \Sigma$, on a $L[i] \geq M[d, i]$, on a donc $\forall i \in \{0, \ldots, m-3\}$ et $\forall u = u_0 \ldots u_{i+1} \ldots u_{m-1} \in \Sigma^m$, $L[i+1] \geq M[u_{i+1}, i+1]$ et donc

$$score(u_0 \ldots u_{i+1}) + L[i+1] \geq score(u_0 \ldots u_{i+1}) + M[u_{i+1}, i+1].$$

Propriété 3 : Pour prouver que $A_M \subseteq A_L$, on prend $x \in A_M$ et on veut montrer que $x \in A_L$.

Comme $x \in A_M$ on a $score(u_0 \ldots u_x) + M[u_x, x] \geq \theta$, or on a montré que $score(u_0 \ldots u_x) + L[x] \geq score(u_o \ldots u_x) + M[u_x, x]$, donc $score(u_0 \ldots u_x) + L[x] \geq \theta$, et donc $x \in A_L$.

Propriété 4 : On veut montrer que $\forall i \in \{0, \ldots, m-2\}$ et $\forall\ u = u_0 \ldots u_i \in \Sigma^{i+1}$: il existe $v = u_0 \ldots u_i v_{i+1} \ldots v_{m-1} \in \Sigma^m$, tel que $score(v) = score(u) + M[u_i, i]$. On va procéder par récurrence sur i.

• Pour $i = m-2$. Soit $u \in \Sigma^{i+1}$. On veut montrer que $\exists v \in \Sigma^m$ tel que $score(v) = score(u_0 \ldots u_{m-2}) + M[u_{m-2}, m-2]$.
Comme $M[u_{m-2}, m-2] = \max_{b\in\Sigma}(P[db, m-2])$, on prend

$$b' \in \operatorname*{argmax}_{b\in\Sigma} P[db, m-2]$$

et on pose $v = u_0 \ldots u_{m-2}b'$.

On a donc $score(v) = score(u_0 \ldots u_{m-2}) + score(u_{m-2}, b')$, et $score(u_{m-2}b') = P[u_{m-2}b', m-2] = M[u_{m-2}, m-2]$ car $b' \in \operatorname*{argmax}_{b\in\Sigma} P[db, m-2]$, on a donc $score(v) = score(u_0 \ldots u_{m-2}) + M[u_{m-2}, m-2]$.

• Soient $i < m-2$ et $w \in \Sigma^{i+1}$; on suppose que $\forall u \in \Sigma^{i+2}$, $\exists v \in \Sigma^m$ tel que $score(v) = score(u_0 \ldots u_{m-2}) + M[u_{m-2}, m-2]$.

On a par définition $M[w_i, i] = \max_{b\in\Sigma}(P[w_ib, i] + M[b, i+1])$, on prend

$$b' \in \operatorname*{argmax}_{b\in\Sigma}(P[w_ib, i] + M[b, i+1])$$

et $u' = w_0 \ldots w_ib'$. Par l'hypothèse de récurrence, il existe $v \in \Sigma^m$ tel que $score(v) = score(u') + M[b', i+1]$.

Or $score(u') = score(w) + P[w_ib']$, donc $score(v) = score(w) + P[w_ib'] + M[b', i+1]$. Comme $P[w_ib'] + M[b', i+1] = M[w_i, i]$, on a donc $score(v) = score(w) + M[w_i, i]$.

□

Les propriétés 1 et 2 décrites précédemment montrent que le score maximum atteignable à une position donnée calculé à partir de la LAT est plus grand ou égal à celui obtenu grâce à la LAM. Par conséquent, comme montré par la propriété 3, le nombre de positions où le score maximum atteignable dépasse le seuil est plus important avec la LAT qu'avec la LAM. Cela indique que la LAT est moins sélective que la LAM. Cela repose sur le fait que les valeurs de la LAT sont des majorants alors que celles de la LAM sont des bornes supérieures. Ces dernières sont des valeurs réelles existantes comme décrit dans la propriété 4.

3 Recherche par fenêtre glissante

Dans cette section, nous allons présenter plus en détail la première stratégie que nous avons exploré : la recherche par fenêtre glissante. Après avoir décrit la recherche naïve, qui consiste à calculer le score de chaque fenêtre possible, nous montrons comment améliorer cette recherche en utilisant la LAT et la LAM.

3.1 Recherche naïve

L'algorithme de recherche naïve de motif dans un texte est un algorithme de recherche par fenêtre glissante. Chaque sous-chaîne de T de longueur m est potentiellement une occurrence dont le score est supérieur ou égal à θ. T contient $n-m+1$ sous-chaînes de longueur m, qu'on nomme fenêtre. Dans le cas d'une recherche naïve, on considère chaque fenêtre à la position i, pour i compris entre 0 et $n - m + 1$. Pour chaque fenêtre, on calcule son score en parcourant toutes les positions de la fenêtre et en sommant le score de chaque position (boucle for interne). La figure 4 en montre le principe. Si le score de la fenêtre est supérieur ou égal au seuil θ, alors, on imprime la position et le score de celle-ci.

3.2 Recherche semi-naïve

L'algorithme de recherche semi-naïve possède la même structure que l'algorithme naïf. Chaque fenêtre de la séquence est traitée. Ce qui diffère est l'évaluation de chacune de ses fenêtres. Pour chaque préfixe on calcule un score partiel auquel on ajoute un majorant du score du suffixe. On obtient l'estimation du score maximum atteignable pour cette fenêtre à partir de ce préfixe. Si l'estimation du score maximum atteignable n'atteint pas le seuil, le calcul de cette fenêtre est abandonné. Afin de calculer les majorants des « sous-fenêtres », nous utilisons deux structures distinctes : la LAT et la LAM.

3.3 Algorithmes de recherche semi-naïfs

Deux algorithmes différents de recherche peuvent donc être employés selon que l'on choisit d'utiliser la LAT ou la LAM. Ils sont décrits dans l'algorithme 3.

Les deux algorithmes présentés précédemment ont une structure identique. Les variations sont aux lignes 1a, 1b, 8a et 8b. En ce qui concerne les lignes 1a et 1b, il s'agit de l'initialisation de la structure d'évaluation des majorants qui varie en

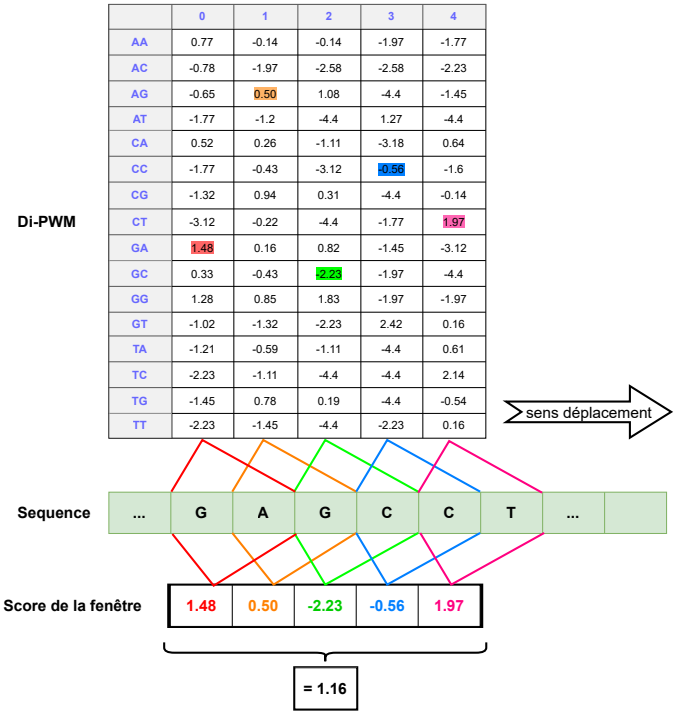


FIGURE 4 – Recherche naïve par fenêtre glissante d'une di-PWM sur une séquence d'ADN. On regarde sur la séquence quel est le dinucléotide correspondant aux positions i et $i+1$ et on recherche son score dans la colonne i de la di-PWM. On fait de même pour chaque position-1 d'une fenêtre de longueur m. Les scores de chaque dinucléotide sont sommés afin d'obtenir le score total de la fenêtre. Ce calcul est effectué pour la fenêtre suivante commençant un nucléotide plus loin que la précédente, et ainsi de suite, sur la totalité de la séquence.

fonction de celle que l'on choisit d'utiliser. Quant aux lignes 8a et 8b, la différence est dans la manière d'accéder aux valeurs de ces majorants selon si on a une table ou une matrice.

Notons que la complexité en temps dans le pire des cas de ces deux algorithmes de recherche est similaire et qu'elle est de l'ordre de $O(nm)$ où m est la taille du motif et n la taille de la séquence dans laquelle on effectue la recherche.

3.4 Synthèse et variantes

Pour la recherche de motifs par fenêtre glissante, nous avons donc deux algorithmes semis-naïfs au fonctionnement similaire. Leur spécificité réside dans l'usage d'une structure différente pour le calcul des majorants à ajouter aux scores partiels des fenêtres. Après avoir montré leurs propriétés théoriques de ces structures, nous étudions l'efficacité en pratique des algorithmes dans la section 6.

Nous avons également développé une variante connue de la recherche de mots. Il s'agit de traiter la fenêtre dans le sens opposé. En effet, au lieu de lire le motif de gauche à droite, on peut le lire de droite à gauche. L'algorithme a le même fonctionnement, simplement le calcul de la structure des majorants se fait dans l'autre sens, et il en est de même pour la lecture de la fenêtre. Cette approche

Algorithme 3 : Procedure semi-naive di-PWM with LookAheadTable or LookAheadMatrix

Input : Alphabet Σ of size σ, di-PWM matrix P of size $\sigma^2 \times (m-1)$ for a motif of length m, sequence T of length n, θ score threshold

Output : All matching positions of P in T and their score

```
1a  L ← Table of size (m − 1)
1b  M ← Matrix of size σ × (m − 1)
 2  for i ∈ {0, . . . , n − (m − 1)} do
 3      score ← 0
 4      for j ∈ {0, . . . , (m − 2)} do
 5          d ← x[i + j]
 6          b ← x[i + j + 1]
 7          score ← score + P[db, j]
8a          if j < m − 2  AND  score + L[j + 1] < θ then
8b          if j < m − 2  AND  score + M[b, j + 1] < θ then
 9              Break
10          else
11              if j = m − 2  AND  score ≥ θ then
12                  Report ⟨i, score⟩
```

est intéressante dans certains cas, selon les spécificités du motif. Ces effets sont également traités dans la section 6.4.2.

4 Énumération de mots

Une autre approche utilisée pour la recherche de motifs probabilistes est de segmenter le processus en deux étapes distinctes. La première étape consiste à énumérer les mots valides, c'est à dire les mots de taille m dont le score est supérieur ou égal à Θ. Ensuite, on effectue une recherche exacte des mots valides dans un texte. Le logiciel MOTIF implante cette stratégie en deux étapes et propose un algorithme d'énumération des mots valides pour une PWM et pour un seuil donné [3]. Cet algorithme adopte une approche de programmation par contraintes. Le principe de notre stratégie d'énumération pour une di-PWM est illustré par la figure 5.

4.1 Question

Pour une raison combinatoire, il est très coûteux de générer tous les mots possibles de longueur m pour un alphabet de taille σ : en effet, cela correspond à σ^m possibilités. Pour les di-PWM de HOCOMOCO, l'alphabet utilisé est de taille 4 et la taille des motifs varie entre 10 et 30 nucléotides. Il est donc déraisonnable de calculer le score de tous les mots possibles, pour ensuite ne conserver que ceux satisfaisant la condition de score voulue. Notre stratégie consiste à ne générer en entier que les mots satisfaisant la condition de score, mots que nous appelons *mots valides*.

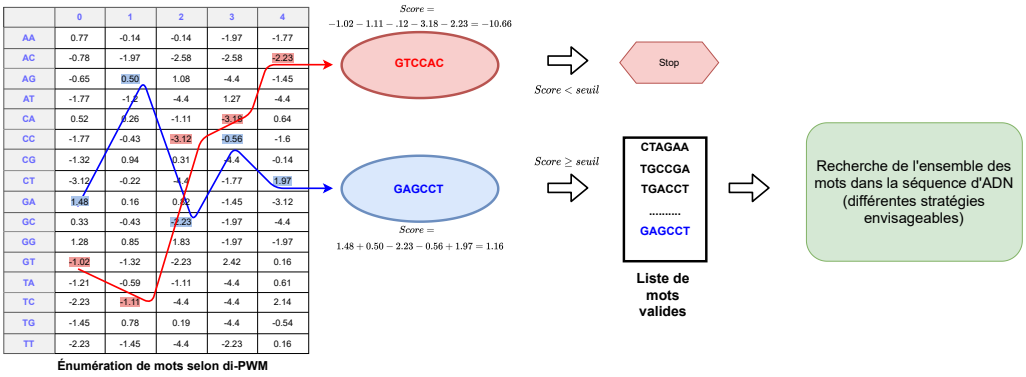


FIGURE 5 – Principe de la stratégie d'énumération de mots à partir d'une di-PWM. Les mots valides sont les mots énumérés de taille m dont le score calculé selon la di-PWM est supérieur ou égal au seuil fixé. Ces mots sont ensuite recherchés dans la séquence d'ADN.

Il s'agit donc pour une matrice di-PWM P et un seuil θ, d'énumérer tous les mots dont le score est supérieur ou égal à θ dans P, ainsi que leurs scores.

4.2 Stratégie adoptée

Nous avons choisi pour répondre à notre question de nous baser sur la simulation d'un *trie* et d'utiliser le paradigme algorithmique de séparation et évaluation (*branch-and-bound*).

4.2.1 *Trie*

La stratégie est de générer les mots dans l'ordre lexicographique et de couper après certains préfixes ; pour cette raison, nous avons choisi de simuler d'un *trie*. Le *trie*, aussi appelé arbre des préfixes, est une structure de données qui a la forme d'un arbre enraciné. Il a pour caractéristique que chaque descendant d'un même nœud possède le même préfixe. On retrouve les mots en traversant l'arbre en suivant les branches en allant jusqu'aux feuilles.

4.2.2 Méthode de *branch-and-bound* ou par séparation et évaluation

Le principe d'un algorithme par séparation et évaluation [1]est d'éviter de générer toutes les solutions possibles. Le but est de diviser le problème en sous-problèmes qui seront évalués afin de ne considérer que ceux susceptibles d'apporter une solution. Le principe appliqué à notre situation est schématisé dans la figure 6.

Dans notre cas, nous utilisons la LAT ou la LAM afin de procéder à cette évaluation. Pour chaque nœud, on simule l'ajout d'un nucléotide (parmi notre alphabet) et on détermine si le score partiel de ce préfixe sommé au majorant, c'est-à-dire le score maximum atteignable, peut atteindre le seuil θ. De cette condition dépendra la création d'un nœud descendant ou non.

1. Wikipedia : Séparation et évaluation

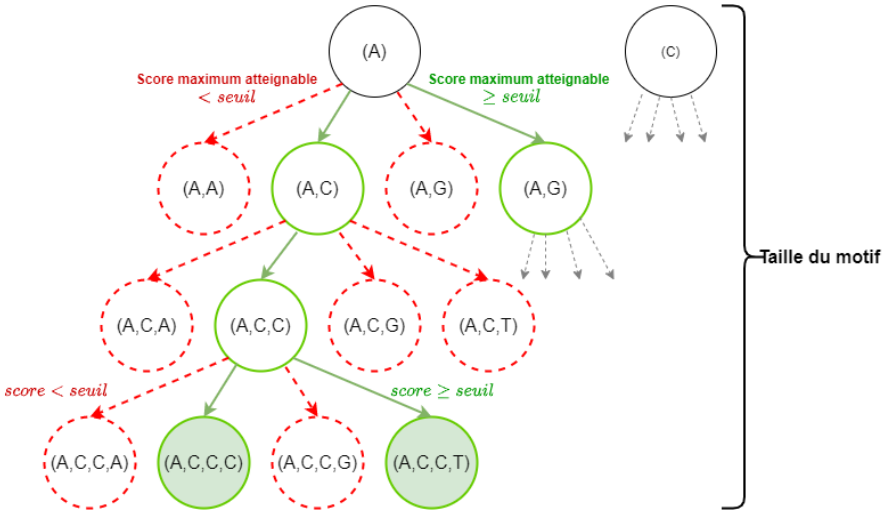


FIGURE 6 – Principe de l'algorithme par séparation et évaluation. Seulement deux racines sont représentées ici par souci de clarté et d'espace, mais ce qui est développé pour la racine (A), est également appliqué aux nœuds (C), (G), (T). Les noeuds descendants ne sont créés que s'ils satisfont la condition du score maximum atteignable supérieur ou égal au seuil. Au dernier niveau de l'arbre, on évalue directement le score et non le score maximum atteignable afin de ne garder que les mots correspondant à nos critères de sélection.

4.2.3 Algorithme d'énumération

L'algorithme 4 décrit l'algorithme d'énumération qui construit partiellement le *trie* avec une stratégie de profondeur d'abord. Nous avons choisi de créer autant de racines que d'éléments de notre alphabet. À partir de ces racines seront créés les nœuds descendants passant l'évaluation du score grâce à la LAT ou la LAM. Chaque nœud est caractérisé par 3 éléments : son niveau, sa chaîne de nucléotides et son score. Dans l'algorithme 4, nous avons utilisé la notation "objet" pour accéder à ces éléments, c'est-à-dire $nœud.l$ pour le niveau de l'objet nœuds, $nœud.v$ pour la liste de nucléotides et $nœud.s$ pour son score. Par exemple, pour le nœud (A,C,C,C) de la figure 6, $nœud.l = 3$, $nœud.v = (A, C, C, C)$ et $nœud.s = 4.3$.

Les nœuds créés sont stockés de manière temporaire dans une pile et en seront retirés dès qu'ils seront évalués. Seuls seront stockés, les mots complets ayant un score supérieur ou égal au seuil θ ainsi que leurs scores. Nous obtenons une liste de mots valides et leurs scores.

4.3 Synthèse

Avec cette approche le processus de recherche de motifs probabilistes se fait en deux temps : tout d'abord on effectue un pré-traitement où les mots possédant un score supérieur au seuil défini et selon une di-PWM sont énumérés, par la suite, on recherche ces mots valides dans une séquence.

Complexité de l'énumération en fonction de la sortie En ce qui concerne la partie énumération des mots, la complexité en temps est dans le cas d'un algorithme

Algorithme 4 : Enumerate words over threshold diPWM using with LookAheadTable or LookAheadMatrix

Input : Alphabet Σ of size σ, di-PWM matrix P of size $\sigma^2 \times (m-1)$ for a motif of length m, θ score threshold

Output : List A of all words scoring above the threshold and their score

1a $L \leftarrow$ Table of size $(m-1)$

1b $M \leftarrow$ Matrix of size $\sigma \times (m-1)$

2 $A \leftarrow$ Empty list

3 Each nodes gets : a level l, a list of nucleotides v and a score s

4 **for** $c \in \{0, \dots, \sigma - 1\}$ **do**

5 $\quad root \leftarrow (0, 0, (c))$

6 $\quad pile.push(root)$

7 **while** $pile.notEmpty()$ **do**

8 $\quad parent \leftarrow pile.pop()$

9 $\quad$ **for** $d \in \{0, \dots, \sigma - 1\}$ **do**

10 $\quad\quad b \leftarrow parent.v.last()$

11 $\quad\quad$ **if** $parent.l = m - 2$ **then**

12 $\quad\quad\quad child \leftarrow (0, 0, ())$

13 $\quad\quad\quad child.l \leftarrow parent.l + 1$

14 $\quad\quad\quad child.v \leftarrow parent.v.append(d)$

15 $\quad\quad\quad child.s \leftarrow parent.s + P[bd, parent.l]$

16 $\quad\quad\quad$ **if** $child.s \geq \theta$ **then**

17 $\quad\quad\quad\quad A.append(child.v, child.s)$

18a $\quad\quad$ **if** $parent.s + P[bd, parent.l] + L[parent.l + 1] \geq \theta$ **then**

18b $\quad\quad$ **if** $parent.s + P[bd, parent.l] + M[d, parent.l + 1] \geq \theta$ **then**

19 $\quad\quad\quad child \leftarrow (0, 0, ())$

20 $\quad\quad\quad child.l \leftarrow parent.l + 1$

21 $\quad\quad\quad child.v \leftarrow parent.v.append(d)$

22 $\quad\quad\quad child.s \leftarrow parent.s + P[bd, parent.l]$

23 $\quad\quad\quad pile.push(child)$

24 **return** A

exhaustif de l'ordre de $O(\sigma^m)$, c'est à dire si on énumère tous les mots possibles de longueur m pour un alphabet de taille σ. Quant à notre algorithme, il n'énumère pas tous les mots possibles. Nous examinons de la manière suivante la complexité en temps de celui-ci.

On note s le nombre de mots valides en sortie de notre algorithme. Chaque mot valide étant de longueur m, la taille de la sortie est donc $O(ms)$. L'énumération des mots se fait en profondeur d'abord dans le trie. On examine le nombre d'opérations nécessaires pour rejeter les mots non-valides. On segmente ce nombre d'opérations en nombres d'opérations pour exclure les non-valides entre deux solutions valides successives (dans l'ordre d'énumération). La branche d'un mot valide comporte m nœuds. Pour chaque nœud, on va examiner σ éléments pour lesquels le nombre d'opérations est constant. On a donc de l'ordre de σm opérations pour éliminer les éléments non-valides entre deux éléments valides. Ayant s éléments valides, le nombre d'opérations pour éliminer les éléments non-valides est de σms. L'ensemble des opérations de notre algorithme est donc l'ensemble des opérations pour énumérer les éléments valides : $O(ms)$ et des opérations nécessaires pour exclure les éléments non-valides : $O(\sigma ms)$. On a donc une complexité en temps totale $O(ms + \sigma ms)$ ce qui est équivaut à $O(\sigma ms)$. Le temps de calcul de notre algorithme est donc linéaire en fonction de la taille de la sortie ms.

Quant à la complexité en espace, elle est au pire des cas pour la pile de $O(m\sigma)$ et pour la liste des mots valides $O(ms)$. Cette complexité en espace peut être améliorée si au lieu de stocker les résultats dans une liste, ceux-ci sont écrits au fur et à mesure dans un fichier par exemple.

Comme les branches de recherches sont indépendantes, on pourrait améliorer la complexité en temps en parallélisant l'énumération des mots.

5 Algorithmes de recherche après énumération

Maintenant que les mots sont énumérés, différentes approches vont permettre d'effectuer cette recherche. L'algorithme d'Aho-Corasick est une solution pour la question nommée Recherche d'ensemble de mots ou *set pattern matching* en anglais. Il en existe d'autres, tel que MPSCAN. Une autre alternative est de rechercher la liste de mots dans une structure de donnée comme un index par exemple.

5.1 Aho-Corasick (AC)

Le principe de l'algorithme d'Aho-Corasick [1] est de construire un *trie* à partir des mots que l'on recherche. À partir de cet arbre des préfixes, un automate est créé en liant chaque nœud au plus long suffixe disponible. Cela permet ainsi de naviguer dans l'arbre lors du parcours du texte pour rechercher les mots.

L'espace occupé par l'automate est linéaire en la somme des longueurs des mots à chercher, sa construction aussi. La recherche dans un texte T avec l'automate prend un temps linéaire en la longueur de T.

Dans notre application, nous donnons à AC en entrée la liste des mots valides énumérés et le texte T. La recherche produit en sortie les positions des occurrences

pour chaque mot valide (occurrence dont nous connaissons le score grâce à l'énumération). On en déduit que l'algorithme de Aho-Corasick trouve toutes les occurrences des mots valides dans T en temps $O(ms+n)$, puisque $O(ms)$ correspond à la création de l'automate et $O(n)$ au parcours du texte de longueur n.

Il est intéressant de noter qu'une fois l'automate créé il est réutilisable avec d'autres textes en entrée. Par conséquent, pour la recherche d'un même motif avec le même seuil, mais dans une autre séquence, il n'est pas utile de recréer un automate, mais il suffit de lancer une recherche sur le nouveau texte. Dans cette approche, ce sont les mots valides qui sont indexés.

5.2 Recherche des mots dans une structure de données

Nous venons de voir qu'on peut indexer les mots valides et utiliser le texte pour rechercher ces mots indexés. Une autre stratégie est d'indexer le texte T dans lequel on veut effectuer notre recherche. Une fois T indexé, on recherche les mots un par un dans cette structure de données. Différentes procédures de requêtes sur l'index permettent de dire si le mot est présent ou non dans T, combien de fois il est présent, ou bien encore de localiser ces occurrences.

Il existe différentes façons d'indexer un texte telles que la Transformée de Burrows-Wheeler (BWT), la table des suffixes (SA), etc.

Cette approche est intéressante lorsque l'on travaille avec différents motifs mais toujours sur le même texte T. T est indexé une seule fois, puis on recherche de nombreux motifs dedans sans avoir à refaire l'indexation de T.

5.3 Synthèse

Ces différentes stratégies de recherche se basent sur le principe d'indexation, mais appliqué à un niveau différent. Indexer un texte consiste à pré-traiter le texte en le lisant une fois afin d'organiser les informations pour pouvoir ensuite faire une recherche rapide dessus. Dans le cas de l'algorithme d'Aho-Corasick, les mots sont indexés mais dans la recherche dans une structure de données, il s'agit de T qui est indexé. Dans le premier cas, cette approche est avantageuse lorsqu'on travaille sur les mêmes motifs mais qu'on effectue des recherches sur différents textes, comme différents chromosomes ou génomes. Dans le second cas, c'est l'opposé, l'approche est intéressante si on travaille sur le même génome mais qu'on fait varier les motifs.

6 Résultats et analyses

Nous avons implanté les algorithmes proposés dans ce mémoire et mesuré l'efficacité en pratique de la recherche semi-naïve, de l'énumération, de la recherche après énumération des mots valides, et finalement examiné quelques caractéristiques des di-PWM de la base HOCOMOCO.

6.1 Implémentation

La mise en place a été faite sous un système LINUX via une machine virtuelle.

Lors de cette étude, nous avons choisi d'utiliser le langage Python version 3.9.4 et 3.9.5 dans un environnement virtuel crée par le gestionnaire de paquets conda 4.10.1. Les bibliothèques numpy 1.20 et pandas 0.22 ont été utilisées pour la manipulation des données, matplotlib 3.4.2 et seaborn 0.11 nous ont permis de visualiser nos données et nos résultats. De plus, pour la manipulation des séquences au format fasta, nous avons utilisé la bibliothèque biopython 1.78.

Afin d'effectuer la recherche grâce à l'algorithme Aho-Corasick, nous avons employé une bibliothèque python Pyahocorasick 1.4.2 qui fournit des fonctions permettant la création d'un automate et la recherche dans le texte. Ce module est implémenté en C.

Nous avons utilisé les di-PWM de la base de données HOCOMOCO pour les FT chez l'homme. Elles sont au nombre de 292 et la taille des motifs varie de 10 à 27 nucléotides. Notre texte de recherche est un segment du chromosome 1 humain d'une taille proche de 50 millions de nucléotides.

6.2 Efficacité de la *LookAheadTable* vs la *LookAheadMatrix*

Dans la section 2.2, nous avons défini la LAT et la LAM et démontré leurs propriétés. En effet, nous avons pu montrer que la LAM est plus précise que la LAT. Il est intéressant de pouvoir valider expérimentalement ces propriétés et de voir concrètement quelles en sont les conséquences sur l'efficacité de l'une par rapport à l'autre, et dans quelle proportion. Pour ce faire, nous avons étudié leur efficacité dans le cas de la recherche semi-naïve et lors de l'énumération des mots.

6.2.1 Recherche semi-naïve

Afin, de comparer l'efficacité de la LAT et de la LAM, nous avons comparé pour les mêmes matrices et le même texte, le temps de calcul employé par l'une et par l'autre. Pour chaque matrice, nous avons effectué le calcul suivant :

$$(Temps_{LAT} - Temps_{LAM})/Temps_{LAM}$$

On en observe le résultat sur la figure 7.

On voit sur la figure 7 que le temps de calcul avec la LAT est en moyenne supérieur de 9% à celui avec la LAM pour une même di-PWM. On constate, comme nous le pensions grâce aux propriétés, que la table est moins contraignante que la matrice. Il y a plus de calculs effectués avec la table qu'avec la matrice pour le même résultat final. On peut donc en conclure que dans le cas de la recherche semi-naïve la LAM est plus efficace que la LAT.

On remarque que nous avons deux valeurs négatives. Cela étant inattendu, nous avons donc analysé ces deux di-PWM. Tout d'abord, il faut préciser que ces valeurs de temps ont été calculées une seule fois. Le temps de calcul mis par un ordinateur peut varier en fonction de l'activité de l'ordinateur. Il y a donc une marge d'erreur. Pour la valeur négative la plus importante, nous avons réitéré ce calcul 10 fois et obtenu 9 fois sur 10 un ratio positif. De plus, le ratio moyen de ces 10 itérations est de 0.006, ce qui est faible. Les deux valeurs de temps étant très proches, il est très facile que le ratio bascule d'un côté ou d'un autre avec les variations de temps

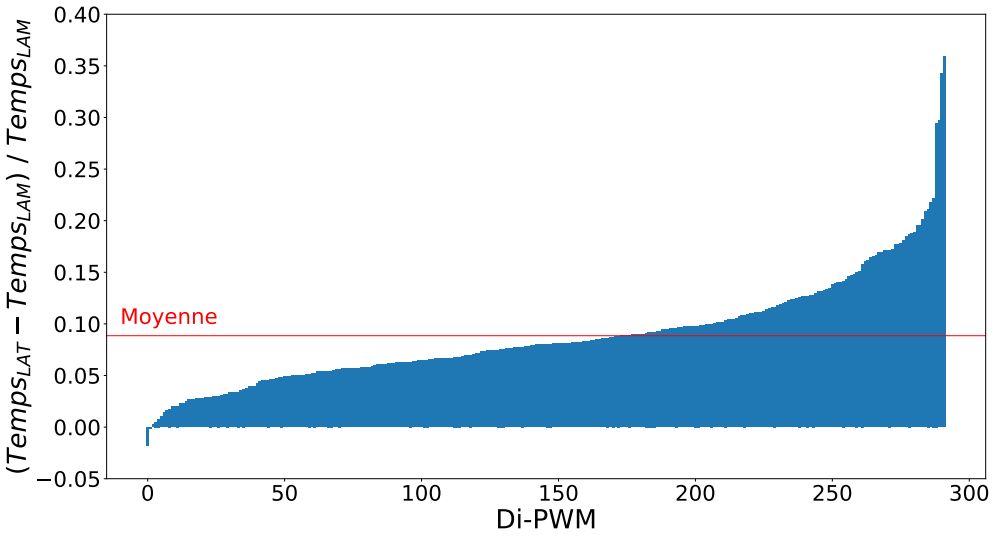


FIGURE 7 – Temps de calcul avec la LAT relatif au temps de calcul avec la LAM pour les 292 di-PWM provenant de la base de données HOCOMOCO. Ratio seuil : 0.90. La moyenne est de 9%.

de calcul d'un ordinateur. Dans l'idéal, nous aurions aimé avoir le temps d'effectuer de plus nombreuses itérations de ces calculs pour l'ensemble de ces di-PWM. Les valeurs que nous avons ici sont essentiellement indicatives.

6.2.2 Énumération des mots

Nous comparons l'impact de la LAT et la LAM sur l'efficacité de notre algorithme de séparation et évaluation pour l'énumération des mots. Pour comparer leur impact lors de l'énumération, il nous semble intéressant d'observer les interruptions d'évaluation des branches du trie, interruptions que nous appelons « coupes ». Tout d'abord, nous analysons à la figure 8 le nombre total de coupes avec la table (LAT) relatif au nombre total de coupes avec la matrice (LAM) pour toutes les di-PWM humaines de HOCOMOCO.

On observe que ce rapport est toujours positif, indiquant que le nombre de coupes est plus important lors de l'utilisation de la LAT qu'avec la LAM. Cela signifie que pour obtenir le même nombre de mots final, il a été nécessaire de réaliser davantage de coupes avec la LAT ; ces coupes sont donc moins efficaces c'est-à-dire plus tardives ou plus profondes dans le trie. En moyenne, la LAM réduit de 56% le nombre total de coupes.

Il nous parait intéressant d'avoir une vision globale de la répartition de ces coupes. Pour chacune des méthodes, nous comptabilisons le nombre de coupes qui sont faites à l'intérieur de l'arbre et celles qui sont réalisées au niveau des feuilles. Les coupes au niveau des feuilles sont les mots qui ont été générés en totalité mais qui finalement ne passent pas le critère de seuil. Ce sont donc des mots complets énumérés inutilement. Les coupes à l'intérieur de l'arbre sont les coupes plus précoces qui permettent de couper des branches et donc d'éviter l'énumération d'un

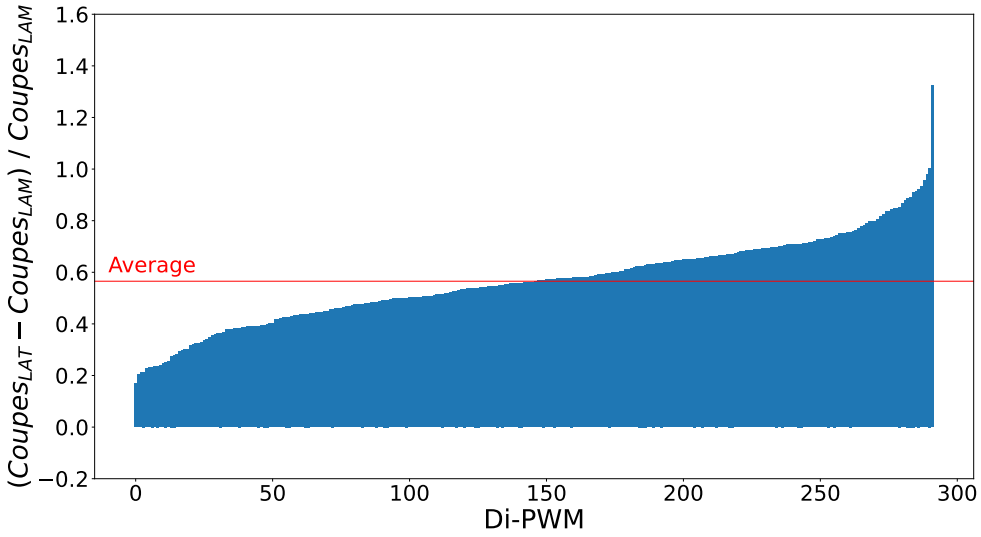


FIGURE 8 – Nombre total de coupes avec la LAT relatif au nombre total de coupes avec la LAM pour les 292 di-PWM provenant de la base de données HOCOMOCO. Ratio seuil : 0.90. La moyenne est de 56%.

ensemble de mots.

Pour chaque matrice, nous effectuons le calcul suivant :

$$(Coupes_{LAT} - Coupes_{LAM})/Coupes_{LAM}$$

Les résultats sont présentés dans la figure 9 pour les coupes internes et dans la figure 10 pour les coupes au niveau des feuilles.

On constate, d'une manière générale, que dans la quasi totalité des cas, ce rapport est positif. En moyenne, le nombre de coupes internes est inférieur de 40% avec la LAM. Quant au nombre de coupes au niveau des feuilles, il est de 140% moindre avec la LAM, pouvant atteindre des valeurs extrêmes proches de 600%. Les rapports positifs attestent qu'avec l'utilisation de la table de plus nombreux mots n'atteignant pas les critères de seuil sont énumérés. La matrice permet des coupes plus précoces et donc une économie de calculs conséquente.

On peut donc en conclure que comme nous l'avions démontré de manière théorique l'utilisation de la LAM pour l'évaluation est plus efficace que celle de la LAT, que ce soit pour l'énumération des mots ou pour la recherche semi-naïve.

6.3 Analyse de l'énumération

Nous avons vu que quel que soit l'algorithme, la LAM est plus efficace. Nous souhaitons à présent observer plus précisément le comportement de l'algorithme d'énumération en fonction des di-PWM, du seuil ou de la taille des motifs.

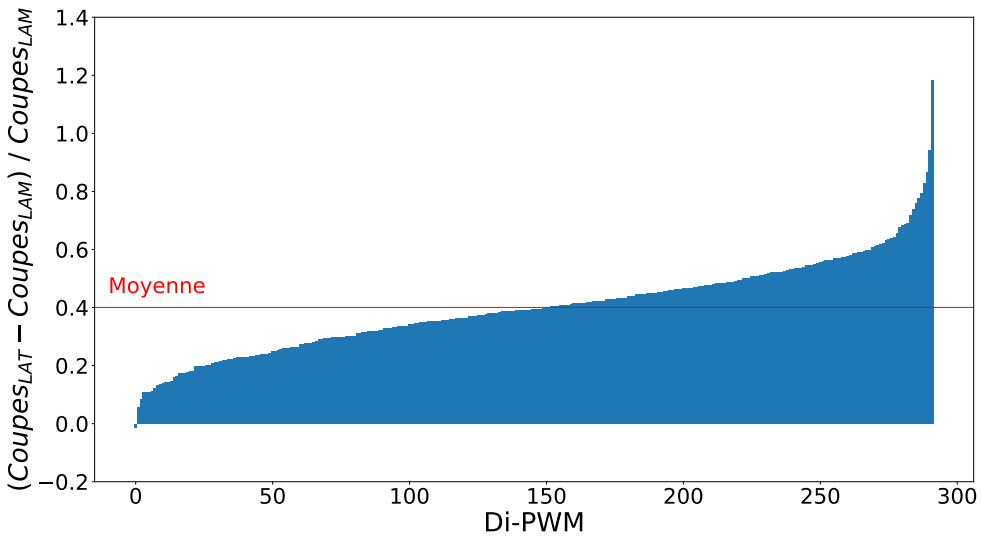


FIGURE 9 – Nombre de coupes internes avec la LAT relatif au nombre de coupes internes avec la LAM pour les 292 di-PWM provenant de la base de données HOCOMOCO. Ratio seuil : 0.90. La moyenne est de 40%.

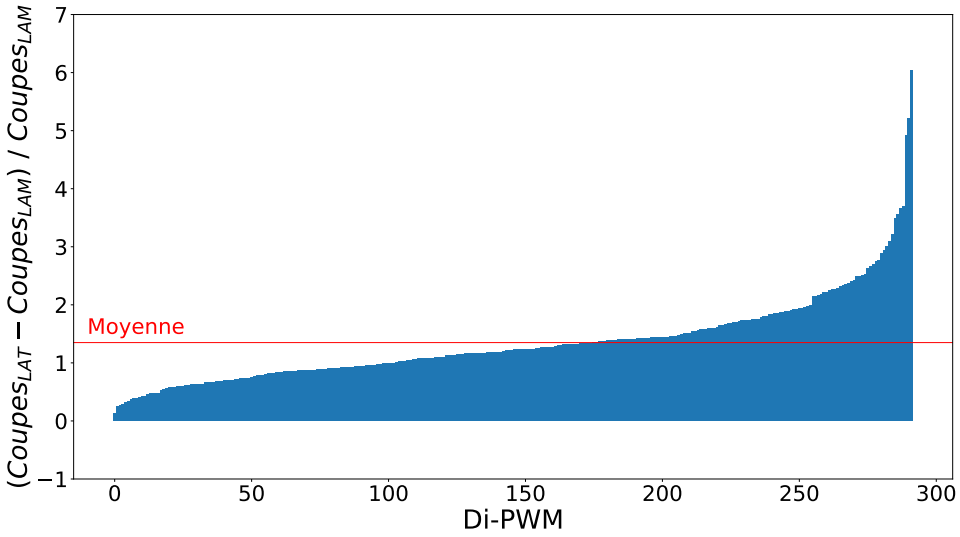


FIGURE 10 – Nombre de coupes au niveau des feuilles avec la LAT relatif au nombre de coupes au niveau des feuilles avec la LAM pour les 292 di-PWM provenant de la base de données HOCOMOCO. Ratio seuil : 0.90. La moyenne est de 140%.

6.3.1 Comportement des di-PWM lors de l'énumération

Par ailleurs, nous avons observé l'évolution du nombre de mots énumérés en fonction du seuil fixé et en fonction de la taille des motifs. Pour des raisons pratiques, nous avons stoppé l'énumération lorsque celle-ci atteignait 100 millions de mots. Nous avons donc représenté graphiquement le nombre de mots énumérés pour chaque

di-PWM en les ordonnant par taille de motif. Nous avons décliné cette représentation pour 4 ratios de seuil différents : 0.95, 0.90, 0.85 et 0.80. Cela nous permet de visualiser l'évolution du nombre de mots énumérés en fonction du seuil. Ces résultats sont présentés à la figure 11.

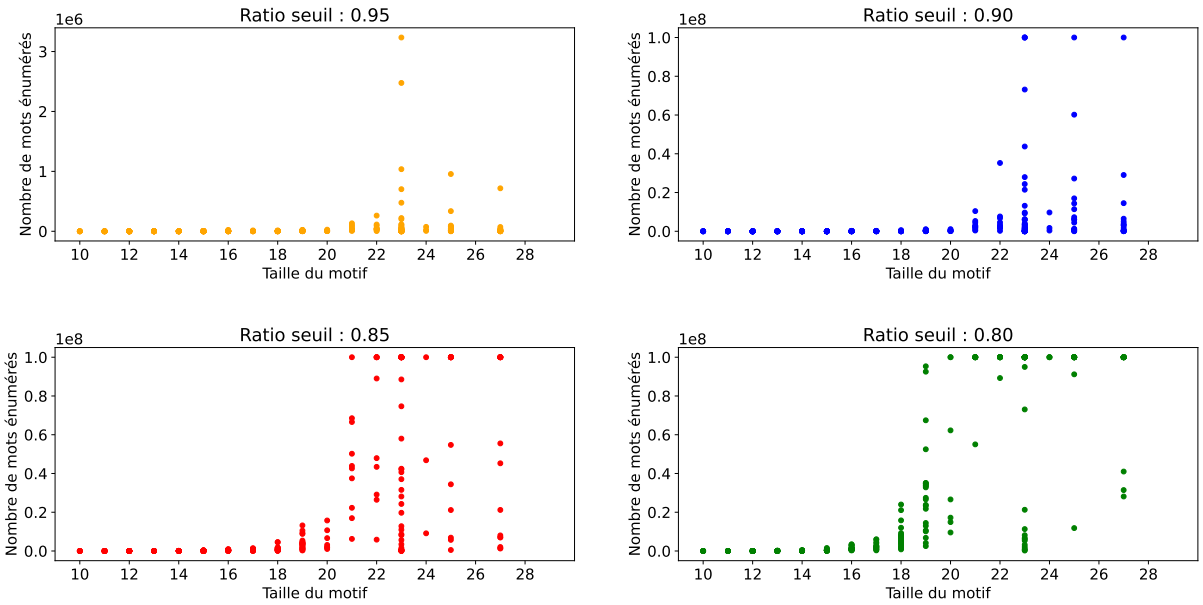


FIGURE 11 – Nombre de mots énumérés en fonction de la taille du motif. Chaque sous figure correspond à un seuil. Il est important de noter que l'énumération a été stoppée à 100 millions de mots énumérés, ce qui est visible sur les figures par l'accumulation de points à cette valeur. Notons que l'echelle du nombre de mots énumérés est de $1e^8$ à l'exception du ratio seuil 0.95 où elle est de $1e^6$.

On note une tendance du nombre de mots énumérés à augmenter à la fois avec la taille du motif mais aussi avec la baisse du seuil. Ce résultat n'est pas surprenant. Cependant, ce qui est remarquable, est que pour un motif de même longueur, les di-PWM n'ont pas le même comportement. Avec la diminution du seuil, le nombre de mots énumérés pour certaines d'entre elles va augmenter de façon linéaire alors que pour d'autres le nombre de mots énumérés va augmenter exponentiellement.

On peut donc en conclure que le contenu du motif en lui-même a une influence sur le nombre de mots énumérés pour une taille de motif et un seuil donnés. Nous aborderons cet aspect dans la partie 6.4.

6.3.2 Évolution du temps d'énumération

Nous avons également analysé le temps de calcul nécessaire à l'énumération. Nous avons calculé pour chaque di-PWM le temps de calcul pour l'énumération divisé par le nombre de mots énumérés afin d'étudier le comportement de notre algorithme. Cela équivaut au temps de calcul moyen pour énumérer un mot. Nous avons choisi de représenter ces informations sous forme d'un diagramme en violon à la figure 12.

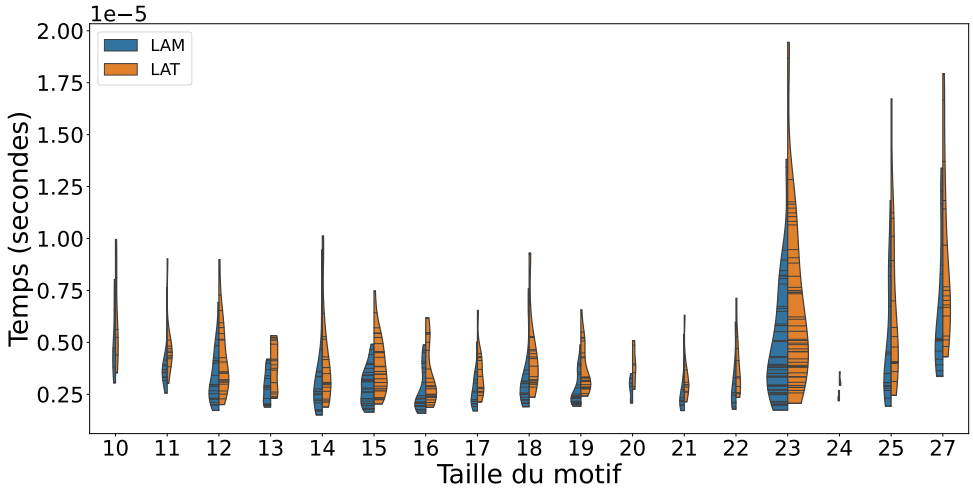


FIGURE 12 – Temps pour énumérer un mot, pour toutes les di-PWM de HOCOMOCO (partie humaine), groupées en fonction de la taille des motifs. Ratio seuil : 0.90

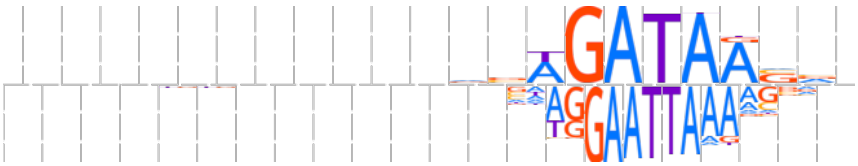
Grâce à ces résultats, nous remarquons que le temps pour énumérer un mot est stable quelle que soit la taille du motif, de plus ce temps reste assez bas, il est compris entre 0.2 et 0.75×10^{-5} secondes. Cela montre l'efficacité de notre algorithme d'énumération, car le temps de calcul augmente de manière linéaire avec le nombre de mots à énumérer, c'est-à-dire en fonction de la sortie de celui-ci et non de manière exponentielle.

6.4 Analyse des motifs

Comme nous l'avons vu précédemment le nombre de mots valides à énumérer ne peut être prédit par la taille du motif et le seuil donné. Il va déprendre de la constitution du motif en lui-même. Nous allons chercher à déterminer ce qui peut créer de telles différences entre deux di-PWM de même taille.

6.4.1 Les motifs

Les di-PWM peuvent être représentées visuellement par un logo dont la construction dépend du contenu en information des positions. En effet, plus une position est conservée, c'est-à-dire une position pour laquelle on retrouve très fréquemment le même nucléotide, plus son contenu en information est élevé. À l'opposé, si à une position donnée chaque nucléotide est équiprobable, alors le contenu en information est nul. Une colonne du logo correspond à une position du motif et sa hauteur va dépendre du contenu en information à cette position. Les lettres correspondent aux nucléotides présents à cette position et leur hauteur est fonction de leur fréquence observée. Pour éclairer notre propos, nous allons comparer deux di-PWM et leurs logos. Ces deux di-PWM sont des modèles de motifs de taille 23 nucléotides. Il s'agit des di-PWM pour les FT GATA2 et ZNF274. Leurs logos respectifs apparaissent à la figure 13 (13a et 13b respectivement).



(a) Logo de la di-PWM du facteur de transcription GATA2

(b) Logo de la di-PWM du facteur de transcription ZNF274

FIGURE 13 – Visualisation en LOGO de deux di-PWM de même longueur (23 nuc.). Elles correspondent aux sites de liaison des facteurs de transcription GATA2 et ZNF274. Chaque colonne de di-PWM est représentée : la hauteur des symboles est proportionnelle à leur fréquence relative à cette position.

Aux vues des deux figures, force est de constater que les deux di-PWM malgré leur taille commune de 23 nucléotides, sont très différentes dans leur constitution. On observe que ZNF274 a un contenu en information important à toutes les positions contrairement à GATA2 qui ne possède un contenu en information important que sur moins d'une dizaine de positions. Nous allons étudier les conséquences de cela sur l'énumération des mots. On voit sur la figure 14 l'évolution du nombre de mots énumérés pour chacune des di-PWM en fonction du seuil. Pour des raisons pratique, nous avons limité les calculs de l'énumération à 500 millions de mots.

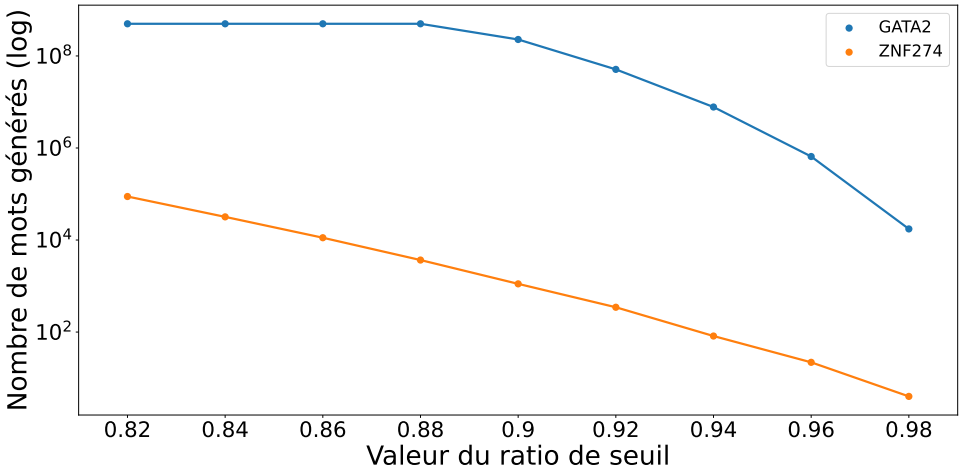


FIGURE 14 – Nombre de mots générés en fonction du seuil pour 2 di-PWM dont le motif est de longueur 23 nucléotides. Il s'agit des di-PWM des facteurs de transcription GATA2 et ZNF274. L'énumération a été limitée à 500 millions de mots ce qui explique le plateau pour GATA2.

Ici encore, on observe que les deux di-PWM ont un comportement différent l'une

de l'autre. Bien qu'on ait dû limiter les calculs, on constate pour la di-PWM du FT GATA2 que le nombre de mots énumérés décroît de manière exponentielle passant de 500 millions de mots jusqu'au seuil 0.88 à 17 421 mots à un ratio seuil de 0.98. Quant à la di-PWM du FT ZN274 la diminution du nombre de mots énumérés est exponentielle également, mais avec un gros écart en terme d'ordre de grandeur, passant de 88 230 mots pour un ratio seuil de 0.80 à 4 mots pour un ratio seuil à 0.98.

Notre hypothèse est que lorsque le contenu en information est faible à une position donnée, le score pour chaque élément sera quasi identique et donc non déterminant, et toutes les combinaisons envisageables. Si cela se produit à plusieurs positions, le nombre de combinaisons acceptables va augmenter de manière exponentielle.

6.4.2 Possibilités

Nous n'avons pas d'outil nous permettant de prédire le comportement d'une di-PWM lors de l'énumération, même si parfois aux vues du logo on peut l'imaginer sans en être certains. De plus, les logos que nous avons choisi sont des extrêmes afin de voir les différences aisément mais cela n'est pas toujours aussi marqué, donc, pas toujours aussi facilement prévisible.

Nous avons donc développé la possibilité dans notre implémentation de lancer notre programme sans lister les mots et leurs scores mais uniquement en les comptant. Cela permettra à l'utilisateur, lors de l'emploi du module, d'effectuer un test de son seuil avec sa di-PWM et de voir si le nombre de mots à énumérer et donc à rechercher par la suite est exploitable.

Une autre possibilité à laquelle nous avons pensé mais que nous n'avons implémenté que pour certains cas particuliers, est la possibilité de sélectionner les colonnes ayant un contenu en information non nul ou supérieur à un certain seuil, créant ainsi une di-PWM réduite. Les mots sont énumérés en un nombre plus raisonnable puis la recherche dans le texte de cet ensemble de mots est menée. Celle-ci renvoie alors les positions trouvées pour la di-PWM réduite. Il faut ensuite traiter chacune de ces positions comme un site potentiel pour le motif de taille complète et vérifier si cette position correspond à une réelle réponse à notre di-PWM et à notre seuil de départ.

Les différents profils de di-PWM ont également une conséquence sur la recherche semi-naïve. Nous avons donc développé une variante de notre algorithme de recherche semi-naïve, cf. section 3.4.

Il va également traiter toutes les fenêtres du texte mais il va les aborder par la dernière position du motif et non la première. Par conséquent, selon le contenu en information des positions de début ou de fin de la di-PWM une approche sera plus avantageuse que l'autre. Pour l'illustrer, nous avons comparé le temps mis par les deux approches pour une même matrice, pour un même seuil et le même texte avec l'utilisation de la *LAM*. La figure 15 présente les résultats du calcul suivant pour chacune des matrices :

$$Temps_{GD} - Temps_{DG}$$

où $Temps_{GD}$ est le temps de recherche semi-naïve par traitement de la fenêtre de gauche à droite et $Temps_{DG}$ est le temps de la recherche semi-naïve par traitement

de la fenêtre de droite à gauche.

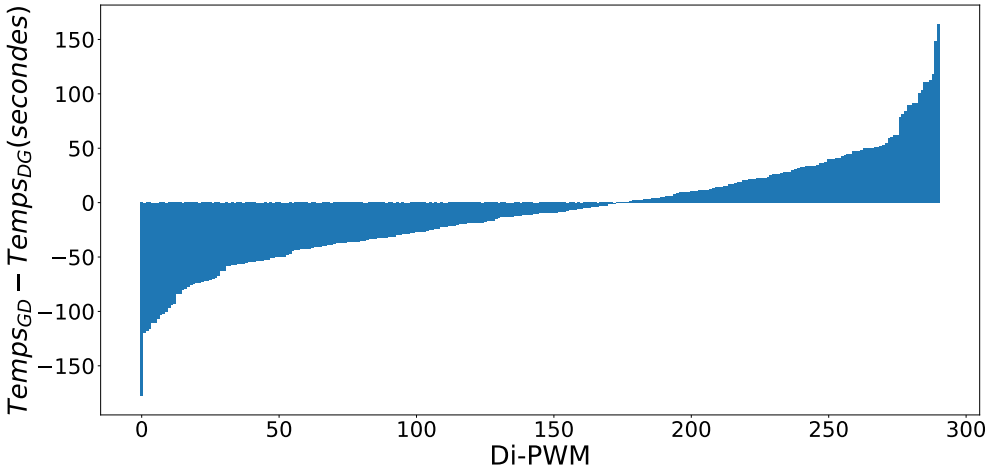


FIGURE 15 – Différence de temps pour la recherche semi-naïve avec lecture du motif de gauche à droite(GD) avec lecture du motif de droite à gauche(DG).

On constate que pour environ 40% des di-PWM il est intéressant de les traiter dans le sens opposé.

6.5 Temps de recherche

Bien que nos codes puissent encore être optimisés, il nous semble intéressant d'avoir un ordre de grandeur du temps nécessaire à la recherche de motifs avec nos algorithmes. Nous avons donc pris pour exemple quelques di-PWM aux tailles variées dans la table 1. Pour cette recherche nous avons fixé un ratio de seuil à 0.9 et utilisé une séquence d'ADN humaine issue du chromosome 1 d'une taille d'environ 50 millions de nucléotides. De plus, nous utilisons la *LAM* que nous savons plus efficace.

Nom di-PWM	Taille du motif	Temps de recherche Semi-naïve	Temps d'énumération + temps de recherche Aho-Corasick
COE1	23	221 s	68 + 27 = 96 s
CREB1	23	148 s	35 + 8 = 43 s
EGR1	21	117 s	6 + 3 = 9 s
GATA1	21	140 s	20 + 7 = 27 s
ATF2	14	125 s	0.05 + 1.4 = 1.45 s
DUX4	14	79 s	0.0008 + 1 = 1.1 s

TABLE 1 – Quelques exemples du temps mis pour effectuer une recherche de motif dans une séquence d'ADN. Pour cette recherche, le ratio de seuil a été fixé à 0.9, la *LAM* a été utilisée et la séquence d'ADN est un segment du chromosome 1 humain d'une longueur d'environ 50 millions de nucléotides.

On constate que le temps cumulé de l'énumération et de la recherche grâce à l'algorithme d'Aho-Corasick reste inférieur au temps nécessaire pour effectuer la même recherche avec la stratégie de recherche semi-naïve. Or il existe d'autres alternatives tel que MPSCAN [6] basé sur un algorithme de recherche d'ensemble de mots sur une séquence d'ADN, qui est encore plus efficace en temps qu'Aho-Corasick. Cela donne donc des pistes d'optimisation de la recherche après énumération des mots.

7 Conclusion

Le problème initial de ce stage était de développer de nouveaux algorithmes pour la recherche de motifs de type di-PWM dans une séquence d'ADN. L' objectif était d'implémenter ces algorithmes en python afin de les tester, de les comparer et de les rendre disponible à la communauté sous forme d'un module Python.

Nous avons développé deux stratégies différentes pour traiter ce problème. Nos deux approches ont en commun l'utilisation d'une structure d'évaluation afin de limiter le nombre de calculs, la LAT ou la LAM. Nous avons pu montrer de manière théorique et expérimentale que quelque soit l'algorithme utilisé la structure que nous avons spécifiquement créé pour les di-PWM, la LAM, est plus efficace.

Notre premier algorithme est un algorithme de recherche par fenêtre glissante qui va traiter toutes les fenêtres de la séquence mais qui va limiter le nombre de calculs pour chaque fenêtre dès que cela est possible. Nous en avons également développé une variante qui va aborder la fenêtre non plus de gauche à droite mais de droite à gauche. Cette variante aura un intérêt sur certaines di-PWM, en fonction de la constitution du motif.

Notre deuxième algorithme novateur pour les di-PWM a un fonctionnement différent car il va tout d'abord énumérer les mots valides à partir de la di-PWM et du seuil donné. Dans un second temps, cet ensemble de mot va être indexé et former un automate grâce auquel on va pouvoir rechercher les mots lors du parcours du texte de la séquence d'ADN. Nous avons vu que d'autres approches étaient possibles suite à l'énumération des mots valides.

Par ailleurs, ce travail a été présenté sous forme d'un poster lors de la conférence JOBIM durant ce mois de juillet.

Les tests que nous avons effectués ont montré que les deux approches que nous avons développées sont tout à fait utilisables. Bien que le code ne soit pas encore optimisé les temps de calculs des deux stratégies sont raisonnables. On constate que la stratégie par énumération est la plus efficace et peut encore être améliorée.

Le module Python que nous souhaitons mettre à disposition de la communauté n'est pas encore terminé mais nous espérons qu'il le sera bientôt.

Bien évidemment, si nous avions plus de temps, de nombreuses perspectives seraient envisageables. Tout d'abord, au niveau de la base de données HOCOMOCO, nous avons pensé qu'il pourrait être intéressant de stocker pour chaque di-PWM les mots énumérés pour les seuils les plus couramment utilisés. Cela permettrait ainsi aux utilisateurs de la base de données de ne pas à avoir à les générer. De même, pourrait être noté pour les di-PWM le sens de traitement de la fenêtre le plus avantageux pour les utilisateurs souhaitant effectuer une recherche par fenêtre glissante.

En ce qui concerne le module, on peut imaginer l'étoffer en intégrant d'autres fonctionnalités de recherche après énumération, telle que l'utilisation d'un FM-index ou autre.

Quant au niveau algorithmique, nous avions vu lors du projet bibliographique des motifs plus complexes sur lesquels il serait intéressant de travailler et d'appliquer les principes que nous avons employés pour les di-PWM. Une prochaine étape pourrait être de développer ces algorithmes pour des Modèles de Markov Cachés.

Références bibliographiques

[1] Alfred V. Aho et Margaret J. Corasick. « Efficient string matching ». In : *Communications of the ACM* 18.6 (juin 1975), p. 333-340. doi : 10.1145/360825.360855. url : https://doi.org/10.1145/360825.360855.

[2] Ivan Kulakovsky et al. « From binding motifs in ChIP-seq data to improved models of transcription factor binding sites ». In : *Journal of Bioinformatics and Computational Biology* 11.01 (fév. 2013), p. 1340004. doi : 10.1142/s0219720013400040. url : https://doi.org/10.1142/s0219720013400040.

[3] David Martin, Vincent Maillol et Eric Rivals. « Fast and Accurate Genome-Scale Identification of DNA-Binding Sites ». In : *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. 2018, p. 201-205. doi : 10.1109/BIBM.2018.8621093.

[4] Cinzia Pizzi, Pasi Rastas et Esko Ukkonen. « Fast Search Algorithms for Position Specific Scoring Matrices ». In : *Bioinformatics Research and Development*. Sous la dir. de Sepp Hochreiter et Roland Wagner. Berlin, Heidelberg : Springer Berlin Heidelberg, 2007, p. 239-250. isbn : 978-3-540-71233-6. doi : 10.1007/978-3-540-71233-6_19.

[5] Cinzia Pizzi, Pasi Rastas et Esko Ukkonen. « Finding Significant Matches of Position Weight Matrices in Linear Time ». In : *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 8.1 (jan. 2011), p. 69-79. doi : 10.1109/tcbb.2009.35. url : https://doi.org/10.1109/tcbb.2009.35.

[6] Eric Rivals et al. « MPSCAN: Fast Localisation of Multiple Reads in Genomes ». In : *Algorithms in Bioinformatics*. Sous la dir. de Steven L. Salzberg et Tandy Warnow. Berlin, Heidelberg : Springer Berlin Heidelberg, 2009, p. 246-260. isbn : 978-3-642-04241-6. doi : 10.1007/978-3-642-04241-6_21.