PSH-DB, un système clé-valeur permettant l'indexation et la recherche de séquences ADN

Jocelyn de Goër, Myoung-Ah Kang, Xavier Bailly, Engelbert Mephu-Nguifo

PSH-DB, un système clé-valeur permettant l'indexation et la recherche de séquences ADN

Jocelyn DE GOËR
EPIA, INRA, VetAgro Sup,
63122, Saint Genès Champanelle,
France
Université Clermont Auvergne, CNRS,
LIMOS,
F-63000 Clermont-Ferrand
France
jocelyn.degoer@inra.fr

Myoung-Ah KANG
Université Clermont Auvergne, CNRS,
LIMOS,
F-63000 Clermont-Ferrand
France
kang@isima.fr

Xavier BAILLY
EPIA, INRA, VetAgro Sup, 63122, Saint
Genès Champanelle,
France
xavier.bailly@inra.fr

Engelbert MEPHU-NGUIFO
Université Clermont Auvergne, CNRS,
LIMOS,
F-63000 Clermont-Ferrand
France
engelbert.mephu_nguifo@uca.fr

ABSTRACT

L'accroissement constant des capacités de séquençage de l'ADN, entraîne l'émergence de nouveaux questionnements biologiques. Le stockage et le traitement de cette masse de données en constante évolution, reste un enjeu majeur pour les années à venir. Durant le processus d'analyse des données génomiques, la recherche de sous-séquences, au travers de bases de données de génomes de référence, est une tâche incontournable**.**

Nous présentons un système de type clé-valeur in-memory (PSH-DB - Perceptual Sequence Hashing Data Base), couplé avec une fonction de hachage perceptuelle (PSH - Perceptual Sequence Hashing), spécifiquement conçue pour l'indexation et la recherche de séquences ADN. Lors de la phase d'indexation, les clés de hachage sont calculées à partir des séquences ADN et stockées sous forme de chaîne binaire. Ceci permet nativement le calcul de Distance de Hamming, nécessaire pour comparer les clés de hachage et ainsi à partir d'une séquence requête, retourner les séquences les plus proches présentes dans la base.

La fonction de hachage PSH, est basée sur des concepts de hachage perceptuel, utilisés pour indexer et comparer des images numériques. Les séquences ADN/ARN à indexer, doivent être converties sous la forme de matrices de pixels, afin de créer de nouvelles structures 2-D directement représentatives de celles-ci. Les clés de hachage sont générées à l'aide d'une Transformée en Cosinus Discrète à Coefficients Signés. Elles s'avèrent être moins volumineuses que les séquences à partir desquelles elles ont été calculées. Malgré la diminution de la taille des données entre une séquence et sa clé de hachage, les clés de hachage conservent la propriété de pouvoir être comparées à l'aide de la Distance de Hamming. Par conséquent, les clés de hachage générées à partir de deux séquences proches, auront une distance de Hamming faible.

Les expérimentations menées à partir de données réelles démontrent les capacités d'indexation et de recherche du système en termes de sensibilité par rapport à BLAST, l'outil de référence en bio-informatique, mais aussi en termes de diminution des données et de vitesse d'exécution des fonctions de recherche par rapport au système clé-valeur REDIS.

CCS CONCEPTS

• **Theory of computation** → **Design and analysis of algorithms** → Streaming, sublinear and near linear time algorithms → Bloom filters and hashing

• **Applied computing** → **Life and medical sciences** → Bioinformatics

KEYWORDS

Séquence ADN, hachage perceptuel, table de hachage, indexation

1 INTRODUCTION

L'évolution constante des techniques de séquençage de l'ADN, entraîne la production de plus en plus massive de données génomiques, pour un coût de plus en plus bas. L'apparition de séquenceurs

toujours plus modulables et portatifs [1], qui permettent la lecture de fragments d'ADN, de plus en plus long, va conduire à une démocratisation certaine de ces outils. Outre l'émergence de nouveaux questionnements biologiques [2], le stockage et les besoins d'analyse rapide de cette masse d'informations est un enjeu majeur pour les années à venir. Après les étapes de séquençage, l'une des premières étapes de leur analyse, est la caractérisation des séquences ADN obtenues, afin de déterminer le génome de référence le plus proche, en utilisant des bases de données de références. La recherche de fragments de séquences ADN/ARN, au travers de bases de données est aussi utilisée pour les tâches d'assemblage de plusieurs séquences à partir d'une référence [3], pour étudier les mutations ou pour déterminer une sous-séquence commune entre plusieurs séquences. Afin de répondre à ces problématiques, de nombreuses méthodes en bio-informatique ont été proposées, utilisant des techniques de comparaison de textes [4] dérivées en majorité de l'algorithme de Boyer–Moore [5], d'alignement local [6], d'indexation de k-mers [7], d'utilisation d'arbres des suffixes [8] comme structure de données. Bien que ces méthodes soient reconnues comme étant très efficaces, leur utilisation peut être limitée de par leur complexité algorithmique, ce qui peut avoir pour conséquence de rendre difficile le changement d'échelle en termes de taille des données dans les années à venir.

La littérature décrit aussi l'utilisation de table de hachage [9], permettant la recherche exacte et l'alignement de séquences ADN. Le processus d'indexation, nécessite le découpage en sous-séquences (k-mer), des séquences à indexer et pour chacun d'entre eux de conserver leurs positions sur la séquence dont ils sont issus. Ceci peut donc s'avérer couteux en espace de stockage.

Compte tenu de l'accroissement des données à analyser, l'un des enjeux est de créer des algorithmes permettant d'identifier rapidement les génomes de référence les plus proches, d'une séquence nouvellement séquencée. Ceci afin d'effectuer un rapide prétraitement permettant de conserver uniquement des séquences de références pertinentes pour qu'elles soient par la suite analysées par des outils plus spécifiques aux questionnements biologiques.

Parallèlement, l'identification de documents (audio et image) au travers de bases de données de référence est un domaine, qui fait l'objet de nombreuses recherches depuis plusieurs années [10]. Les principales méthodes se basent sur des concepts de calculs d'empreintes, et plus particulièrement de hachage perceptuel [11]. À partir d'un document (audio ou image), plusieurs clés sont calculées à l'aide d'une fonction de hachage perceptuel. Les clés de hachage sont alors uniques, ont une taille très inférieure aux documents d'origine mais restent néanmoins représentatives des données fournies en entrée. Ces clés peuvent alors être comparées entre elles. Ceci permet de constituer des bases de données de références, les clés de hachage servant de système d'indexation, pour l'identification de document requête.

Y. Ke et al. [12] décrivent une technique d'identification de musique via une méthode d'analyse d'images, où le signal audio, un signal à une dimension (1D), est converti en plusieurs images à deux dimensions (2D). Cette méthode qui donne des

résultats très satisfaisants, démontre l'intérêt d'effectuer une conversion d'un signal 1D en un signal 2D, afin de ne plus se baser sur les caractéristiques d'un signal audio mais sur sa représentation sous forme d'images. Le passage à une structure 2D apportant donc plus de diversité, pour la création de clés de hachage.

S. Prungsinchai et al. [13], décrivent une fonction de hachage perceptuel permettant de comparer des images numériques. Cette méthode se base sur une transformée en cosinus discrète (TCD) [14] à coefficients signés [15], qui possède, des caractéristiques de regroupement fréquentiel, comparables à une Transformée de Fourier Discrète (TFD) [16], mais avec une complexité algorithmique moins importante [17]. À partir d'une image numérique, cette fonction de hachage a la particularité de conserver la structure de l'image d'origine, en se basant uniquement sur les signes des coefficients de la TCD.

Concernant les travaux menés en bio-informatique, *MC Saldías et Al.* [18], ont pu démontrer un réel potentiel concernant l'utilisation de méthodes issues de l'analyse d'images, pour aligner des séquences ADN. L'article décrit le processus d'encodage des séquences ADN sous forme d'images 2D puis l'utilisation d'une méthode de corrélation de phase de TFD. Bien que les temps de réalisation des processus d'alignement apparaissent nettement plus lents que l'outil de référence en bio-informatique BLAST, cet article démontre un certain intérêt quant à l'utilisation de méthodes permettant de comparer des images pour comparer des séquences nucléotidiques. L'approche méthodologique utilisée dans cet article est proche des méthodes de hachage perceptuel, car elle se base sur une fonction de traitement du signal basée sur une TFD, qui permet de déterminer les fréquences significatives d'une image et ainsi de les utiliser comme identifiants discriminants.

Nous proposons PSH-DB, un système clé-valeur in-memory, utilisant une fonction de hachage perceptuel, conçue spécifiquement pour l'indexation des séquences nucléotidiques (ADN/ARN). Cette première implémentation a pour objectif de démontrer l'intérêt de la fonction PSH, pour indexer et rechercher des séquences exactes ou proches au sein d'une base de données de séquences nucléotidiques de référence. La caractéristique principale du moteur PSH-DB est qu'il permet de stocker nativement les clés de hachage sous forme de chaînes binaires (bitset), contrairement à la majorité des moteurs NoSQL, qui stockent les clés sous forme de nombres entiers ou de chaines de caractères. Le stockage au format bitset permet d'optimiser l'espace de stockage, mais surtout de calculer nativement et rapidement des distances de Hamming, entre deux clés de hachage, via l'utilisation de l'opérateur booléen XOR.

PSH-DB est architecturé autour d'un modèle client-serveur multi-threads et propose dans sa version actuelle les fonctions de base nécessaires aux différentes expérimentations réalisées.

Dans un premier temps, nous présentons les différentes étapes de l'algorithme de la fonction PSH, puis l'architecture de PSH-DB et son utilisation afin de rechercher des séquences exactes et approchées au sein de différentes bases de données de référence. Afin d'être indexées, les séquences de références sont converties sous la forme de matrices de pixels puis les clés de hachage sont calculées via la fonction PSH.

2 MATÉRIELS ET MÉTHODES

Dans cette section, nous décrivons tout d'abord, les différentes étapes de l'algorithme de la fonction PSH, puis nous présentons la structure de données PSH-DB. Enfin, nous présentons les phases de validation au travers de différentes expérimentations.

2.1 Quelques définitions

Séquence ADN/ARN : succession de nucléotides (A, T ou U, C, G), contenant l'information génétique d'un être vivant.

Paire de bases (pb) : appariement de deux bases nucléiques situées sur deux brins complémentaires d'ADN ou d'ARN. La paire de bases est l'unité de mesure déterminant la taille d'une séquence ADN.

Taux de mutation : Le taux de mutation désigne la quantité de changement de l'information génétique entre deux séquences ADN.

k-mer : sous-séquence de taille k paires de bases.

2.1 La fonction de hachage PSH

La fonction PSH, est une fonction de hachage appartenant à la famille des fonctions de hachage perceptuel. Elle est dérivée de fonctions de hachage permettant l'indexation d'images numériques et a été adaptée pour répondre aux caractéristiques des séquences nucléotidiques (ADN/ARN). De façon générale, les séquences nucléotidiques sont stockées sous la forme de chaînes de caractères (format FASTA).

Afin de pouvoir appliquer la fonction PSH, la séquence que l'on souhaite hacher doit être convertie sous la forme d'une matrice de pixels de taille *N x M*. La fonction PSH accepte en entrée des séquences de taille *W* et renvoie des clés de hachage de taille *W'*. L'objectif est de calculer, pour une séquence *S* de taille *W*, une emprunte représentative, au format binaire de taille fixe *W'*, appelée clé de hachage *H*. Une clé de hachage est directement représentative de la séquence *S* et sa taille *W'* est toujours inférieure à *W*. À l'instar des fonctions de hachage, la fonction PSH n'est pas réversible, ce qui signifie qu'à partir d'une clé de hachage il n'est pas possible de recalculer la séquence d'origine dont elle est issue.

La fonction PSH répond à plusieurs critères :

- Une clé de hachage doit être unique et directement représentative des données à partir desquelles elle a été calculée.
- La taille d'une clé de hachage doit être bien inférieure aux données fournies en entrée.

2.2 Conversion sous forme de matrice de pixels

Afin de pouvoir utiliser la fonction PSH, la séquence nucléotidique à hacher, est convertie, sous forme d'image, ou plus précisément, est encodée en

niveau de gris au sein d'une matrice de pixels (nombres entiers) de taille *N x M* (cf. *Fig.1*). Cette opération est réversible et n'entraîne pas à ce stade de perte d'information. Chaque nucléotide devient ainsi une valeur d'intensité lumineuse (L_1, L_2, L_3 et L_4), en fonction de son type (*A*, *T* ou *U*, *C*, *G*). Lors du processus d'encodage, le nucléotide suivant le dernier d'une ligne, sera donc le premier de la ligne suivante. Par conséquent, les séquences ADN habituellement en dimension 1D, sont converties en dimension 2D. Ceci permet de créer artificiellement des structures 2D et d'utiliser les propriétés de regroupement fréquentiel de la TCD pour générer les clés de hachage.

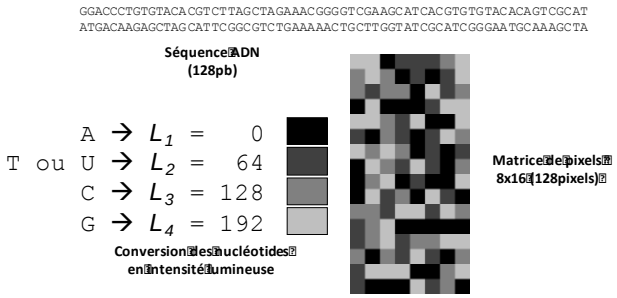


Figure 1 : Conversion d'une séquence ADN/ARN en matrice de pixels.

2.3 Génération d'une clé de hachage à partir d'une séquence ADN

Les étapes de calcul d'une clé de hachage sont illustrées en *Fig.2*. L'algorithme peut être décrit comme suit :

1. Soit une séquence d'ADN/ARN *S* de taille *W*
2. *S* est convertie en une matrice de pixels *P*, de taille *N x M*.
3. Une TCD est appliquée sur *P*, et les coefficients de la TCD sont stockés dans une matrice *P'*.
4. Une fonction *signum* (sgn) est appliquée sur la matrice *P'* afin d'obtenir les signes des différents coefficients de la TCD.
5. Tout ou partie des coefficients binaires forment la clé de hachage.

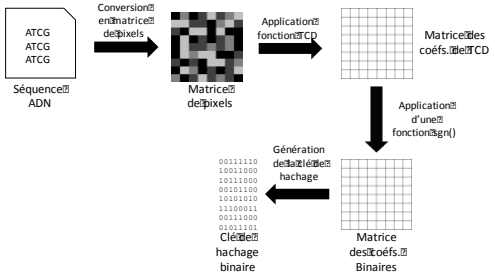


Figure 2 : Principales étapes de l'algorithme de la fonction PSH

2.4 Application d'une Transformée en Cosinus Discrète (TCD)

La Transformée en Cosinus Discrète (TCD) (cf. *Eq.1*) est une transformation mathématique proche de la transformée de Fourier Discrète (TFD), mais la TCD implique uniquement l'utilisation de Cosinus et produit des coefficients réels, tandis que la TFD utilise à la fois des Sinus et des Cosinus et génère des nombres complexes. La TCD permet de convertir des données depuis le domaine spatial vers le domaine fréquentiel. Elle autorise un changement de domaine d'étude, tout en gardant la même fonction étudiée. Sa fonction inverse, la Transformée en Cosinus Discrète Inverse (TCDi) (cf. *Eq.2*), permet le passage depuis le domaine fréquentiel vers le domaine spatial. La particularité de la TCD est qu'elle permet de concentrer l'information d'une image en un nombre de coefficients réduits mais significatifs. Lorsque la fonction TCD est appliquée à une image

(cf. *Fig.3*), sa représentation fréquentielle est obtenue dans une matrice de coefficients. La matrice de coefficients a les mêmes dimensions que l'image originale. Les hautes fréquences sont regroupées dans la partie supérieure gauche de la matrice des coefficients. Ils représentent la structure de l'image et les basses fréquences représentent les zones homogènes. En raison de ces caractéristiques de regroupement fréquentiel, la TCD est couramment utilisée au sein d'algorithmes de compression d'images ou de vidéos, tel que JPEG et MPEG.

$$DCT\ (i,j) = \frac{1}{\sqrt{2}}c(i)c(j)\sum_{x=0}^{N-1}\sum_{y=0}^{M-1} pixel(x,y)cos\left(\frac{(2x+1)i\pi}{2N}\right)cos\left(\frac{(2x+1)j\pi}{2M}\right) \quad (1)$$

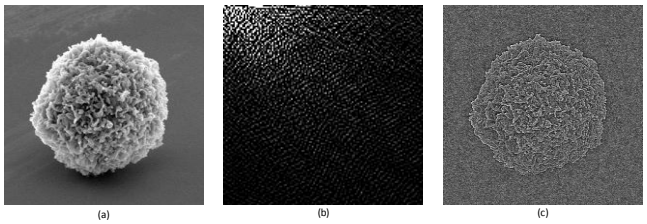
$c(x) = \frac{1}{\sqrt{2}}$ si x vaut 0 et 1 si $x > 0$

Équation 1 : Définition de la fonction Transformée en Cosinus Discrète, pour une matrice N x M

$$pixel(x,y) = \frac{1}{\sqrt{2N}}\sum_{x=0}^{N-1}\sum_{y=0}^{M-1} c(i),\ c(j)DCT(i,j)cos\left(\frac{(2x+1)i\pi}{2N}\right)cos\left(\frac{(2x+1)j\pi}{2M}\right) \quad (2)$$

$c(x) = \frac{1}{\sqrt{2}}$ si x vaut 0 et 1 si $x > 0$

Équation 2 : Définition de la fonction Transformée en Cosinus Discrète Inverse, pour une matrice N x M



(a) (b) (c)

Figure 3 : (a). Image test dans le domaine spatial. (b). Matrice des coefficients de la TCD correspondant à l'image test dans le domaine fréquentiel. (c). Image reconstituée à partir de l'information structurelle des signes de coefficients de la TCD.

2.5 Application d'une TCD à coefficients signés

L'application d'une fonction *signum*, à la matrice des coefficients de la TCD, a pour effet de ne conserver que les signes des coefficients réels. Cette méthode est connue sous le nom de TCD à coefficients signés. Elle permet, de ne conserver, que les structures principales, mais qui sont hautement représentatives de l'image originale (cf. *Fig.3*). Les signes des coefficients, ont donc un rôle essentiel car ils sont porteurs de l'information structurelle de l'image et ceci malgré la suppression de 98,43% de l'information, les coefficients étant codés sous 64bits et les signes uniquement sur 1 bit.

$$sgn\big(TCD(i,j)\big) = \begin{cases} 0\ si\ TCD(i,j) < 0 \\ 1\ si\ TCD(i,j) > 0 \end{cases} \quad (3)$$

Équation 3 : **Définition de la fonction signum, pour une matrice de coefficients d'un TCD de taille i x j**

2.6. Conversion sous forme de matrice de pixels

La clé de hachage est calculée à partir de la matrice des signes des coefficients. Par conséquent, elle est nativement au format binaire. Le nombre de coefficients binaires étant directement en relation avec le nombre de coefficients de la TCD qui sont eux

même directement issus du nombre de pixels de l'image d'origine ; la taille maximale d'une clé de hachage ne peut donc pas être supérieure à celle-ci. Cependant, si l'on souhaite ne conserver qu'une partie des coefficients binaires, dans ce cas, on notera que plus une clé de hachage est petite, plus sa probabilité de collision augmente et moins elle contient d'informations structurelles représentatives de l'image d'origine.

2.7. Taille d'une clé de hachage par rapport à la séquence ADN d'origine

L'objectif d'une fonction de hachage est de conserver la propriété de représentativité tout en réduisant la taille des clés de hachage générées par rapport aux données en entrée. Si l'on considère une séquence nucléotidique au format FASTA, où chaque nucléotide est représenté sous forme d'un caractère, encodé sous 1 octet (8 bits) de données, la clé de hachage correspondante calculée par la fonction PSH aura au minimum, une taille 8x inférieure. Chacun des signes de la matrice des coefficients étant encodé sur 1 bit de données.

2.8. Comparaison des clés de hachage

Une des caractéristiques essentielles de la fonction PSH, est qu'il est possible de déterminer une distance entre deux clés de hachage. Ceci est dû à la manière dont elles sont calculées, car, malgré le fait qu'elles soient au format binaire (bitset), elles contiennent toujours une partie de l'information structurelle de la matrice de pixels (image), dont elles sont issues (propriété de représentativité). Afin de calculer la distance entre deux clés de hachage, nous avons choisi d'utiliser la distance de Hamming [19], qui est la distance de référence pour comparer deux chaines binaires. La distance de Hamming est une distance mathématique qui exprime la somme des différences entre deux séquences de même longueur (Cf. *Eq4*). Les séquences peuvent être des suites de nombres binaires mais aussi se composer d'éléments provenant d'autres systèmes numériques ou alphanumériques. Pour calculer la Distance de Hamming entre deux séquences binaires, son implémentation ne nécessite que l'utilisation d'instructions de très bas niveau (XOR et PopCount), gérées nativement par les processeurs des machines. Nous avons vu précédemment que les clés de hachage issues des séquences sont générées au format binaire. La distance de Hamming renvoie donc un indice de distance, plus cet indice est faible et plus les deux séquences sont similaires. En revanche, la distance entre deux clés de hachage, ne permet pas de déterminer avec précision le taux de divergence entre deux séquences.

$$d(a,b) = \sum_{i=0}^{n-1} (a_i \oplus b_i) \quad (4)$$

Équation 4 : Définition de la Distance de Hamming entre deux séquences binaires *a* et *b*. $\oplus$ désigne l'opérateur OU exclusif

2.9. Le moteur PSH-DB

PSH-DB, est la structure de données développée conjointement avec la fonction PSH, pour les phases d'expérimentations. La principale caractéristique qui a motivée son développement est le fait que la majorité des moteurs de bases de données clé-valeur existants ne proposent pas de stocker les clés sous la forme de chaînes binaires. Elles doivent donc être converties en Entiers Longs (encodées sous 64 bits), ce qui rend nativement les calculs de distance de

Hamming impossibles, sans devoir effectuer au préalablement des conversions de type de données. Elles peuvent être aussi gérées en tant que chaînes de caractères, ce qui se révèle non optimal en termes d'espace de stockage. PSH-DB a été pensé pour répondre précisément à ces besoins spécifiques. Son moteur a été développé autour d'un modèle client-serveur TCP multi-thread, permettant la connexion simultanée de plusieurs clients. Sa structure de données de type clé-valeur utilise la fonction PSH comme fonction de hachage. Les clés de hachage sont stockées nativement sous le type de données *bitset* proposé par la bibliothèque standard (*std*), du langage *C++11*, pour un stockage optimal des chaines binaires. Le modèle de données repose sur une structure de type *unordered_map* de la bibliothèque standard (std) du langage *C++11*. Cette structure a été choisie car elle permet de renvoyer les valeurs associées à une clé avec une complexité temporelle de *O(1)*.

Fonctions :	Descriptions :
set <*clé*> <*valeur*>	Associe un valeur à une clé
get <*clé*>	Retourne la valeur correspondante à une clé
get_hamming <*clé*> <*distance*>	Renvoie toutes les clés ayant une distance de Hamming inférieur à la distance et clé passé en paramètre
del <*clé*>	Supprime un coupe clé-valeur à partir de sa clé
dbsize	Renvoie le nombre total de clés
flushall	Supprime tout les couples clé-valeur

Tableau 2 : Liste des fonctions principales de PSH-DB

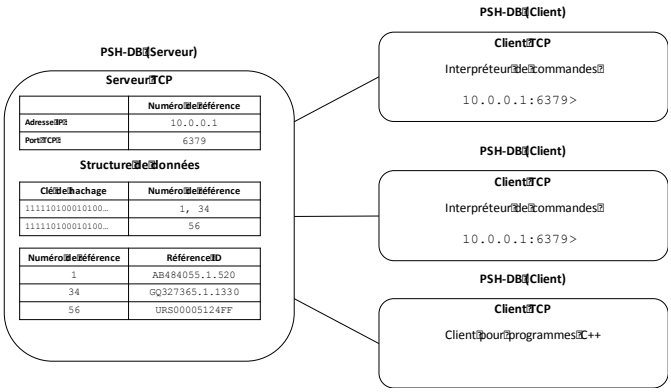


Figure 5 : Structure générale du système PSH-DB

2.9. Implémentation logicielle

La fonction PSH ainsi que le moteur PSH-DB ont été développés en *C++11* (compilateur *g++ 4.9*) avec la librairie *OpenCV 2.4*, utilisée pour la conversion des séquences en matrices de pixels ainsi que la fonction DCT. L'expérimentation a été réalisée sur un serveur tournant sous Ubuntu 14.04 et équipée d'un processeur *Intel Xeon CPU E5-4620 @ 2.20GH* et de 128Go de mémoire vive.

3. ÉVALUATION THÉORIQUE

Durant les phases de validations théoriques et expérimentations, les paramètres suivants ont été utilisés :

Taille de séquence :	1024 pb
Taille de sous-séquence :	128 pb
Taille de la matrice de pixels :	8x16
Taille de la matrice des coefficients :	8x16
Taille des clés de hachage	64 bits (8 x 8)

Tableau 2 : Liste des paramètres utilisés pour les expérimentations

Ces paramètres ont été sélectionnés pour être représentatifs des données utilisées en bio-informatique, où les séquences peuvent varier de plusieurs dizaines, à plusieurs milliers de paires de

bases. De plus, la taille des k-mers servant à calculer les clés de hachage a été fixée en fonction de la vitesse de calcul de la fonction PSH et de la diminution des données induite entre une séquence et sa clé de hachage. La fonction PSH prend alors en entrée une séquence de 128pb et génère une clé de 64 bits (8 octets), ce qui entraine une diminution théorique de 93,75%.

3.1. Distribution statistique des clés de hachage

L'un des éléments qui concourt à la validation d'une fonction de hachage est l'intervalle de valeur au sein duquel les clés de hachage sont générées et leurs distributions statistiques au sein de cet intervalle. Durant les phases d'expérimentations, nous avons paramétré la fonction PSH afin qu'elle renvoie des clés de hachage d'une taille de 64 bits. Les phases d'évaluation des distributions statistiques ont été réalisées en prenant en compte ce paramètre. Elles ont consisté à générer aléatoirement 500 000 séquences ADN, puis les clés de hachage correspondantes (cf. *Fig.6*).

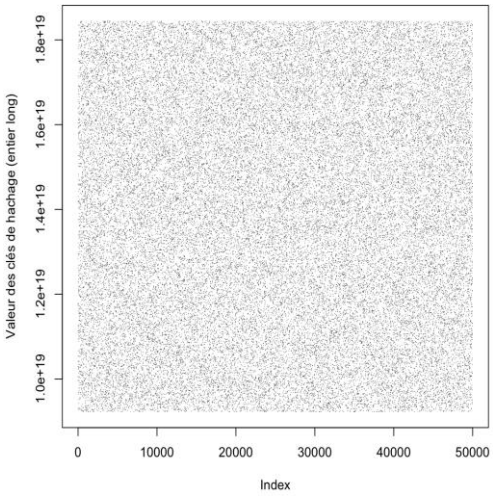


Figure 6 : Distribution statistique des valeurs de 50 000 clés de hachage générées aléatoirement.

3.2. Probabilité de collision

Lors du développement d'une fonction de hachage, qui par définition est injective, il est nécessaire de faire une étude de probabilité de collision [20]. Ceci permet de déterminer les limites induites par la taille des clés renvoyées. Le terme collision désigne pour une fonction de hachage, le fait d'obtenir une clé de hachage identique, mais calculée à partir de deux documents strictement différents. Cette probabilité est calculée via une loi de probabilité appelé le Paradoxe des Anniversaires [21]. La *Fig.7*, montre la probabilité d'avoir une collision en fonction d'un nombre de clés de hachage donné.

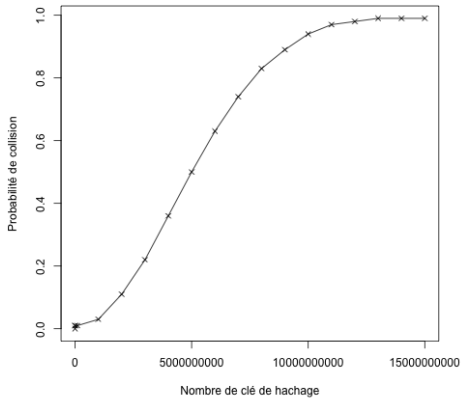


Figure 7 : Probabilité (en ordonnée), d'avoir une collision en fonction du nombre de clés de hachage (en abscisse).

3.3. Calcul théorique des Distances de Hamming en fonction des taux de mutation

Afin de déterminer les intervalles de distance de Hamming en fonction des taux de mutation entre deux séquences, une phase de simulation théorique a été réalisée. Pour cette phase de simulation, 500 millions de séquences ADN ont été générées aléatoirement (séquences primaires) et les clés de

hachage correspondantes ont été calculées avec la fonction PSH. À partir de ces séquences, 4 séquences divergentes, ainsi que les clés de hachages correspondantes ont été calculées avec des taux de 5%, 10% 20% et 30%. La distance de Hamming, entre les clés de hachage des séquences primaires et les clés de hachage des séquences divergentes dont elles sont issues, a ensuite été calculée. La *Fig.8*, présente la distribution des taux de mutation entre paires de séquences, en fonction des distances de Hamming.

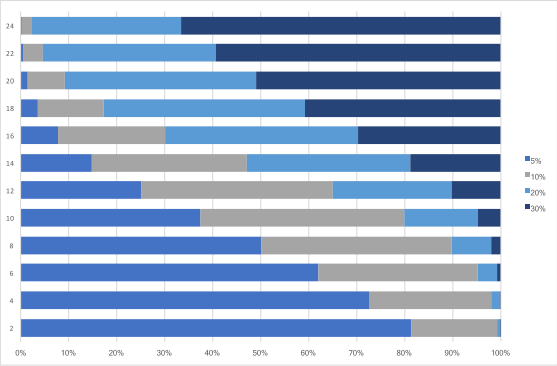


Figure 8 : Distribution des distances de Hamming (en ordonnée) en fonction du taux de mutation entre deux séquences (en abscisse).

4. ÉVALUATIONS

Nous présentons les expérimentations qui ont été réalisées afin d'évaluer l'utilisation de la fonction PSH pour indexer et rechercher des séquences nucléotidiques au sein de la structure PSH-DB.

L'objectif de ces différentes expérimentations était d'évaluer notre système par rapport à 3 aspects :

- Recherche de sous-séquences exactes
- Étude de la taille des données au sein de la structure PSH-DB
- Recherche de sous-séquences approchées

4.1. Recherche de sous-séquences exactes

La première expérimentation avait pour objectif d'évaluer la sensibilité et les temps d'exécution du système PSH-DB durant des tâches de recherche de sous-séquences exactes. Une base de données, regroupant 3 bases de données d'ARN de référence, a été constituée, pour un total de plus de 17 millions de séquences de référence et 14,6 Go de données (Cf. *Tab.1*). Toutes les séquences de référence ont été hachées et stockées au sein du moteur PSH-DB. L'objectif était de rechercher dans notre base de données des séquences de référence ayant au moins 128pb en commun avec un ensemble de requêtes composé de 2000 séquences.

Base de données :	Nombre de séquences :	Taille :
RDB (https://rdp.cme.msu.edu)	3 070 243	3,6 Go
SILVA RNA (https://www.arb-silva.de)	4 984 666	4,8 Go
RNA Central (https://rnacentral.org)	9 386 122	6,2 Go
TOTAL :	17 441 031	14,6 Go

Tableau 3 : Séquences de référence utilisées pour l'expérimentation n°1

Afin d'évaluer la sensibilité de notre système nous avons utilisé l'outil de référence en bio-informatique BLAST, que nous avons paramétré pour qu'il ne recherche que des séquences exactes de 128pb au sein de sa base de données. Néanmoins, il est important de relativiser les effets que pourraient avoir cette contrainte sur les résultats renvoyés par BLAST, car l'une des premières étapes de l'algorithmes de BLAST est de rechercher des sous-séquences exactes.

Par la suite, afin de comparer notre structure de données PSH-DB en termes de stockage et de temps d'exécution des requêtes, nous avons effectué la même expérimentation, mais en utilisant le moteur

NoSQL REDIS. Initialement, lors des développements méthodologiques de la fonction PSH, le moteur REDIS était utilisé en tant que structure de données. Ce moteur avait été choisi, car il proposait une solution libre, présentant une structure de type clé-valeur in-memory et des requêtes de type GET avec une complexité temporelle en O(1). Cependant, son incapacité à stocker des chaines binaires de façon optimale et l'impossibilité d'effectuer des requêtes de type Distance de Hamming sur les clés, ont motivé le développement du moteur PSH-DB, qui en reprend les caractéristiques principales et ajoute les caractéristiques recherchées.

4.2 Indexation des références

Pour procéder à l'indexation des séquences de références au sein de PSH-DB, chaque séquence a été découpée en sous-séquence ou k-mers, d'une taille de 128pb, avec un décalage de 64pb (cf. *Fig.9*). Pour indexer l'intégralité d'une séquence, dès lors que sa taille n'est pas un multiple de la taille des k-mers fixée en paramètre (128 pour cette expérimentation), il est nécessaire de définir un k-mer de fin de séquence. Ainsi, pour une séquence de taille de 530pb et des k-mers de 128, le k-mer de fin de séquence débutera à partir de la position 402 (530 – 128).

Pour chaque k-mer une clé de hachage correspondante a été calculée. C'est cette clé de hachage qui est stockée au sein de la structure PSH-DB.

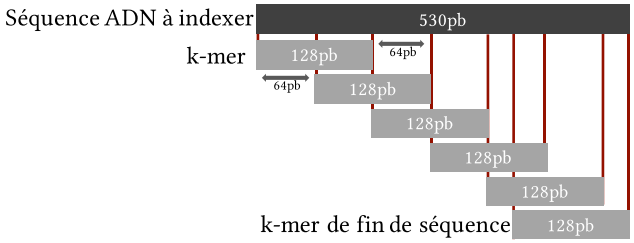


Figure 9 : Processus d'indexation d'une séquence de référence par découpage en k-mers

4.3. Création du jeu de requêtes

Pour cette expérimentation, un jeu de requêtes a été généré à partir des séquences issues de la base de données d'ARN de références. À partir des séquences de référence de la base de données, 1000 requêtes, d'une taille de 1024pb ont été extraites aléatoirement, et 1000 requêtes supplémentaires de même taille, ont été générées de façon aléatoire.

4.4. Interrogation de la base de données

Pour chacune des 2000 séquences requêtes, les différents k-mers possibles, d'une taille de 128pb, ont été calculés à partir du début et jusqu'à la fin de la séquence, avec un décalage d'une paire de bases. Pour une séquence, de taille W, le nombre de k-mers de taille K possible est donc de $W - K$. Les clés de hachage correspondantes aux différents k-mers ont été calculées via la fonction PSH. Ces clés de hachage ont été alors recherchées au sein de la base de données.

4.5. Paramétrage de BLAST

Pour cette expérimentation la version de BLAST 2.2.28+ a été utilisée. BLAST a été paramétré de façon à rechercher, des sous-séquences exactes d'une taille minimale de 128pb, avec 100% d'identité, entre les séquences requêtes et les séquences de références de la base de données. Concernant les résultats renvoyés, nous avons utilisé les paramètres par défaut où, BLAST renvoie le top 500 des séquences ayant un plus fort taux d'alignement. BLAST a aussi été paramétré pour exploiter 8 threads en parallèle. Afin de ne pas être pénalisée par les temps d'accès au disque dur, par rapport à PSH-DB, la base de données de BLAST a été placée sur un RAMDisk, de même que les fichiers de sortie de résultats.

4.6. Recherche de séquences proches

La seconde expérimentation que nous avons menée avait pour objectif d'évaluer la recherche de k-mers proches mais non exactes au sein de PSH-DB, en utilisant les propriétés de comparabilité des clés de hachage calculées par la distance de

Hamming. Une base de données, de 4 séquences d'ADN de référence, a été constituée (Cf. *Tab.4*). Ces séquences ont été choisies car elles ne présentaient aucune homologie. Elles ont été indexées et stockées au sein de PSH-DB en utilisant la méthode décrite en section 4.2. L'objectif était de calculer les distances de Hamming entre les clés de hachage issues d'une séquence requête et toutes les clés présentes dans la base de données. Ceci afin de déterminer s'il était possible de fixer un seuil en fonction des paramètres utilisés et au final de retrouver la séquence de référence dont était issue une séquence requête comportant des taux de mutation. Il est cependant important de noter que les taux de mutation ont été appliqués sur les séquences entières et non pas sur les k-mers. Ce qui signifie qu'en fonction des positions de mutation appliquées sur les séquences requêtes, les k-mers qui en sont issus, peuvent avoir des taux de mutation variables, compris entre 0% et 100%.

Nom de la séquence de référence :	Taille :
Borrelia burgdorferi B31, complete genome	910 724 pb
Escherichia coli O157:H7 str. Sakai DNA, complete genome	5 498 450 pb
West Nile virus lineage 2, complete genome	10 962 pb
Salmonella enterica subsp. enterica serovar Typhimurium str. LT2 chromosome, complete genome	4 857 432 pb

Tableau 4 : Séquences de référence utilisées pour l'expérimentation n°2.

Durant cette expérimentation, nous nous sommes plus particulièrement focalisé sur la sensibilité de la méthode de recherche. Les temps de traitement présentés *Tab.8*, ne sont donnés qu'à titre indicatif et ne peuvent pas être considérés comme représentatifs. En effet, l'expérimentation a consisté à comparer toutes les clés issues des différents k-mers des séquences requêtes avec toutes les clés de la base de données, ceci afin de pouvoir établir des statistiques sur les données obtenues.

4.7. Création du jeu de requêtes

Pour cette seconde expérimentation, 6000 séquences requêtes, d'une taille de 1024pb ont été utilisées. Le *Tab.5* présente le nombre de séquences requêtes qui ont été extraites à partir des séquences de références et les taux de mutation qui leur ont été appliqués. Il est à noter qu'une des séquences de référence n'a pas été utilisée pour extraire des requêtes et que nous avons ajouté 1000 requêtes générées totalement aléatoirement.

Séquences de références :	Nombre de requêtes :	Taux de mutation :
Borrelia burgdorferi	1000	5%
Borrelia burgdorferi	1000	10%
Escherichia coli	1000	5%
Escherichia coli	1000	10%
WestNile	1000	5%
WestNile	1000	10%
Salmonella enterica	0	-
Séquences aléatoires	1000	-

Tableau 5 : Séquences de référence utilisées pour l'expérimentation n°2

5. RESULTATS

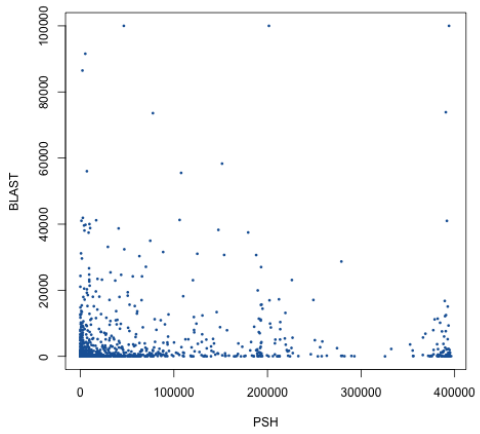


Figure 10 : Comparaison du nombre de séquences de référence renvoyées par PSH (en abscisse) et BLAST (en ordonnée)

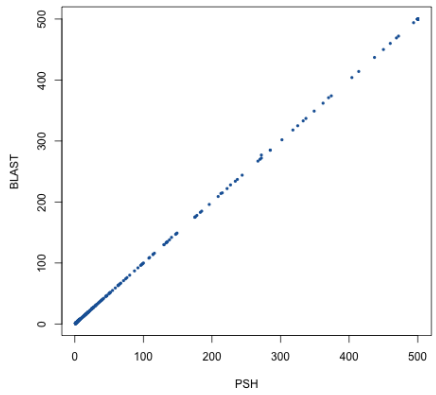


Figure 11 : Nombre de résultats renvoyés par PSH (en abscisse), communs au top 500 des résultats de BLAST (en ordonnée)

	PSH-DB	BLAST	Taux :
Nombre moyen de résultats renvoyés par requête	745 624	4 384	-
Résultats avec requêtes aléatoires	0	0	0%
Résultats commun	291 047	319 951	90,96%
Résultats communs (top 500 BLAST)	53 804	53 819	99,97%

Tableau 6 : Synthèse des résultats renvoyés par BLAST et PSH-DB

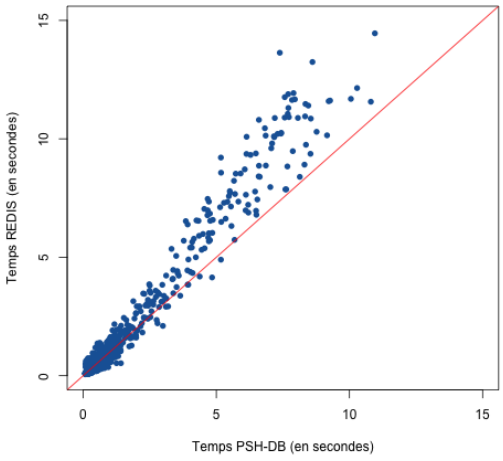


Figure 12 : Temps d'exécution de PSH-DB (en abscisse) face à REDIS (en ordonnée)

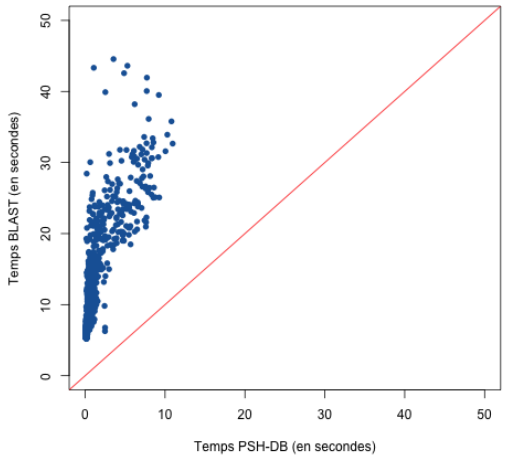


Figure 13 : Temps d'exécution de PSH-DB (en abscisse) face à BLAST (en ordonnée)

	BLAST :	PSH-DB :	REDIS :
Taille des bases de données	6 Go	4 Go	12 Go
Temps d'indexation	35 min	153 min	181 min
Temps de recherche	198 min	21 min	24 min

Tableau 7 : Taille des structures de données et des temps d'exécution entre BLAST, PSH-DB et REDIS

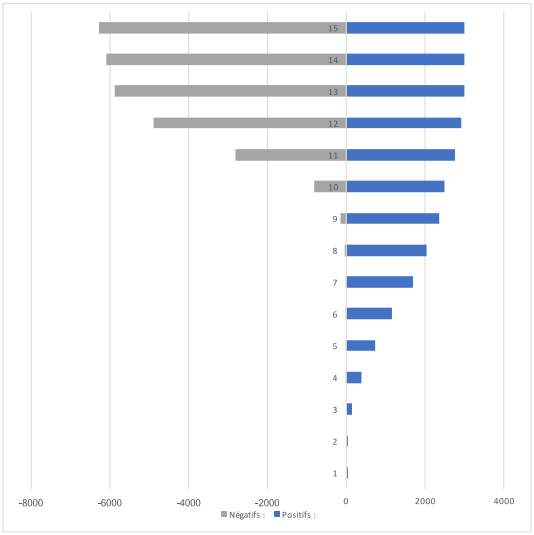


Figure 14 : Distribution du nombre de requêtes (positives ou négatives) en fonction de la Distance de Hamming avec des séquences requêtes ayant un taux de 10% de mutation

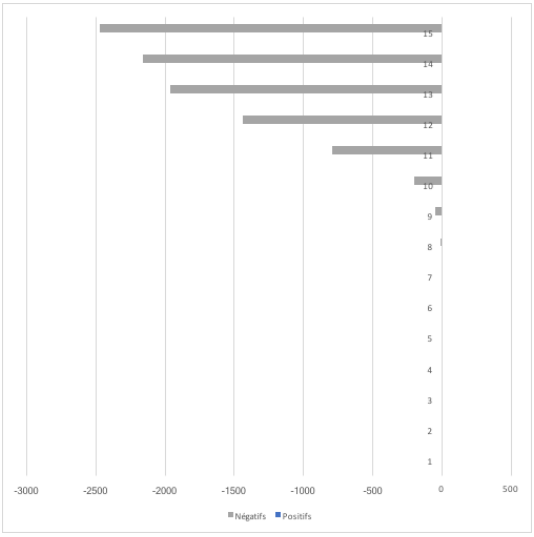


Figure 15 : Distribution des requêtes issues des séquences générées aléatoirement, en fonction de la distance de Hamming.

Taux de mutation	Positifs (seuil < 8)	Négatifs (seuil < 8)	Temps d'exécution
5%	100%	2,53%	486 min
10%	99,33	2,83%	492 min

Tableau 8 : Synthèse des résultats renvoyés par PSH-DB en fonction du taux de mutation.

6. DISCUSSION

Les différentes phases d'expérimentation présentées, avaient pour objectif d'évaluer l'utilisation de la fonction PSH et de sa structure de données PSH-DB, d'un point de vue théorique avec des données réelles. Elles ont été réalisées en utilisant les paramètres présentés *Tab.2*.

6.1. Évaluations théoriques

Concernant les évaluations théoriques, les résultats présentés montrent que les clés de hachage calculées par la fonction PSH en fonction des séquences ADN ont une distribution statistique homogène et qu'aucun intervalle n'est surreprésenté ou sous-représenté (Cf. *Fig.6*). Cette caractéristique est essentielle pour une fonction de hachage et permet de garantir l'uniformité des probabilités de collision. D'autre part l'étude de probabilité de collision (Cf. *Fig.7*), a démontré qu'en renvoyant des clés de hachage de 64 bits, PSH avait une probabilité de 50% de générer deux clés identiques à partir de deux séquences différentes pour 5 milliards de clés de hachage. Enfin au niveau théorique, l'étude de distribution statistique des distances de Hamming entre deux clés, en fonction des taux de mutations, a permis d'illustrer la propriété de représentativité de la fonction PSH. La *Fig.8*, montre une relation forte entre Distance de Hamming et taux de mutation des séquences dont elles sont issues. Néanmoins, une Distance de Hamming entre deux clés de hachage, ne permet pas précisément de déduire le taux exact de mutation.

6.2. Expérimentations à partir de données réelles.

La première expérimentation, qui consistait à rechercher des k-mers exacts au sein de la base de données, avait pour objectif d'évaluer la fonction PSH, sur des critères de sensibilité, de structure de données et de temps d'exécution.

En termes de sensibilité, la *Fig.10* et le *Tab.6* montre que PSH-DB renvoie en moyenne un nombre de résultats beaucoup plus important que BLAST. L'étude approfondie de ces résultats devra être menée afin de quantifier plus précisément les taux de positifs et les taux de faux positifs. Cependant, d'après l'étude de collision (cf. *Sec.3.2*), si deux clés de hachage sont identiques, la probabilité qu'elles soient issues de deux séquences différentes est très faible. Enfin, la *Fig.11*, qui présente le nombre de résultats communs entre PSH et les résultats de BLAST, démontre la capacité de PSH à détecter 99% des séquences du top 500 de BLAST. Cette première expérimentation a aussi été l'occasion de comparer la taille des bases de données entre les systèmes BLAST, REDIS et PSH-DB. Pour des structures de données in-memory, la taille des données est naturellement un critère

important. Le *Tab.7* montre que PSH-DB occupe le plus petit espace mémoire, suivi de BLAST et REDIS. Concernant BLAST, il est toutefois nécessaire de souligner que la base de données contient l'intégralité des séquences. REDIS, quant à lui, a été tributaire de son incapacité à gérer des clés au format *bitset* et du fait que l'espace mémoire occupé pour une clé, est de 50 octets au minimum, alors que dans notre cas les clés avaient une taille de 8 octets.

En ce qui concerne les temps d'exécution présentés *Tab.7*, les phases d'indexation de PSH-DB et REDIS s'avèrent être plus lentes que BLAST. Ceci peut s'expliquer par le fait que BLAST utilise une méthode d'indexation différente de PSH-DB, où le calcul de millions de clés de hachage a été nécessaire pour cette expérimentation. Au niveau des temps de recherche de k-mers exacts, PSH-DB et REDIS s'avèrent être plus rapides que BLAST, avec un léger avantage pour PSH-DB (cf. *Fig.13*). La *Fig.12*, montre la distribution des temps d'exécution entre PSH-DB et BLAST. Ceci s'explique en grande partie par la structure de type clé-valeur de ces deux systèmes où les requêtes de type GET ont une complexité temporelle constante. Les variations des temps de recherche de PSH-DB et REDIS sont principalement dues au nombre de résultats renvoyés par les serveurs aux clients. Il est aussi important de souligner que les algorithmes de recherche de PSH-DB et REDIS n'étaient pas parallélisés alors que BLAST pouvait exploiter 8 threads en parallèle.

La seconde expérimentation, avait pour objectif d'évaluer la possibilité de retrouver pour chaque séquence de notre panel, les séquences de référence les plus proches, malgré des taux de mutation compris entre 5% et 10%. Les résultats illustrés par la *Fig.14* montre la distribution cumulée totale des distances de Hamming, calculées à partir des séquences requête ayant été générées depuis les séquences de référence, avec un taux de mutation de 10%. La colonne « positifs » représente le nombre de clés de hachage appartenant aux séquences requête ayant été attribuées avec succès, à la séquence de référence dont elles étaient issues. La colonne « négatifs » représente les clés ayant été attribuées de façon erronée à une séquence de référence. En observant les distributions des distances de Hamming, issues de ces deux colonnes, il est possible de définir un intervalle significatif (distance de Hamming <= 8), permettant de déterminer la proximité entre deux clés de hachage, donc entre deux k-mers et par extension, une zone d'homologie entre deux séquences.

La *Fig.15*, montre la distribution cumulée totale des Distances de Hamming, calculée à partir des séquences générées aléatoirement. De façon identique à la *Fig.14*, nous pouvons observer qu'aucune des clés de hachage de ces séquences requête n'a eu de Distance de Hamming <= 8 avec les séquences de référence.

Enfin le *Tab.8* présente la synthèse de cette expérimentation. Les résultats présentés pour les jeux de séquences requête présentant des taux de mutation de 5% et 10% montrent des résultats très encourageants avec de forts taux de détection >99% et des taux de faux positifs compris entre 2% et 3%, avec un seuil de distance de Hamming établi à 8.

7. CONCLUSION

Durant les phases d'analyse des données génomiques, la recherche de sous-séquences au travers de bases de données de génomes de référence, est une tâche incontournable. Cet article présente PSH-DB, un système de type clé-valeur in-memory, couplé avec une fonction de hachage perceptuel (PSH), spécifiquement conçu pour l'indexation et la recherche de séquences ADN. Le développement de la fonction PSH, avait pour objectif d'évaluer la pertinence de l'utilisation de techniques non traditionnelles en bio-informatique, mais utilisées pour l'indexation des images numériques. Les résultats obtenus des différentes expérimentations sont très encourageants, ils démontrent que la fonction PSH permet d'indexer et de rechercher efficacement au sein d'une base de données de référence, des séquences ADN/ARN exactes ou approchées, comportant des taux de mutation. La structure de données PSH-DB, développée spécifiquement pour stocker des clés de hachage et effectuer des recherches via la distance de Hamming, a aussi démontré son efficacité en termes de rapidité d'accès aux données, mais aussi en termes d'optimisation de l'espace de stockage pour les clés binaires, face à un moteur clé-valeur tel que REDIS. Concernant la suite de ce travail, les prochaines étapes vont tout d'abord consister à optimiser la méthode de recherche par distance de Hamming au sein de PSH-DB, via l'implémentation de méthodes de recherche au sein de l'espace de Hamming [22]. Puis, de nouvelles expérimentations seront effectuées, afin de comparer PSH-DB avec d'autres outils en bio-informatique plus récents que BLAST, tels que KRAKEN [23] ou BWA [24]. Ces outils permettant, à partir d'une séquence requête de rechercher les séquences les plus proches au sein d'une base de données.

Enfin, par la suite nous évaluerons aussi l'utilisation de PSH-DB, pour l'indexation et la recherche de séquences protéiques.

INFORMATIONS COMPLÉMENTAIRES

L'intégralité des données ayant servi aux différentes expérimentations ainsi que les versions compilées de programmes, sont disponibles à l'adresse : http://epia.clermont.inra.fr/jgoer/PSH-DB

RÉFERENCES

[1] A. B. R. McIntyre, L. Rizzardi, A. M Yu, N. Alexander, G. L. Rosen, D. J. Botkin, S. E. Stahl, K. K. John, S. L. Castro-Wallace, K. McGrath, A. S. Burton, A. P. Feinberg and C. E. Mason. 2016. Nanopore sequencing in microgravity. npj Microgravity (2016) 16035

[2] M. Pop, S. L. Salzberg, 2008. « Bioinformatics challenges of new sequencing technology » Trends in Genetics, Volume 24, Issue 3, March 2008, Pages 142-149

[3] S. Schbath, V. Martin, M. Zytnicki, J. Fayolle, V. Loux, J-F. Gibrat. Mapping Reads on a Genomic Sequence: An Algorithmic Overview and a Practical Comparative Analysis. Journal of Computational Biology. June 2012, 19(6): 796-813. doi:10.1089/cmb.2012.0022

[4] P. Kalsi, H. Peltola, J. Tarhio. Comparison of Exact String Matching Algorithms for Biological Sequences. In: M. Elloumi, J. Küng, M. Linial, R.F. Murphy, K. Schneider, C. Toma. (eds) Bioinformatics Research and Development. Communications in Computer and Information Science, vol 13. Springer, Berlin, Heidelberg

[5] R. S. Boyer, J. S. Moore. A fast string searching algorithm. Communications of the ACM, 20(10):762–772, 1977

[6] S. Altschul, W. Gish, W. Miller, E. Myers, D. Lipman. Basic Local Alignment Search Tool. Journal of Molecular Biology, 215 :403-410

[7] N. Välimäki, E. Rivals (2013). Scalable and Versatile k-mer Indexing for High-Throughput Sequencing. Data. ISBRA 2013: 237-248

[8] L. Feng, A. Jean, C. P. Leng, L. Danbo. Identification of DNA Signatures via Suffix Tree Construction on a Hybrid Computing System. WSEAS Transactions on Biology and Biomedicine, Issue 2, Volume 9, April

[9] S. Misra, A. Agrawal, W-K. Liao, A. Choudhary. Anatomy of a hash-based long read sequence mapping algorithm for next generation DNA sequencing. Bioinformatics 2011; 27 (2): 189-195. doi: 10.1093/bioinformatics/btq648

[10] Wei Li, Yaduo Liu, and Xiangyang Xue. Robust audio identification for mp3 popular music. In Proceedings of the 33rd international ACM SIGIR Conference on Research and Development in Information Retrieval, pages 627–634, 2010.

[11] L. Chang, W. G. Yan, W. D. Wang. Research on Robust Image Perceptual Hashing Technology Based on Discrete Cosine Transform. In: M. Zhu (eds) Business, Economics, Financial Sciences, and Management. Advances in Intelligent and Soft Computing, vol 143. Springer, Berlin, Heidelberg

[12] Y. Ke, D. Hoiem, R. Sukthankar. Computer vision for music identification. 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), 2005, pp. 597-604 vol. 1.

[13] S. Prungsinchai, F. Khelifi, A. Bouridane, "DCT sign-based robust image hashing," 8th International Conference for Internet Technology and Secured Transactions (ICITST-2013), London, 2013, pp. 401-405

[14] K. R. Rao, P. Yip. Discrete Cosine Transform: Algorithms, Advantages, Applications (Academic Press, Boston, 1990)

[15] L. Yu, S. Sun (2006). Image robust hashing based on DCT signs, Intern. Conf. on Intelligent Information Hiding and Multimedia Signal Processing IIH-MSP, vol. 1, pp. 131-134, 2006. (Pubitemid 351337105)

[16] S. W. Smith. Chapter 8: The Discrete Fourier Transform. The Scientist and Engineer's Guide to Digital Signal Processing (Second ed.). San Diego, Calif.: California Technical Publishing

[17] I. Ito, H. Kiya. DCT Sign-Only Correlation with Application to Image Matching and the Relationship with Phase-Only Correlation. 2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07, Honolulu, HI, 2007, pp. I-1237-I-1240

[18] M. Curilem Saldías, F. Villarroel Sassarini, C. Muñoz Poblete, A. Vargas Vásquez, I. Maureira Butler. Image Correlation Method for DNA Sequence Alignment. PLOS ONE 7(6): e39221

[19] Richard Hamming error-detecting and error-correcting codes Bell System Technical Journal 29(2):147-160, 1950

[20] B. Schneier, Applied Cryptography, 2nd edition, John Wiley and Sons, Inc, 1996.

[21] Mohammad Peyravian, Allen Roginsky, Ajay Kshemkalyani, On probabilities of hash value matches, Computers & Security, Volume 17, Issue 2, 1998, Pages 171-176, ISSN 0167-4048,

[22] Qingmei Xiao, Motoyuki Suzuki and Kenji Kita. Fast Hamming Space Search for Audio Fingerprinting Systems. IZMIR2011 Conference, Miami

[23] Wood DE, Salzberg SL: Kraken: ultrafast metagenomic sequence classification using exact alignments. Genome Biology 2014, 15:R46

[24] Li H. and Durbin R. (2009) Fast and accurate short read alignment with Burrows-Wheeler Transform. Bioinformatics, 25:1754-60.