



République Algérienne Démocratique et Populaire



Ministère de l'Enseignement Supérieure et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Électronique et d'Informatique

Département Informatique

Mémoire de Master

Filière : Informatique

Spécialité :

Bio-informatique

**Thème**

---

**La recherche de similarité dans deux séquences d'ADN  
basée sur les k-mers**

---

**Soutenu le : 03/ 11 / 2020**

**Sujet Proposé par :**

Mme : IGHILAZA Chahra-zad

**Présenté par :**

FOUDILI Sirine

BAGHBAGHA Hadjer

**Devant le jury composé de :**

**Mme BOUKHEDOUMA** Présidente

**Mme SALMI** Membre

PFE N° (Spécialité/ N°/ Année) : Bio-informatique/011/2020

## Dédicaces

Je dédie ce modeste travail :

A ma mère chérie Nassira I., ma raison de vivre, la personne qui m'a donné naissance et qui m'a submergée de tendresse, ma première école où j'ai appris les plus importantes leçons de vie.

A mon premier héros, mon cher père Mustapha, mon premier professeur, qui avec ses conseils m'a guidé vers le bon chemin.

Ils sont les deux personnes qui sont toujours près pour moi et qui sont toujours là pour moi, ils n'hésitent jamais à m'encourager, ils garnissent mon chemin avec de l'espoir et ils continuent souvent à m'enrichir d'une force énorme pour continuer le combat pour arriver jusqu'au bout, ils ont su m'inculquer le sens de la responsabilité, de l'optimisme et de la confiance en soi face aux difficultés de la vie. Sans oublier la confiance qu'ils ont mis en moi et mes choix, leurs grands sacrifices, ils acceptent de rester loin l'un de l'autre seulement pour que ma mère m'accompagne à la cité universitaire et de rester plus de 1400km loin de mon père. Je leur dois ce que je suis aujourd'hui et ce que je serai demain et je ferai toujours de mon mieux pour rester leur fierté et ne jamais les décevoir. Que Dieu le tout puissant les préserve, leur accorde la santé, le bonheur, la quiétude de l'esprit et les protège de tout mal.

A mon cher frère Ahmed, mon bras droit, la personne qui prend ma main vers le chemin de joie avec, son encouragement, son humeur, son amour, il ne cesse jamais à m'aider.

A mes chères sœurs que j'aime Fatma et Sarah.

A mes chers neveux que j'aime, mes amours : Ferdous, Mehdi, Yassine, Yasser, Mohamed, Adem et le bébé attendu.

A mes très chères amies Rania Z et Sarah M.

A ma binôme Sirine F. et sa famille.

A toute la famille, les amis et les collègues qui m'ont soutenu d'une manière ou d'une autre.

**Hadjer.**

## **Dédicaces**

Je dédie ce travail à mes chers parents, qui m'ont tant soutenue et encouragée, et sans qui je n'aurais jamais pu réaliser ce projet ni réussi à le concrétiser. Bien qu'aucune dédicace ne pourrait refléter le respect et l'amour que j'ai pour vous, j'en profite de l'occasion pour vous exprimer ma profonde reconnaissance et ma considération pour les sacrifices que vous avez consenti pour mon instruction et mon bien-être.

À mon adorable sœur Afaf et à mon frère bien aimé Moujtaba. Je vous dédie ce travail en vous souhaitant un avenir radieux, plein de bonheur et de succès.

À tous les membres de ma famille, la famille Foudili et la famille Allaoui, dans l'espoir d'être à la hauteur de les rendre fiers de moi.

À mes chères amies Lina, Malek, Meriem, Sihem, Rayda, Wafa et Rahma pour leur soutien et leurs encouragements continus.

**Sirine.**

## Remerciements

En premier lieu, nous tenons à remercier le bon Dieu de nous avoir données le courage et la force pour faire ce travail et nous le prions toujours qu'il soit à nos côtés.

Nous tenons à exprimer toute notre reconnaissance à notre encadrante Mme « IGHILAZA Chahra-zad » : nous la remercions vivement de nous avoir encadrées, orientées, aidées et conseillées. Ses critiques constructives et ses remarques, nous ont appris à être plus rigoureuses dans notre travail, ce qui nous servira, sans aucun doute, dans nos futures vies professionnelles.

Nous présentons de tout cœur, nos remerciements les plus distingués aux membres de jury qui ont accepté de juger notre travail.

Nous adressons également nos sincères remerciements à tous les professeurs, intervenants pour leur disponibilité durant notre cursus universitaire.

Enfin, nous remercions toute personne qui par ses paroles, ses écrits, ses conseils ainsi que ses critiques a guidé nos réflexions, ainsi que nos collègues et amis qui nous ont soutenues dans la réalisation de ce mémoire.

*Hadjer et Sirine.*

## Résumé :

La comparaison de séquences d'ADN est une pratique fondamentale pour de nombreuses applications de biologie et l'une des branches les plus importantes de la bio-informatique, puisque l'analyse et le traitement des données moléculaires s'effectue, " in silico ", au moyen de calculs complexes informatisés.

Des méthodes classiques basées sur l'alignement ont prouvées leur efficacité pour la comparaison de séquences d'ADN. Cependant, elles sont coûteuses et s'adaptent mal aux données biologiques volumineuses. Or les banques d'ADN se développent en une croissance exponentielle. De ce fait, le recours à l'indexation est nécessaire.

Dans ce travail, nous exposons d'abord l'état de l'art biologique et informatique en liaison avec la recherche de similarité entre les paires de séquences d'ADN. Ensuite, nous traitons notre problématique, en utilisant une méthode de comparaison basée sur les k-mers et les structures d'index.

Le projet consiste aussi à implémenter les résultats de la conception dans un logiciel dédié à la comparaison de séquences d'ADN et qui offre également aux biologistes la possibilité d'appliquer les résultats obtenus au domaine de la phylogénie. Notre étude expérimentale effectuée sur des données réelles a montré que les méthodes basées sur les k-mers et l'indexation sont plus efficaces en pratique par rapport aux méthodes classiques en terme de temps de calcul.

**Mots-clés :** Bio-informatique, alignement, comparaison d'ADN, algorithmique du texte, comparaison sans alignement, k-mers, génomique, recherche approchée de motifs, structures d'index, filtres de Bloom, hachage, phylogénie.

## Abstract :

DNA sequence comparison is a fundamental practice for many applications of biology and one of the most important branches of bioinformatics, since the analysis and processing of molecular data is carried out, "in silico", via computer simulation.

Classical alignment-based methods have proven their effectiveness for DNA sequence comparison. Though, they are expensive and adapt poorly to large biological data. Yet, DNA data bases are developing exponentially. Therefore, the use of indexation is necessary.

In this work, we first expose the state of the art in biology and informatics in conjunction with the search for similarities between pairs of DNA sequences. Next, we deal with our problem, using a k-mer based comparison and index structures.

The project also involves the implementation of the design results in a software dedicated to the DNA sequences comparison and which also offers biologists the possibility of applying the results obtained to the field of phylogeny. Our experimental study of real data showed that the k-mers based and indexing methods are more efficient in practice compared to conventional methods in terms of computation time.

**Keywords:** Bio-informatics, alignment, DNA comparison, algorithmic text, free-alignment comparison, k-mers, genomics, approximate string matching, index structures, Bloom filters, hashage, phylogeny.

# Table de matières

Dédicaces .....	II
Dédicaces .....	III
Remerciements .....	IV
Résumé : .....	V
Abstract : .....	VI
Table de matières .....	VII
Table de figures .....	X
Table de tableaux .....	XII
Table des algorithmes.....	XIII
<b>Introduction générale.....</b>	<b>1</b>
<b>Chapitre 1 : Contexte Biologique.....</b>	<b>3</b>
1.1. L'ADN.....	3
1.2. Le génome et le gène .....	4
1.3. Le séquençage.....	4
1.4. La phylogénie .....	5
1.4.1. Les applications de la phylogénie .....	6
1.5. Les banques de données biologiques.....	7
1.5.1. Les banques de données généralistes .....	7
1.5.2. Les banques de données spécialisées .....	8
1.6. Conclusion .....	9
<b>Chapitre 2 : Algorithmique du texte .....</b>	<b>10</b>
2.1. Notations et définitions.....	10
2.2. Algorithmes de recherche exacte d'un motif.....	11
2.2.1. Algorithme naïf (fenêtre glissante) .....	12
2.2.2. Algorithme de Knuth-Morris-Pratt (KMP) .....	12
2.3. Algorithmes de recherche exacte multiple de motifs .....	14
2.3.1. Algorithme de Rabin-Karp.....	14
2.3.2. Algorithme d'Aho-Corasick.....	16
2.4. Algorithmes de recherche approchée de motifs .....	17
2.4.1. Distance de Hamming .....	18
2.4.2. Distance de Levenshtein.....	18
2.4.3. Algorithme de Needleman-Wunsch .....	20
2.4.4. Algorithme de Smith –Waterman .....	21
2.5. Structure d'index .....	23

2.5.1.	Les tables de suffixes .....	23
2.5.2.	Les arbres de suffixes .....	24
2.5.3.	FM-index .....	25
2.5.4.	Les filtres de Bloom .....	27
2.6.	Conclusion .....	28
<b>Chapitre 3 : Recherche de similarité à base de k-mers .....</b>		<b>29</b>
3.1.	Comparaison des séquences d'ADN .....	29
3.1.1.	La comparaison par alignement .....	30
3.1.2.	La comparaison sans alignement.....	31
a)	Les méthodes basées sur l'évaluation de l'information : .....	31
b)	Les méthodes basées sur les fréquences de sous-séquences d'une longueur définie : .....	31
3.2.	Notre stratégie de comparaison .....	32
3.2.1.	Méthode de programmation dynamique .....	32
3.2.2.	Méthode heuristique .....	33
3.2.2.1.	Comparaison et indexation des k-mers avec dictionnaires .....	34
3.2.2.1.1.	Comptage de k-mers .....	35
3.2.2.1.2.	Vecteur de k-mers .....	37
3.2.2.1.3.	Calcul de distance .....	38
3.2.2.2.	Comparaison et indexation des k-mers avec les filtres de Bloom .....	39
3.2.2.2.1.	Comptage de k-mers par les filtres de Bloom.....	42
3.2.2.2.2.	Calcul de distance .....	44
3.3.	Analyse phylogénétique .....	45
3.3.1.	Méthode UPGMA .....	45
3.4.	Conclusion.....	47
<b>Chapitre 4 : Mise en œuvre et expérimentation .....</b>		<b>48</b>
4.1.	Ressources matérielles et logicielles .....	48
4.1.1.	Ressources matérielles .....	48
4.1.2.	Ressources logicielles .....	48
4.1.2.1.	Langage de programmation Python .....	48
4.1.2.2.	Framework Flask .....	50
4.2.	Architecture de l'application .....	50
4.3.	Présentation de l'application .....	51
4.3.1.	Lecture de données.....	51
4.3.2.	Affichage de l'alignement.....	52
4.3.3.	Détails de la comparaison .....	53
4.3.4.	Affichage de l'arbre phylogénétique.....	55



4.3.5. Les tests de fonctionnement .....	55
4.4. Expérimentations.....	57
4.4.1. Benchmarks .....	57
4.4.2. Discussion .....	58
4.5. Conclusion.....	64
<b>Conclusion générale et perspectives .....</b>	<b>66</b>
Références bibliographiques .....	68

## Table de figures

<b>Figure 1.1:</b> La structure d'une cellule. ....	3
<b>Figure 1.2:</b> Le séquençage automatique de l'ADN. ....	5
<b>Figure 1.3:</b> Exemple d'un résultat d'un séquençage. ....	5
<b>Figure 1.4:</b> Exemple d'un arbre phylogénétique. ....	6
<b>Figure 2.1:</b> Exemple de l'utilisation de l'algorithme naïf. ....	12
<b>Figure 2.2:</b> Exemple de l'algorithme KMP. ....	13
<b>Figure 2.3:</b> Exemple d'un automate d'Aho-Corasick. ....	17
<b>Figure 2.4:</b> Exemple de la distance de Levenshtein. ....	19
<b>Figure 2.5:</b> Exemple d'un alignement global maximal. ....	20
<b>Figure 2.6:</b> Exemple d'un alignement local. ....	22
<b>Figure 2.7:</b> Exemple d'un arbre de suffixes. ....	24
<b>Figure 2.8:</b> Exemple de filtre de Bloom. ....	27
<b>Figure 3.1:</b> Exemple de décomposition d'une séquence d'ADN en 7-mers. ....	34
<b>Figure 3.2:</b> Schématisation de la comparaison et indexation des k-mers avec dictionnaires. ....	35
<b>Figure 3.3:</b> Exemple d'indexation à base de k-mers. ....	36
<b>Figure 3.4:</b> Exemple d'un filtre de Bloom. ....	39
<b>Figure 3.5:</b> Exemple d'un faux positif dans un filtre de Bloom. ....	40
<b>Figure 3.6:</b> Schématisation de la comparaison et l'indexation des k-mers avec filtres de Bloom. ....	42
<b>Figure 3.7:</b> Exemple de la 1ère itération lors d'une création d'un vecteur de k-mers. ....	44
<b>Figure 3.8:</b> Exemple d'application de la méthode UPGMA (itération 1). ....	46
<b>Figure 3.9:</b> Exemple d'application de la méthode UPGMA (itération 2). ....	46
<b>Figure 3.10:</b> Exemple d'un arbre phylogénétique. ....	46
<b>Figure 4.1:</b> Structure du micro framework Flask. ....	50
<b>Figure 4.2:</b> Architecture de l'application. ....	51
<b>Figure 4.3:</b> Lecture de données via l'application. ....	52
<b>Figure 4.4:</b> Résultat d'un alignement par paire en utilisant Needleman-Wunsch. ....	52
<b>Figure 4.5:</b> Visualisation d'une comparaison de séquences à base de k-mers. ....	53
<b>Figure 4.6:</b> Matrice de scores résultante de la comparaison de 4 séquences d'ADN. ....	53
<b>Figure 4.7:</b> Présentation d'une partie des détails d'un alignement de séquences. ....	54
<b>Figure 4.8:</b> Présentation de l'arbre phylogénétique. ....	55
<b>Figure 4.9:</b> Erreur générée lorsque aucun fichier ni aucune séquence ne sont introduits. ....	55
<b>Figure 4.10:</b> Erreur générée si le format du fichier n'est pas autorisé. ....	56
<b>Figure 4.11:</b> Erreur générée si le format de séquences n'est pas valide. ....	56
<b>Figure 4.12:</b> Erreur générée si le nombre de séquences saisies est insuffisant. ....	57
<b>Figure 4.13:</b> Variation du temps de réponse selon le paramètre k pour le fichier d'Influenza_H1N1 avec les 3 méthodes de comparaison. ....	60
<b>Figure 4.14:</b> Variation du temps de réponse selon le paramètre k pour le fichier d'Orchid avec les 3 méthodes de comparaison. ....	61
<b>Figure 4.15:</b> Variation du temps de réponse selon le paramètre k pour le fichier d'Hantavirus avec les 3 méthodes de comparaison. ....	61
<b>Figure 4.16:</b> Variation du temps de réponse selon le paramètre k pour le fichier d'HIV avec les 3 méthodes de comparaison. ....	62
<b>Figure 4.17:</b> Variation du temps de réponse selon le paramètre k pour le fichier d'Human_Viruses avec les 2 méthodes de comparaison. ....	62

**Figure 4.18:** Impact de la taille des k-mers et des jeux de données sur les performances. .... 64

## Table de tableaux

<b>Tableau 2.1:</b> Codage de l'alphabet nucléotidique. ....	15
<b>Tableau 2.2:</b> Exemple d'une table de suffixes. ....	23
<b>Tableau 3.1:</b> Matrice de scores résultante d'une comparaison de 3 séquences d'ADN. ....	38
<b>Tableau 3.2:</b> Exemple d'application de la méthode UPGMA (itération 0). ....	46
<b>Tableau 4.1:</b> Les caractéristiques techniques des machines utilisées. ....	48
<b>Tableau 4.2:</b> Modules utilisés pour le développement de l'application. ....	49
<b>Tableau 4.3:</b> Description de fichiers utilisés. ....	57
<b>Tableau 4.4:</b> Performances des méthodes de comparaison en fonction du paramètre k et de la taille des jeux de données. ....	59

## Table des algorithmes

<b>Algorithme 2.1</b> : Recherche Naïve .....	12
<b>Algorithme 2.2</b> : Recherche KMP .....	13
<b>Algorithme 2.3</b> : Recherche-Rabin-Karp .....	16
<b>Algorithme 2.4</b> : Aho-Corasick .....	17
<b>Algorithme 2.5</b> : Dist-Levenshtein .....	19
<b>Algorithme 2.6</b> : Needleman-Wunsch .....	21
<b>Algorithme 2.7</b> : Smith-Waterman .....	22
<b>Algorithme 3.1</b> : Constuction_dictionnaire.....	37

## Introduction générale

La bio-informatique est un domaine de recherche qui analyse et interprète des données biologiques, au moyen de méthodes informatiques, afin de créer de nouvelles connaissances en biologie.

On peut distinguer deux aspects de la bio-informatique :

- Aspect analyse de données : Application des algorithmes et modèles statistiques dans l'objectif d'interpréter, classer, comprendre des données biologiques.
- Aspect développement : Développement des modèles mathématiques et outils associés pour résoudre des problèmes biologiques.

Ce domaine a plusieurs champs d'application tels que :

- La bio-informatique statistique et la bio-informatique des populations.
- La bio-informatique des réseaux, qui s'intéresse aux interactions entre gènes, protéines, cellules, organismes.
- La bio-informatique structurale, qui traite de la reconstruction, de la prédiction ou de l'analyse de la structure 3D ou du repliement des macromolécules biologiques (protéines, acides nucléiques).
- La bio-informatique des séquences, qui traite de l'analyse de données issues de l'information génétique contenue dans la séquence de l'ADN ou dans celle des protéines qu'il code. Cette branche s'intéresse en particulier à l'identification des ressemblances entre les séquences, à l'identification des gènes ou de régions biologiquement pertinentes dans l'ADN ou dans les protéines, en se basant sur l'enchaînement ou séquence de leurs composants élémentaires (nucléotides, acides aminés).

Notre travail fait partie de la bio-informatique des séquences, il s'agit d'une réalisation d'un système pour la recherche de similarité entre deux séquences d'ADN en utilisant les k-mers.

Le terme k-mer fait généralement référence à toutes les sous-chaînes de longueur k qui sont contenues dans une chaîne de caractères.

En génomique computationnelle, les k-mers font référence à toutes les sous-séquences (de longueur k) à partir d'une lecture obtenue par séquençage de l'ADN. Les k-mers sont généralement utilisés lors de l'assemblage de séquences, mais peuvent également être utilisés afin d'estimer la distance de similarité entre les séquences.

Une mesure de similarité simple entre deux séquences peut être leur pourcentage de k-mers en commun. L'avantage des k-mers est qu'ils s'indexent et se comparent extrêmement rapidement.

L'objectif de ce projet est d'améliorer le processus de comparaison de deux séquences en comparant les k-mers., qui est traditionnellement effectué grâce à des techniques d'alignement coûteuses. Il est aussi proposé d'utiliser les filtres de Bloom comme structure d'indexation car ils ont l'avantage d'être très économes en mémoire et très rapides en exécution.

Après avoir exposé les différentes approches de comparaison, nous allons utiliser les résultats obtenus pour les appliquer au domaine biologique de la phylogénie, et ce dans le but de résoudre divers problèmes ayant un rapport avec les liens de parenté entre les différentes familles de gènes.

Pour décortiquer le thème choisi, nous avons adopté un plan de quatre chapitres :

- ❖ Le 1er chapitre intitulé : Contexte biologique. Dans ce chapitre, nous avons donné quelques définitions des termes biologiques utilisés en bio-informatique tels que : l'ADN, le génome... etc., ainsi que les bases de données biologiques.
- ❖ Le 2ème chapitre intitulé : Algorithmique du Texte. Ce chapitre est dédié à l'étude de quelques algorithmes utiles dans le domaine de la recherche de motifs.
- ❖ Le 3ème chapitre intitulé : Recherche de similarité à base de k-mers. Nous présentons dans ce chapitre notre contribution pour la comparaison de séquences afin de trouver les liens de parenté.
- ❖ Le 4<sup>ème</sup> chapitre intitulé : Implémentation et analyse des résultats. Ce chapitre est une description de la réalisation de notre projet, les outils utilisés pour développer notre application et ses fonctionnalités ainsi qu'une analyse des résultats obtenus.

# Chapitre 1

## Contexte Biologique

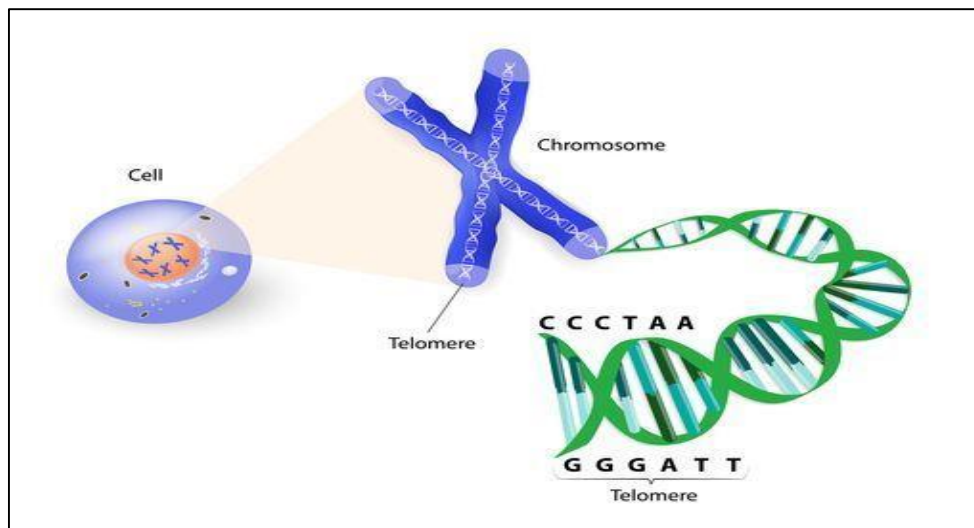
Nous abordons dans ce chapitre les principaux termes et notions biologiques utilisés tout au long de ce mémoire, ainsi le lecteur aura un aperçu des bases biologiques nécessaires en bio-informatique et indispensables à la compréhension du problème traité.

### 1.1. L'ADN

L'ADN (pour acide désoxyribonucléique) est le support de l'hérédité. Cette molécule présente chez tous les êtres vivants est le support universel de l'information génétique.

Elle est constituée de deux chaînes parallèles enroulées l'une autour de l'autre. On parle de la double hélice d'ADN. Chaque chaîne, ou brin d'ADN, est composé d'un enchaînement d'éléments appelés nucléotides (*Robin, 2019*).

Il existe quatre nucléotides différents : l'adénine (A), la thymine (T), la cytosine (C) et la guanine (G).



**Figure 1.1:** La structure d'une cellule (*Robin, 2019*).

En bio-informatique, les séquences nucléiques sont représentées par des chaînes de caractères sur l'alphabet à quatre lettres A, T, G, C. Un cinquième caractère (N) est défini pour tenir compte des erreurs de séquençage.



La connaissance des séquences nucléiques conduit en particulier à la localisation des gènes, à la déduction de séquences en acides aminés des protéines codées par ces gènes, à la contribution à l'analyse moléculaire de leurs expressions et régulations, ainsi qu'à la mise en évidence de mutations ou modifications à l'origine des maladies.

## 1.2. Le génome et le gène

Le génome est l'ensemble du matériel génétique d'un organisme. Il contient à la fois les séquences codantes, c'est-à-dire celles qui codent pour des protéines, et les séquences non codantes. Chez la majorité des organismes, le génome correspond à l'ADN présent dans les cellules.

Le gène est quant à lui un petit morceau d'ADN qui correspond à une information génétique spécifique, on peut le définir telle une fraction de l'ADN.

Il est décrit en énumérant l'ordre de succession des bases A, T, C et G, qui ne se succèdent que par ensembles de trois « *lettres* ». Ces successions, ces " mots " de trois lettres sont nommés « *codons* ».

Les gènes indiquent à chaque cellule son rôle dans l'organisme. Sur leur ordre, elles synthétisent des protéines, chaque protéine a un rôle différent à jouer.

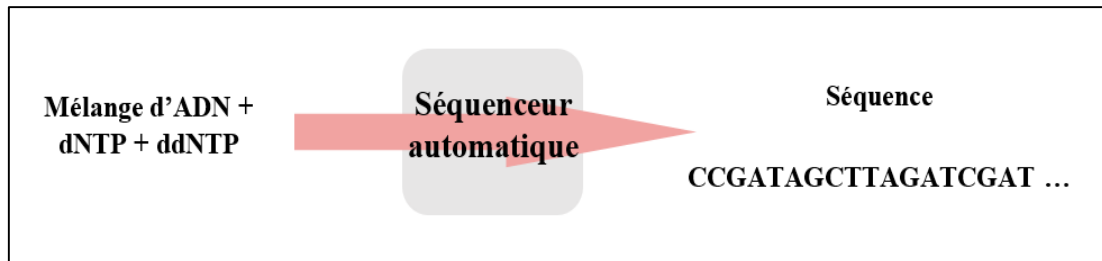
Par exemple, l'hémoglobine est la protéine qui sert à transporter l'oxygène dans le sang. D'autres protéines encore, interviennent pour définir la couleur des yeux ou la forme du visage.

Une anomalie génétique (mutation ou anomalie chromosomique) peut perturber la fabrication des protéines. Elle donne en quelque sorte de « *mauvais ordres* » pour les fabriquer avec pour conséquence absence de fabrication, excès de fabrication ou fabrication anormale. La protéine ne peut donc plus jouer son rôle ce qui engendre une maladie génétique (*Courrèges, 2014*).

## 1.3. Le séquençage

Le séquençage signifie la détermination de la succession des bases nucléotides A, C, G et T prenant part à la structure de l'ADN. C'est un outil essentiel en biologie moléculaire tant en médecine que dans de nombreuses autres disciplines des sciences de la vie.

Les connaissances acquises grâce à cette méthode et la possibilité de séquencer des génomes de grande taille, tel que le génome humain, ont amené les chercheurs à développer des techniques de séquençage de plus en plus sophistiquées. La très grande majorité des séquences réalisées et publiées aujourd'hui sont réalisées sur des séquenceurs automatiques. Ceux-ci sont capables de réaliser les réactions de séquence, puis de les lire (*Lamoril et al., 2008*).

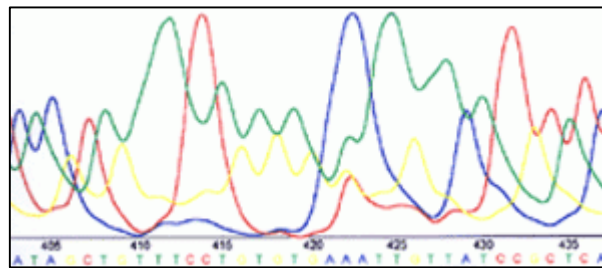


**Figure 1.2:** Le séquençage automatique de l'ADN.

**ddNTP :** Le didésoxyribonucléotide est un nucléotide marqué avec un fluorochrome propre, utilisé pour déduire la succession des différentes bases nucléiques.

**dNTP :** Le désoxyribonucléoside triphosphate, utilisé pour l'amplification de l'ADN.

Le résultat du séquençage est présenté par la machine sous forme de courbes présentant la fluorescence détectée, et l'interprétation qui en est faite en termes de nucléotides.



**Figure 1.3:** Exemple d'un résultat d'un séquençage (Michel & Gilles, 2004).

Les quatre courbes de la figure 3 correspondent à la détection de la fluorescence des fragments d'ADN obtenus. Un pic correspond donc à la détection d'un nucléotide donné dans la séquence (Michel & Gilles, 2004).

La lecture des séquences permet d'étudier l'information biologique contenue par celle-ci. Étant donné l'unicité et la spécificité de la structure de l'ADN chez chaque individu, la séquence de l'ADN permet de nombreuses applications dans le domaine de la médecine, comme, par exemple, le diagnostic, les études génétiques, l'étude de paternité, la criminologie, la compréhension de mécanismes physiopathologiques, la synthèse de médicaments, les enquêtes épidémiologiques.

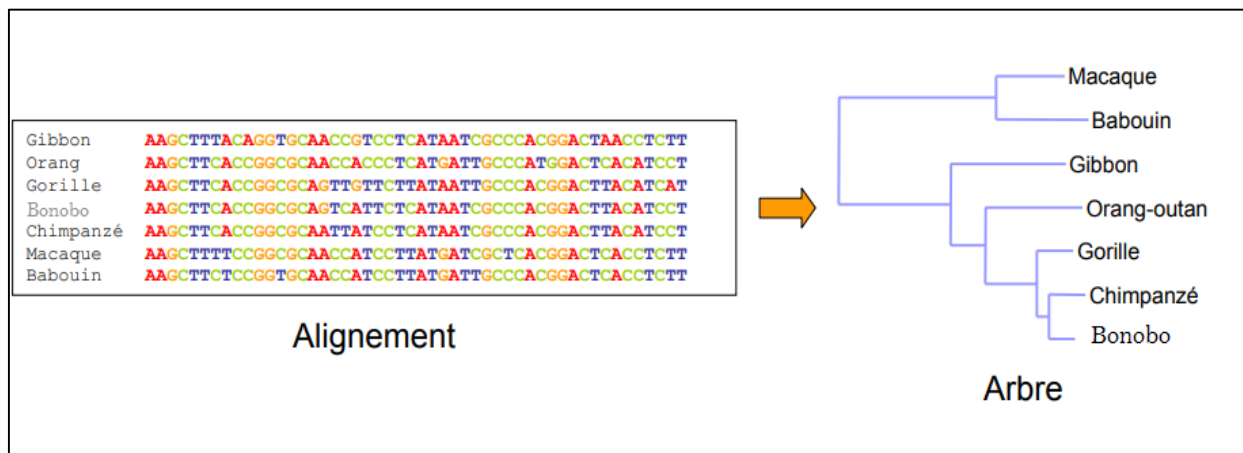
Le séquençage est de fait un remarquable instrument nécessaire à la compréhension des cycles de la vie dans leur globalité.

## 1.4. La phylogénie

La phylogénie moléculaire est un domaine central de la biologie, son but est de reconstruire les relations de parenté entre des séquences de nucléotides ou d'acides aminés. La relation est présentée sous la forme d'un arbre avec une racine unique, les feuilles de l'arbre sont actuellement des espèces observables appelées taxons. Les branches représentent la relation « *est l'ancêtre de* » entre les nœuds.

Cette discipline permet d'étudier l'évolution du génome. En particulier, pour chaque famille multigénique, on peut déterminer l'importance relative des événements de duplications et de transferts horizontaux de gènes. La fiabilité des méthodes de reconstruction phylogénétique repose sur la compréhension des mécanismes d'évolution des séquences (*Philippe et al., 2002*).

L'étude des contraintes fonctionnelles agissant sur les protéines bénéficie de ces avancées. En particulier, la détection, dans une protéine, des positions qui sont soumises à une sélection est devenue assez performante, permettant de prédire les substitutions à l'origine d'un changement de fonction et donc de guider les études expérimentales.



**Figure 1.4:** Exemple d'un arbre phylogénétique (*Claverie, 2006*).

### 1.4.1. Les applications de la phylogénie

Les progrès du séquençage ont amené un flot de données qui se prêtent particulièrement bien à l'analyse phylogénétique. L'utilisation simultanée de nombreux gènes, permettant d'obtenir plusieurs milliers de positions homologues, a permis de résoudre plusieurs questions débattues de longue date.

La phylogénétique peut être utilisée dans plusieurs domaines, par exemple :

- La bio-écologie : pour trouver le déplacement des espèces et étudier les relations hôtes-parasites.
- L'épidémiologie : pour tracer l'évolution d'un virus à travers ses différentes souches.
- La génomique fonctionnelle : pour comprendre les mécanismes de l'évolution moléculaire.

## 1.5. Les banques de données biologiques

Les banques de données biologiques consistent en une collection de fichiers structurés répertoriant des informations venant de divers champs de recherche en biologie tels que la génomique<sup>1</sup>, la protéomique<sup>2</sup>, la métabolomique<sup>3</sup>, la phylogénétique et les puces à ADN. (*Bolser et al., 2012*)

Elles regroupent les données produites principalement par le séquençage des génomes (séquences primaires, structures moléculaires, cartographie, collection de souches ou de clones...etc.).

Cette connaissance facilite la prise en charge des pathologies, permet la création de nouveaux médicaments et permet la découverte de relations inter-espèces au cours de l'histoire de la vie.

Les banques de données sont très variées dans leur volume et leur nature, on distingue deux types principaux :

### 1.5.1. Les banques de données généralistes

Ces banques contiennent des données hétérogènes, Elles offrent ainsi la collecte la plus exhaustive possible. Leur principale mission est de rendre publiques les séquences qui ont été déterminées, celles-ci sont souvent accompagnés par leurs informations (annotations, expertise, bibliographie).

Les bases de données généralistes sont indispensables à la communauté scientifique car elles regroupent la majorité des séquences connues en un seul ensemble ce qui présente un élément fondamental pour la recherche de similitudes avec une nouvelle séquence. De plus, la présence de références à d'autres bases permet d'avoir accès à d'autres informations non répertoriées.

Enfin, la grande diversité d'organismes qui y est représentée permet d'aborder des analyses de type évolutif (*Fondrat, 1997*).

---

<sup>1</sup> La génomique : Branche de la génétique qui étudie le génome.

<sup>2</sup> La protéomique : La science qui étudie les protéomes, c'est-à-dire l'ensemble des protéines d'une cellule, d'un organite, d'un tissu, d'un organe ou d'un organisme à un moment donné et sous des conditions données.

<sup>3</sup> La métabolomique : Une science qui étudie l'ensemble des métabolites primaires (sucres, acides aminés, acides gras, etc.) et des métabolites secondaires dans le cas des plantes.

Exemples de banques de données généralistes :

- NCBI ("National Center for Biotechnology Information").  
<https://www.ncbi.nlm.nih.gov/>
- EBI ("European Bioinformatics Institute").  
<https://www.ebi.ac.uk/>
- Uniprot.  
<https://www.uniprot.org/>
- PDB ("Protein Data Bank").  
<https://www.rcsb.org/>

**1.5.2. Les banques de données spécialisées**

Des banques de données dédiées à des besoins spécifiques liés à l'activité d'un groupe de personnes, ou bien créées au sein des laboratoires.

Elles contiennent des données homogènes d'intérêt très divers, leur principal but est de recenser des familles de séquences autour de caractéristiques biologiques précises comme les signaux de régulation, les promoteurs de gènes, les signatures peptidiques ou les gènes identiques issus d'espèces différentes (*Fondrat, 1997*).

Très souvent ces bases correspondent à des améliorations ou à des regroupements par rapport aux données issues des bases généralistes.

Exemples de banques de données spécialisées :

- SGD ("Génome des Saccharomyces").  
<https://www.yeastgenome.org/>
- MGI ("Génome de la souris").  
<http://www.informatics.jax.org/>
- Transfac ("Facteurs de transcription").  
<http://gene-regulation.com/pub/databases.html#transfac>
- KABATP ("Séquences d'immunoglobines").  
<http://www.bioinf.org.uk/abs/kabatman.html>

## **1.6. Conclusion**

Dans ce chapitre, nous avons défini les concepts biologiques de base qui représentent le centre de notre analyse, tels que l'ADN ainsi que les techniques permettant de l'extraire et de la traiter. Ensuite, nous avons mis le point sur la phylogénie, un domaine biologique qui bénéficie de l'alignement de séquences, auquel nous reviendrons avec plus de détails dans les chapitres suivants. Enfin, nous avons présenté les banques de données biologiques, à partir desquelles nous allons extraire nos données.

L'analyse et l'interprétation de ces données nécessitent des algorithmes spécialisés. C'est ce que nous détaillerons dans le chapitre suivant.

# 2

## Chapitre

### Algorithmique du texte

L'algorithmique du texte est une science informatique permettant de traiter un texte de façon « *exacte* » ou « *approchée* » en utilisant l'ordinateur. Ces algorithmes sont utilisés dans plusieurs domaines tels que la bio-informatique, la recherche documentaire, l'indexation automatique pour les moteurs de recherche, ...etc.

La bio-informatique est issue de la rencontre de l'informatique et de la biologie : la récolte des données en biologie a connu une très forte augmentation ces 30 dernières années. Pour analyser cette grande quantité de données de manière efficace, les scientifiques ont de plus en plus recours au traitement automatique de l'information, c'est-à-dire à l'informatique. Parmi les problèmes issus de la bio-informatique nous nous intéressons dans ce travail à la recherche de motifs.

La recherche de motifs dans un texte (dans notre étude le texte est une séquence d'ADN) peut être exacte ou approchée, unique (un seul motif) ou multiple (plusieurs motifs) et aussi avec ou sans prétraitement du texte ou du motif.

Dans ce chapitre nous allons donc présenter quelques méthodes de recherche de motifs, issues de l'algorithmique du texte, les plus connues à travers un exemple suivi de l'algorithme correspondant. Voici d'abord quelques notations et définitions nécessaires.

#### 2.1. Notations et définitions

1. Un **alphabet**  $\Sigma$  est un ensemble fini non vide d'éléments appelés symboles ou lettres. La cardinalité de cet ensemble est notée  $|\Sigma| = \sigma$ .

Exemple :  $\Sigma_1 = \{a, b, \dots, z\}$ ,  $\Sigma_2 = \{0, 1\}$ ,  $\Sigma_3 = \{A, C, G, T\}$ .

2. Un **mot**  $M$  défini sur un alphabet  $\Sigma$  est une concaténation d'éléments de  $\Sigma$ ,  $M = e_1 e_2 \dots e_m$ .
3. La **longueur d'un mot** est le nombre de lettres qui le composent, et on la note  $|M| = m$ .

Exemple :  $\Sigma_3 = \{A, C, G, T\}$ ,  $M = ACCTTTGG$  est défini sur  $\Sigma_3$  et  $|M| = 8$ .

4. Le **mot vide** (sa longueur vaut zéro) est noté par  $\epsilon$
5. L'ensemble de tous les mots sur l'alphabet  $\Sigma$  est noté  $\Sigma^*$ .

Exemple:  $\Sigma = \{A, C, G, T\}$  et  $\Sigma^* = \{\varepsilon, A, C, G, T, AA, AC, \dots, ACG, \dots, ACCTTG\}$  sont des mots de longueur 1, 2, ..., 3, ..., 6.

6. Un **motif** sur un alphabet  $\Sigma$  est une suite de lettres de  $\Sigma$ .

Un motif peut être un simple mot, une expression régulière, un ensemble fini de mots.

7. Un **facteur** de  $M$  est une suite contiguë de lettres de  $M$ .

Un mot  $u$  est facteur de  $M$  s'il existe deux entiers  $i, j$  tels que :

$$0 \leq i \leq j \leq m - 1 \text{ et } M[i..j] = u. \quad M[i..j] = \varepsilon \text{ si } i > j.$$

8. Un **préfixe** de  $M$  est un facteur qui débute à la 1<sup>ère</sup> position (*indice 0*).

Exemple:  $M = AACGCC, \text{Préf}(M) = \{\varepsilon, A, AA, AAC, AACG, AACGC, AACGCC\}$

9. Un **suffixe** est un facteur qui finit à la dernière position (*indice  $m-1$* ).

Exemple:  $M = AACGCC, \text{Suff}(M) = \{\varepsilon, C, CC, GCC, CGCC, ACGCC, AACGCC\}$

10. Un **préfixe/suffixe est propre** s'il est de longueur strictement inférieure à  $|M| = m$ .

11. Un **ordre lexicographique** noté  $\leq$ , est un ordre sur les mots induit par un ordre sur les lettres.

Exemple:  $ACGAT < AGACT < AGCAT$ .

12. Une **occurrence** d'un mot  $M$  dans un texte  $T$ , lorsque  $M$  est un facteur de  $T$ . Toute occurrence peut être caractérisée par une position sur  $T$ .

Exemple: Le mot  $M = ACA$  apparaît 3 fois dans le texte  $T = ACACCACACA$  aux positions 0; 5; 7. Le nombre d'occurrence de  $M$  dans  $T$  est égal à 3.

13. Un **bord** d'un mot  $M$  est un mot  $u \neq M$ , qui est à la fois préfixe et suffixe de  $M$ .

Exemple : Les bords du mot  $AATAATAA$  sont  $\varepsilon, A, AA$  et  $AATAA$ .

14. Une **période** : Soit  $M$  un mot non vide et  $m = |M|$ , un entier  $p$  tel que  $0 < p \leq m$  est une période de  $M$  notée  $\text{Pér}(M)$ , si  $M[i] = M[i + p]$  pour  $0 \leq i \leq m - p - 1$ . La période de  $M$  non vide est la plus petite de ses périodes.

Exemple :  $M = AATAATAA, \text{Pér}(AATAA) = 3, \text{Pér}(AA) = 6, \text{Pér}(A) = 7$   
et  $\text{Pér}(\varepsilon) = 8$  sont des périodes de  $M$  et  $\text{Pér}(M) = 3$ .

(Crochemore et al., 2001)

## 2.2. Algorithmes de recherche exacte d'un motif

Les algorithmes de recherche exacte (pattern matching) d'un seul motif à la fois sont des méthodes qui permettent de répondre de manière exacte à un problème donné. Autrement dit, l'algorithme permet de localiser tous les caractères (contigus) d'un motif dans un texte.

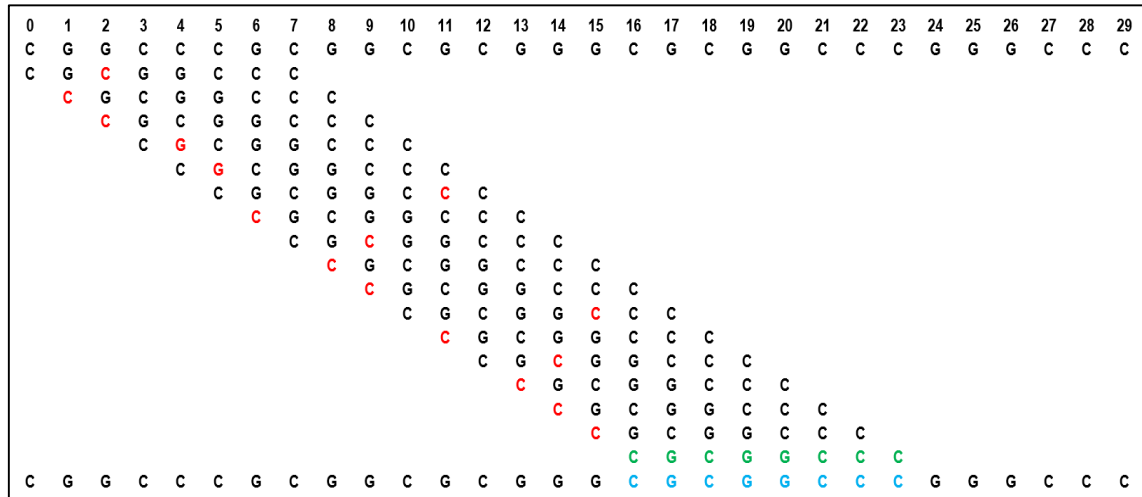


### 2.2.1. Algorithme naïf (fenêtre glissante)

Cet algorithme est loin d'être optimal en complexité temporelle. Par contre, il a une très bonne complexité spatiale et ne nécessite aucun pré-traitement.

L'algorithme utilise le principe de la fenêtre glissante. Il va chercher le motif en décalant d'un caractère en cas d'erreur. De plus, la comparaison motif-texte se fait de gauche à droite. Il renvoie le décalage représentant le début du motif dans le texte ou 0 si le motif n'apparaît pas dans le texte. (*Parreaux, 2018*)

Exemple : On veut trouver le motif « CGCGCCCC » dans la séquence donnée :



**Figure 2.1:** Exemple de l'utilisation de l'algorithme naïf.

---

**Algorithme 2.1 :** Recherche Naïve ( $T, n, S, m$ ) // avec  $T$  : texte,  $S$  : motif,  $n=|T|$  et  $m=|S|$

```

    Pour j de 0 à n-m faire
        i ← 0 ;
        Tant que (T [j + i] = S [i]) et (i < m) faire
            i ← i + 1 ;
        Fin Tant que ;
        Si (i == m) alors Signaler une occurrence de S ;
    Fin Pour ;

```

La complexité de l'algorithme naïf peut demander un très grand nombre de comparaisons selon les situations, ce qui peut entraîner un très long temps de "calcul" avec des chaînes très longues.

Sa complexité est de l'ordre de :  $O(m * n)$  avec  $m = |S|$  et  $n = |T|$

### 2.2.2. Algorithme de Knuth-Morris-Pratt (KMP)

C'est un algorithme de recherche de sous-chaîne (de caractères), permettant de trouver les occurrences d'une chaîne  $P$  dans un texte  $S$  en effectuant un pré-traitement de la chaîne, qui fournit une information suffisante pour déterminer où continuer la recherche en cas de non-correspondance. Cela permet à l'algorithme de ne pas réexaminer les caractères qui ont été précédemment vérifiés, et donc de limiter le nombre de comparaisons nécessaires.

Le principe de l'algorithme est le suivant :

**Phase 1 :** Construire un tableau, indiquant pour chaque position un « *décalage* », c'est-à-dire la prochaine position où peut se trouver une occurrence potentielle de la chaîne.

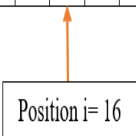
**Phase 2 :** Effectuer la recherche à proprement parler, en comparant les caractères de la chaîne et ceux du texte. En cas de différence, il utilise le tableau pour connaître le décalage à prendre en compte pour continuer la recherche sans retour en arrière. (Iovleff, 2018)

**Exemple :** On applique l'algorithme KMP pour trouver les occurrences du motif « CGCGGCCC » dans la séquence d'ADN suivante : « CGGCCCCGCGGCGCGGGCGCGGCCCGGGCCC ».

$j$	0	1	2	3	4	5	6	7	
Motif	C	G	C	G	G	C	C	C	
Table des Bords	-1	0	-1	0	2	-1	1	1	1

$i$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
Texte $T$	C	G	G	C	C	C	G	C	G	G	C	G	C	G	G	G	C	G	C	G	G	C	C	C	G	G	G	C	C	C
$j$	0	1	2	0	0	0	1	0	1	2	0	1	2	3	4	3	0	1	2	3	4	5	6	7	0	1	2	0	2	3
$T_i=S_j$	y	N	N	y	N	N	N	y	y	N	y	y	y	y	y	N	y	y	y	y	y	y	y	Y	N					



**Figure 2.2:** Exemple de l'algorithme KMP.

**Algorithme 2.2 :** Recherche KMP ( $T, n, S, m$ ) : // avec  $T$  : texte,  $S$  : motif,  $n=|T|$  et  $m=|S|$

Debut

```

i ← 0 ; j ← 0 ;
Tant que (i < n-m+1) faire
    Tant que ((j < m) et (T[i+j] = S[j])) faire
        j ← j+1 ;
    Fin Tant que ;
    Si (j=m) alors retourner (i) ;
    i ← i+j- Bord[j] ;
    Si (Bord[j] > 0) alors j ← Bord[j]
    Sinon j ← 0 ;
    Fin Si ;
Fin Tant que ;
retourner -1 ;

```

Fin ;

Sachant que Bord[j] est une table des bords calculés par l'algorithme suivant :

---

TableauDesBords (S, m) :

---

Bord[0]  $\leftarrow$  -1; i  $\leftarrow$  -1 ;

Pour j  $\leftarrow$  0 à m-1 faire

  Tantque (i  $\geq$  0 et S[i]  $\neq$  S[j]) faire i  $\leftarrow$  Bord[i] Fin Tant que ;

  i  $\leftarrow$  i+1 ;

  si (i=m-1) ou (S[j+1]  $\neq$  S[i]) alors Bord[j+1]  $\leftarrow$  i

  sinon Bord[j+1]  $\leftarrow$  Bord[i] ;

  Fin si ;

Fin Pour ;

---

La complexité de KMP est de l'ordre de :  $O(m + n)$  avec  $m = |S|$  et  $n = |T|$ .

Il existe d'autres algorithmes tels que Boyer Moore (*Sèverine, 2016*), Horspool (*Saldaña, 2018*), ...etc. Chacun a ses spécificités par exemple Boyer Moore est plus adapté pour un grand alphabet et la comparaison se fait de droite à gauche.

### 2.3. Algorithmes de recherche exacte multiple de motifs

Les algorithmes de cette section effectuent un prétraitement soit du texte  $T$  et/ou de l'ensemble des motifs  $P_i$  afin de rechercher toutes les occurrences des motifs  $P_i$  en ne parcourant qu'une seule fois le texte  $T$ .

#### 2.3.1. Algorithme de Rabin-Karp

L'algorithme de Rabin-Karp créé en 1987 pour la recherche d'un ensemble de motifs donnés dans un texte grâce à une fonction de hachage. L'algorithme n'est pas beaucoup employé pour les recherches d'un unique motif mais il a une importance théorique et s'avère très efficace pour la recherche de multiples sous-chaînes. Il s'appuie sur la théorie élémentaire des nombres. En effet, tout bloc de la taille du motif dans le texte ainsi que le motif vont être soumis à un calcul donnant une valeur modulo un nombre premier. A la place de tester si tous les blocs sont égaux au motif, on testera si tous les blocs qui ont le même nombre modulo ce nombre premier sont égaux. On espère avoir moins de tests à effectuer. L'algorithme renvoie le décalage représentant le début du motif dans le texte ou 0 si le motif n'apparaît pas dans le texte. (*Parreaux, 2018*)

Exemple : L'alphabet nucléotidique est  $\Sigma = \{A, C, G, T\}$ . Nous pouvons mapper des caractères individuels de cet alphabet aux chiffres du système de base 4 comme suit :

Caractère	Codage
A	0
C	1
G	2
T	3

**Tableau 2.1:** Codage de l'alphabet nucléotidique.

Considérons une chaîne  $P = p_1 \dots p_m$  dans l'alphabet  $\Sigma$ . Sans perte de généralité, nous utilisons  $p_i$  pour désigner à la fois le caractère en  $\Sigma$  et le chiffre de l'alphabet de base  $K$  qui lui est associé. La valeur numérique  $p$  représentée par  $P$  peut être calculée à l'aide de l'expression suivante :

$$p = p_m + K(p_m - 1 + K(p_m - 2 + \dots + K(p_2 + Kp_1)) \dots)$$

Étant donné une chaîne  $S = s_1 \dots s_n$ , où  $n > m$ , la valeur numérique de  $S[1, m]$ ,  $t_0$  peut être calculée de la même manière:

$$t_0 = s_m + K(s_m - 1 + K(s_m - 2 + \dots + K(s_2 + Ks_1)) \dots)$$

Plus important encore, étant donné  $t_{i-1}$ , la valeur numérique  $t_i$  représentant la sous-chaîne  $S[i+1, i+m]$  peut être calculée de manière incrémentielle comme suit :

$$t_i = K(t_{i-1} - K^{m-1}s_i) + s_{i+m}$$

Nous pouvons pré-calculer  $K^{m-1}$  car ce nombre est une constante.

Prenons la séquence d'ADN suivante :  $S = \text{"ATTCCGT"}$ . Supposons que nous recherchons des sous-chaînes dont la taille est égale à 4.

Il y a  $|S| - 4 + 1 = 7 - 4 + 1 = 4$  sous-chaînes de 4 lettres en  $S$  :  $S[1;4]$ ,  $S[2;5]$ ,  $S[3;6]$  et  $S[4;7]$ .

Nous calculons  $t_0$  pour  $S[1;4]$  en utilisant le codage des symboles nucléotidiques aux chiffres dans le système numérique de base 4 comme suit :

$$S[1;4] = \text{"ATTC"}; t_0 = 1 + 4(3 + 4(3 + 4 \cdot 0)) = 1 + 4(3 + 4 \cdot 3) = 1 + 4 \cdot 15 = 61.$$

$$S[2;5] = \text{"TTCC"}; t_1 = 4(t_0 - 4^3 \cdot 0) + 1 = 4(61 - 0) + 1 = 244 + 1 = 245.$$

$$S[3;6] = \text{"TCCG"}; t_2 = 4(t_1 - 4^3 \cdot 3) + 2 = 4(245 - 64 \cdot 3) + 2 = 4(245 - 192) + 2 = 4 \cdot 53 + 2 = 214.$$

$$S[4;7] = \text{"CCGT"}; t_3 = 4(t_2 - 4^3 \cdot 3) + 3 = 4(214 - 192) + 3 = 4 \cdot 22 + 3 = 91.$$

Son pseudocode est le suivant :

---

**Algorithme 2.3 : Recherche-Rabin-Karp (T, S, p, q) :**


---

Debut

```

n ← longueur [T] ; m ← longueur [S] ;
h ←  $d^{m-1} \bmod q$  ;
k ← 0 ; t0 ← 0 ;
Pour i ← 0 à m faire
    k ← (d.k + S[i]) mod q ; t0 ← (d.t0 + T[i]) mod q ;
Fin Pour ;
Pour s de 0 à (n-m) faire
    Si ((p = ts) et (P[1..m] = T[s+1 .. s+m] )) alors
        Ecrire ("Occurrence trouvée à la position", s) ;
        Si (s < n-m) alors ts+1 = (d(ts - hT[s+1]) + T[s+m+1]) mod q Fin si ;
    Fin Si ;
Fin Pour ;
Retourner 0 ;

```

Fin ;

---

La complexité de cet algorithme est d'ordre  $O(m * n)$  avec  $m = |S|$  et  $n = |T|$

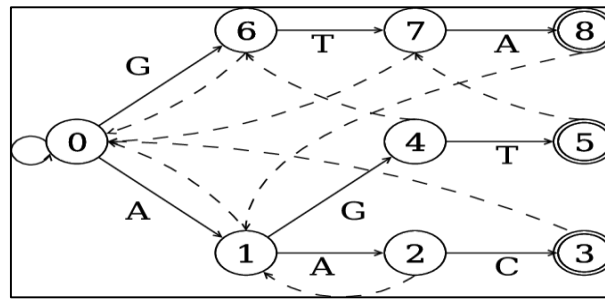
### 2.3.2. Algorithme d'Aho-Corasick

Etant donné l'ensemble de motifs  $M = \{M_1, \dots, M_k\}$  de tailles respectives  $m_1, \dots, m_k$  et le texte  $T$  de taille  $n$ , l'algorithme d'Aho-Corasick permet de trouver toutes les occurrences de tous les motifs  $M_1 \dots M_k$  en un seul passage dans le texte  $T$ . Il peut être vu comme une généralisation de l'algorithme de Knuth-Morris-Pratt pour une recherche multiple. (*Aho & Corasick, 1975*)

L'algorithme est constitué de deux parties :

1. Construire un automate à états finis déterministe pour tous les mots de l'ensemble  $M$ . Pour cela, on construit un « tri ou arbre de préfixes »  $K$ , satisfaisant les contraintes suivantes :
  - Chaque arête est étiquetée par un et un seul caractère.
  - Deux arêtes sortant d'un même sommet ont des étiquettes différentes.
  - Chaque mot  $m_i$  de  $M$  est associé à un sommet  $v$  de  $K$ , i.e. les caractères étiquetant le chemin de la racine de  $K$  à  $v$  forment le mot  $m_i$ . De plus, chaque feuille de  $K$  est associée à un mot de  $M$ . (*El-Mabrouk, 2013*).
2. Parcourir le texte  $T$  qui sera comme entrée pour l'automate afin de trouver les occurrences de tous les mots correspondants.

Exemple : L'automate d'Aho-Corasick correspondant à la recherche des motifs "AAC", "AGT" et "GTA" est le suivant :



**Figure 2.3:** Exemple d'un automate d'Aho-Corasick.

Le pseudocode de l'algorithme est le suivant :

---

**Algorithme 2.4 :** Aho-Corasick ( $M, k, T, n$ ) :

---

Debut

$e \leftarrow \text{Pre-AC}(M, k)$  ;

Pour  $j \leftarrow 0$  à  $(n-1)$  faire

Tant que  $\delta(e, T[j])$  est non définie faire

$e \leftarrow \text{sup}(e)$  ;

Fin Tant que ;

$e \leftarrow \delta(e, T[j])$  ;

Si sortie( $e$ ) alors reporter une occurrence des éléments de sortie( $e$ ) en position  $j$  ;

Fin Pour ;

Fin ;

---

Sachant que :

- $M$  et  $T$  sont le motif et le texte de longueurs respectives  $k$  et  $n$ .
- $\text{sup}$  est le suppléant tel que :  $\text{sup}(q) = u$  où  $u$  est le plus long suffixe propre de  $q$  qui appartient à  $\text{Préf}(X)$ .
- $\text{Pre-AC}$  est une fonction qui réalise la phase de prétraitement.

Cet Algorithme à une complexité de l'ordre de  $O(n + |M| + z)$ , où  $|M|$  est la somme des longueurs des mots de l'ensemble  $M$ , et  $z$  est le nombre total d'occurrence dans  $T$  des motifs de  $M$ .

## 2.4. Algorithmes de recherche approchée de motifs

La recherche d'un motif dans une séquence biologique nécessite souvent de tolérer des différences. D'un point de vue algorithmique, cela revient au problème de la recherche approchée dans un texte. C'est un contexte de recherche où l'on permet l'existence d'un certain nombre d'erreurs entre le motif et ses occurrences dans le texte. (Crochemore et al., 2001 ; Chegrane 2016)

Nous présentons dans cette section des méthodes basées sur le calcul de distance ou de score entre deux motifs afin de déterminer les ressemblances.

### 2.4.1. Distance de Hamming

La distance de Hamming est définie pour deux mots  $U$  et  $V$  de même longueur comme le nombre de positions en lesquelles les deux mots possèdent des lettres différentes :

$$Ham(U, V) = \text{card} \{i / U[i] \neq V[i] \text{ et } 0 \leq i \leq |U| - 1\}$$

En d'autres termes, c'est le nombre de substitutions de caractères nécessaires pour transformer le mot  $U$  en  $V$ .

Exemple : Soient les mots  $U$  et  $V$  tels que :  $U = \text{TG}\text{CAT}\text{TGCA}$ ,  $V = \text{TGGAT}\text{GGA}$ .

Alors :  $Ham(U, V) = 3$ .

Remarque : La distance de Hamming fournit un moyen simple mais pas toujours pertinent pour comparer deux mots puisqu'elle n'autorise qu'une seule opération (la substitution). (*Crochemore et al., 2001*)

### 2.4.2. Distance de Levenshtein

Egalement appelée « *distance d'édition* », la distance de Levenshtein (*Levenshtein, 1966*) entre deux mots  $U$  et  $V$  notée  $Lev(U, V)$  est le nombre minimal de modifications (insertions, délétions ou substitutions) nécessaires pour transformer  $U$  en  $V$ . (*Gueddah et al., 2014*)

Le calcul de  $Lev(U, V)$  se fait grâce à une matrice  $L$  de taille  $(|U| + 1) * (|V| + 1)$ . Chaque case  $L[i, j]$ ,  $0 \leq i \leq |U|$  et  $0 \leq j \leq |V|$  est remplie de la manière suivante :

$$\boxed{\begin{array}{l} L[0, j] = j. \quad L[i, 0] = i. \\ L[i, j] = \min \left\{ \begin{array}{l} L[i-1, j-1] + \begin{cases} 0 & \text{si } U[i-1] = V[j-1] \\ 1 & \text{si } U[i-1] \neq V[j-1] \end{cases} \quad \begin{array}{l} \text{substitution} \\ \text{suppression} \\ \text{insertion} \end{array} \\ L[i-1, j] + 1 \\ L[i, j-1] + 1 \end{array} \right. \end{array}}$$

La distance entre les deux mots se trouve dans le coin inférieur droit de la matrice  $L$ .

Ce calcul se fait en temps  $O(|U| \times |V|)$ . (*Vroland, 2016*)

Exemple : Soit  $U = \text{CTACGATAG}$  et  $V = \text{CTAGTTAG}$  deux séquences, la matrice de Levenshtein est la suivante :

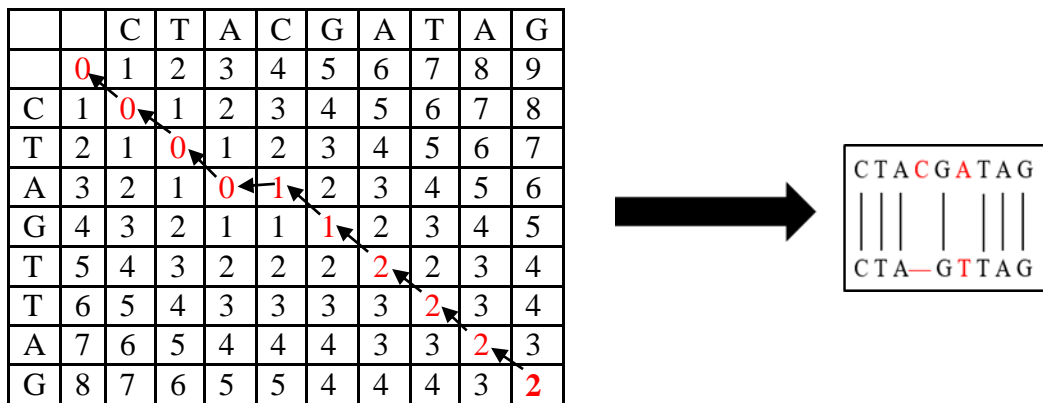


Figure 2.4: Exemple de la distance de Levenshtein.

Donc

$$\text{Lev}(U, V) = 2 \text{ (1 insertion et 1 substitution)}$$

---

**Algorithme 2.5 :** Dist-Levenshtein (P, S) :
 

---

Debut

```

n ← longueur(P) ; m ← longueur(S) ;
Pour i de 0 à n faire T[ i, 0] ← i ; Fin Pour ;
Pour j de 0 à m faire T[0, j] ← j ; Fin Pour ;
pour i de 1 à n faire
    pour j de 1 à m faire
        Si (P[ i ] == S[ j ]) alors c ← 0 sinon c ← 1  Fin si ;
        T[ i, j ] ← min( T[i-1, j] + 1, T[i, j-1] + 1, T[i-1, j-1] + c ) ;
                        // délétion      insertion      substitution
    Fin Pour ;
Fin Pour ;
retourner T[n, m] ;

```

Fin ;

---

La complexité de l'algorithme est d'ordre  $O(n * m)$ .

À partir de la distance de Levenshtein, on peut aligner<sup>4</sup> les deux séquences en remontant le chemin dans la matrice (retour arrière) qui a permis d'obtenir cette distance opération d'édition par opération d'édition.

Exemple d'alignement :

U :	<span style="color: red;">[</span> A <span style="color: red;">]</span>	C	G	<span style="color: blue;">[</span> C <span style="color: blue;">]</span>	T	C	<span style="color: green;">[</span> A <span style="color: green;">]</span>	A	T	<span style="color: magenta;">[</span> - <span style="color: magenta;">]</span>
V :	<span style="color: red;">[</span> A <span style="color: red;">]</span>	C	G	<span style="color: blue;">[</span> T <span style="color: blue;">]</span>	T	C	<span style="color: green;">[</span> - <span style="color: green;">]</span>	A	T	<span style="color: magenta;">[</span> C <span style="color: magenta;">]</span>

Identité  
(match)

Substitution  
(mismatch)

Délétion  
(gap)

Insertion  
(gap)

Afin de quantifier la similitude entre les séquences et trouver l'alignement optimal, un score d'alignement est calculé. Celui-ci peut mesurer soit le rapprochement, soit l'éloignement des séquences.

---

<sup>4</sup> L'alignement de séquences est une méthode utilisée pour la recherche de similarités. Cette méthode consiste à superposer deux séquences (ou plus) pour identifier les régions de concordance.



Le score de l'alignement doit prendre en compte toutes les positions alignées : identités (match), substitutions (mismatch) et indels (gaps). Chacun de ces événements va recevoir un poids, appelé score élémentaire  $Se$ . Le score de l'alignement correspondra à la somme des scores élémentaires correspondant aux positions alignées.

### 2.4.3. Algorithme de Needleman-Wunsch

L'algorithme de Needleman-Wunsch (NW) est un algorithme qui effectue un alignement global<sup>5</sup> maximal de deux chaînes de caractères. Il est couramment utilisé en bio-informatique pour aligner des séquences de protéines ou de nucléotides. (*Brochier-Armanet, 2018*)

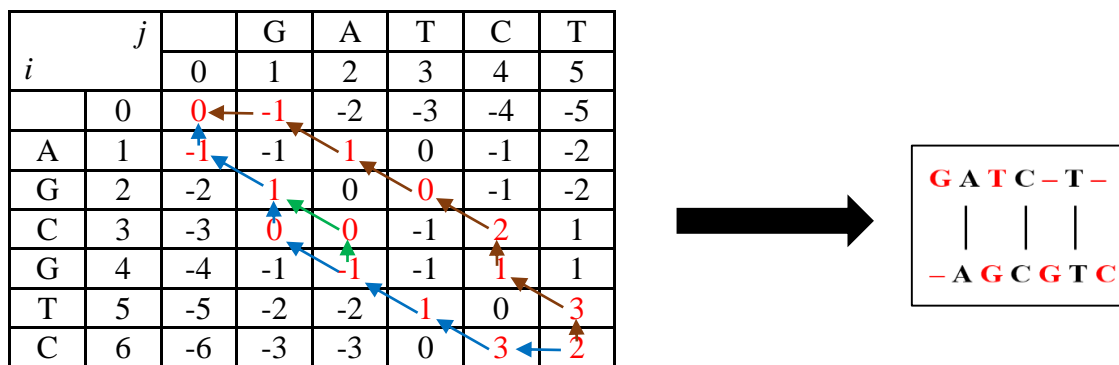
Soit  $U$  et  $V$  deux séquences de tailles respectives  $n$  et  $m$ . L'alignement de  $U$  et  $V$  s'obtient comme suit :

- Choix du système des scores : déterminer les scores des « *match/mismatch* » et « *la pénalité de gap identique* » à chaque **insertion** ou **suppression** (indel).
- Construction de la matrice des scores (ou la matrice de comparaison)  $S$  où chaque  $S[i, j]$  stocke le score optimum de la trajectoire permettant d'arriver à cette case.

Exemple : Pour deux chaînes  $U = \text{GATCT}$  et  $V = \text{AGCGTC}$ , nous obtenons l'alignement suivant :

Avec : *Identité (match)* = 2, *Substitution (mismatch)* = -1, *indel (gap)* = -1

La matrice de Needleman-Wunsch associée est la suivante :



**Figure 2.5:** Exemple d'un alignement global maximal.

<sup>5</sup> Les séquences sont alignées sur l'ensemble de leur longueur.

**Algorithme 2.6 :** Needleman-Wunsch (T, M, S, D) :

Debut

 $n \leftarrow \text{longueur}(T)$  ; $m \leftarrow \text{longueur}(M)$  ; $S(0,0) = 0$  ;Pour  $j$  de 1 à  $j \leq m$  faire  $S(0, j) = -j * D$  ; //  $D$  : pénalité d'insertion ou de suppressionPour  $i$  de 1 à  $i \leq n$  faire  $S(i, 0) = -i * D$  ;Pour  $i$  de 1 à  $i \leq n$  faire    Pour  $j$  de 1 à  $j \leq m$  faire         $a(i-1, j-1) = S(i-1, j-1) + S(u_i, v_j)$  ; //  $S(u_i, v_j)$  score    *match/mismatch*         $a(i-1, j) = S(i-1, j) + D$  ;         $a(i, j-1) = S(i, j-1) + D$  ;         $S(i, j) = \max\{ a(i-1, j-1), a(i-1, j), a(i, j-1) \}$  ;

Fin ;

Fin ;

retourner  $S$  ;

Fin ;

La complexité de l'algorithme est d'ordre  $O(n * m)$ .**2.4.4. Algorithme de Smith –Waterman**

L'algorithme de Smith-Waterman est un algorithme utilisé en bio-informatique pour effectuer un alignement local<sup>6</sup>. Il donne un alignement correspondant au meilleur score possible de correspondance entre les acides aminés ou les nucléotides des deux séquences.

La différence essentielle avec l'algorithme de Needleman et Wunsch est que n'importe quelle case de la matrice de comparaison peut être considérée comme point de départ pour le calcul des scores et que tout score qui devient inférieur à zéro stoppe la progression du calcul des scores. (Smith & Waterman, 1981)

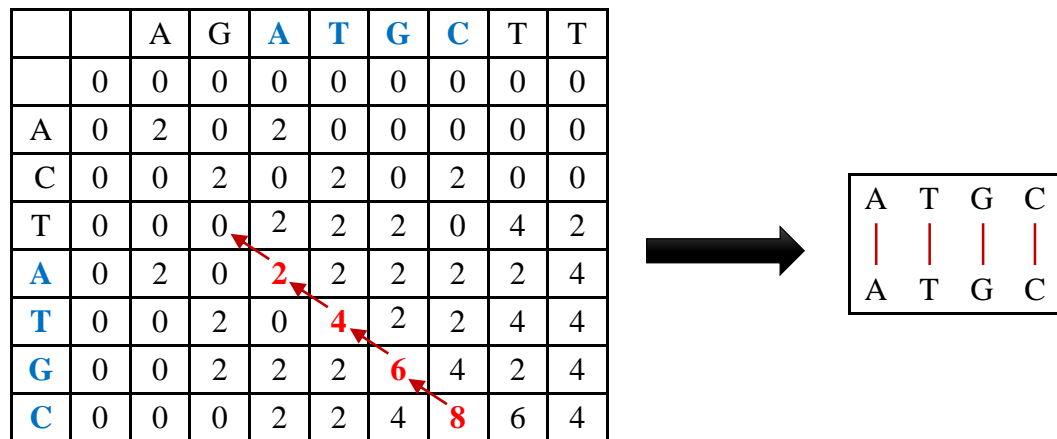
Quand on reconstruit l'alignement par la procédure de « *retour en arrière* », au lieu de partir de la dernière case, on choisira celle qui a le score le plus élevé. La procédure s'arrête lors de la rencontre d'une case contenant un zéro. Cette case indique le début d'un alignement.

Exemple : Soit les deux séquences U et V tel que : U = AGATGCTT et V = ACTATGC.

Avec : *Identité (match)* = 2, *Substitution (mismatch)* = 0, *indel (gap)* = -2

La matrice de Smith-Waterman associée est la suivante :

<sup>6</sup> Chercher et localiser les régions les plus similaires entre les deux séquences et donc, l'alignement se fait entre une partie de chacune des séquences et non pas la totalité.



**Figure 2.6:** Exemple d'un alignement local.

La valeur maximale de la matrice = 8 et en remontant dans la diagonale jusqu'à la case contenant un zéro on obtient l'alignement optimal local ci-dessus.

---

**Algorithme 2.7 :** Smith-Waterman (U, V, S, D):

---

Début

```

n ← longueur(U) ;
m ← longueur(V) ;
S(0,0)= 0 ;
Pour j de 1 à m faire S(0, j) = 0 ;
Pour i de 1 à n faire S(i, 0) = 0 ;
Pour i de 1 à n faire
    Pour j de 1 à m faire
         $a(i-1, j-1) = S(i-1, j-1) + S(u_i, v_j)$  ; //  $S(u_i, v_j)$  score match/mismatch
         $a(i-1, j) = S(i-1, j) + D$  ; //  $D$  : pénalité d'insertion ou de suppression
         $a(i, j-1) = S(i, j-1) + D$  ;
         $S(i, j) = \max\{ 0, a(i-1, j-1), a(i-1, j), a(i, j-1) \}$  ;
    Fin ;
Fin ;
retourner S ;
```

Fin ;

---

La complexité de l'algorithme est d'ordre  $O(n * m)$ .

**Remarque :** Il est important de préciser que les algorithmes de Needleman-Wunsch et Smith-Waterman retournent un score de similitude avec un alignement des deux motifs contrairement à la méthode de Levenshtein qui retourne la distance de dissimilarité.

Nous présentons dans ce qui suit la recherche de motifs avec prétraitement du texte où il s'agit d'indexer le texte afin d'optimiser le temps de recherche.

## 2.5. Structure d'index

Une structure d'index est une structure de données qui permet de mémoriser tous les suffixes d'un mot. Elle est conçue pour donner un accès direct aux facteurs du mot.

Il existe plusieurs structures, voici les plus fréquemment utilisées pour l'exploration de textes.

### 2.5.1. Les tables de suffixes

La table des suffixes est un tableau d'entiers représentant la position de début de chacun des suffixes d'une séquence  $S$  classés par ordre lexicographique. (Manber & Myers, 1990)

Elle peut être associée à d'autres tables annexes comme celle des **LCP** (Longest Common Prefix) qui indique la longueur du préfixe commun entre le suffixe débutant en position  $S_i$  et  $S_{i-1}$ . Les **LCP** sont particulièrement utiles dans le cas de la recherche de facteurs répétés.

Après la construction de la table de suffixes **TS**, il est possible de rechercher les occurrences d'un motif dans la table **TS** par la méthode dichotomique. (Coissac, 2005)

Exemple : On veut réaliser la table de suffixes pour la chaîne suivante :

S=	A	C	G	G	T	C	C	C	G	G	T	A
	1	2	3	4	5	6	7	8	9	10	11	12

LCP	TS	Suffixes
0	12	A
1	1	A C G G T C C C G G T A
0	6	C C C G G T A
2	7	C C G G T A
1	8	C G G T A
4	2	C G G T C C C G G T A
0	9	G G T A
3	3	G G T C C C G G T A
1	10	G T A
2	4	G T C C C G G T A
0	11	T A
1	5	T C C C G G T A

**Tableau 2.2:** Exemple d'une table de suffixes.

La complexité de la construction de la table **TS** est de l'ordre de  $O(n^2 \log n)$ .

Le tri effectue  $O(n \log n)$  comparaisons et chaque comparaison a un coût en  $O(n)$ .

La complexité de la recherche des occurrences du motif est de l'ordre de  $O(m \log n)$  avec  $m = |\text{motif}|$  et  $n = |\text{texte}|$ .

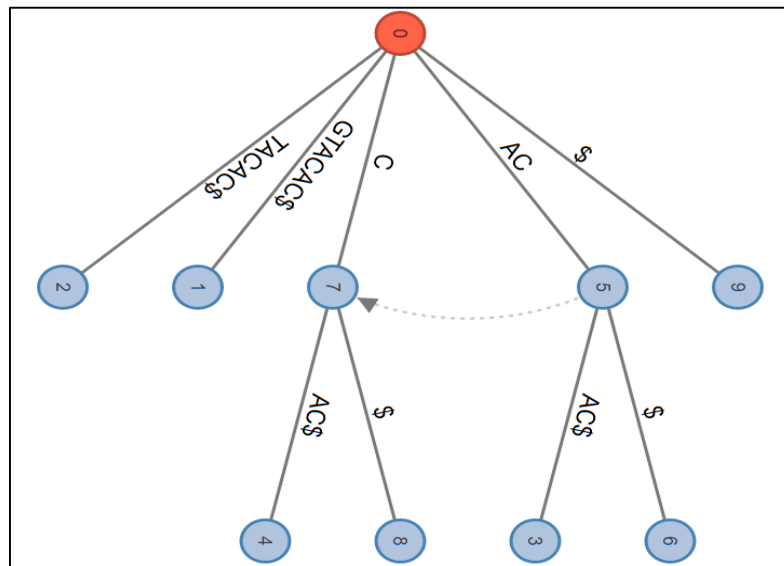
### 2.5.2. Les arbres de suffixes

C'est une structure de données qui désigne les caractéristiques internes des séquences, utilisée pour la recherche exacte d'un mot dans un texte, la recherche de répétitions super-maximales ainsi que pour la recherche de répétitions et la recherche de palindromes ...etc.

Un arbre des suffixes pour un texte  $T$ , dénoté  $\mathcal{T}$ , est un arbre enraciné et orienté, tel que :

- Chaque nœud interne a au moins 2 fils.
- $\mathcal{T}$  à  $n$  feuilles numérotées de 1 à  $n$ .
- Chaque arc est étiqueté par un facteur (sous-mot) de  $T$ .
- Deux arcs sortant d'un même nœud ont des étiquettes débutant par des caractères différents.
- Le chemin de la racine à une feuille  $i$  est étiqueté par le suffixe  $T[i..n]$ . (Hamel, 2012)

Exemple : Étant donné la chaîne « GTACAC », voici l'arbre de suffixes correspondant :



**Figure 2.7:** Exemple d'un arbre de suffixes.

On peut construire un arbre de suffixes en temps  $O(n)$  où  $n$  est la longueur de la chaîne de caractères. La recherche d'un motif exacte est de l'ordre de  $O(|motif|)$ .

L'inconvénient de l'arbre des suffixes est l'espace mémoire qui est de l'ordre de  $O(n \log n)$ . Généralement la table de suffixes est préférée à l'arbre de suffixes pour sa simplicité.

## 2.5.3. FM-index

Le FM-index est la première structure d'indexation à atteindre une taille proche de l'entropie du texte indexé. Cette structure a été créée en 2000 par Paolo Ferragina et Giovanni Manzini (*Ferragina & Manzini, 2000*), elle est une compression sans perte basée sur la Transformée de Burrows-Wheeler<sup>7</sup> (*Burrows & Wheeler, 1994*).

En plus de la compression, le FM-index peut être utilisé pour trouver de façon efficace le nombre d'occurrences d'un motif dans le texte compressé, ainsi que pour localiser la position de chaque occurrence du motif dans le texte compressé. (*Tatiana, 2018*)

Exemple : Soit la chaîne = **AGCTGC\$** (\$ désigne la fin de la chaîne)

1. Construire les conjugués de la chaîne dans une matrice carrée (rotations cycliques).
2. Classer les lignes par ordre alphabétique en précisant la colonne **F (First)** et la colonne **L (Last)** (*i* représente la position du caractère de **F** dans la chaîne originale)

<i>i</i>	F						L
0	A	G	C	T	G	C	\$
1	G	C	T	G	C	\$	A
2	C	T	G	C	\$	A	G
3	T	G	C	\$	A	G	C
4	G	C	\$	A	G	C	T
5	C	\$	A	G	C	T	G
6	\$	A	G	C	T	G	C

Tri

	<i>i</i>	F						L
0	6	\$	A	G	C	T	G	C
1	0	A	G	C	T	G	C	\$
2	5	C	\$	A	G	C	T	G
3	2	C	T	G	C	\$	A	G
4	1	G	C	\$	A	G	C	T
5	4	G	C	T	G	C	\$	A
6	3	T	G	C	\$	A	G	C

La colonne **L** représente la BWT de la chaîne : **BWT(AGCTGC\$) = C\$GGTAC**

Le FM-Index permet le calcul du nombre d'occurrences d'un motif et ses positions dans le texte compressé. Pour cela plusieurs tables sont nécessaires :

- a. Table C[c] :** C[c] est une table calculant pour chaque caractère de l'alphabet le nombre d'occurrences des caractères lexicalement plus petits contenus dans le texte.

Prenons l'exemple précédent, la table **C[c]** pour la chaîne **C\$GGTAC** est la suivante :

c	\$	A	C	G	T
C[c]	0	1	2	4	6

- b. Table OCC[c, k] :** La table **OCC[c, k]** est le nombre d'occurrences du caractère *c* dans le préfixe "**L[0..k]**". Table **OCC[c, k]** pour la chaîne **C\$GGTAC** est la suivante :

<sup>7</sup> La transformée de Burrows-Wheeler (BWT) est un algorithme qui maximise les répétitions de lettres dans un texte, ce qui est très utile en compression de données.

L	C	\$	G	G	T	A	C
k \ c	0	1	2	3	4	5	6
\$	0	1	1	1	1	1	1
A	0	0	0	0	0	1	1
C	1	1	1	1	1	1	2
G	0	0	1	2	2	2	2
T	0	0	0	0	1	1	1

### c. Operations :

- **LF mapping** : Une fonction qui permet la correspondance entre la première et la dernière colonne de la matrice résultant de la BWT. Cela est réalisé à l'aide de la table  $C[c]$  et  $OCC[c, k]$ . Donc pour chaque indice  $i$  dans  $L$ , il existe un indice  $j$  dans  $F$  tel que  $L[i]=F[j]$ . L'opération est définie par la formule suivante :

$$LF(i) = C[L[i]] + OCC[L[i], i]$$

Pour chaque ligne de la matrice résultante de la BWT, le caractère situé dans la dernière colonne  $L[i]$  précède celui de la première colonne  $F[i]$  et également de la chaîne originale.

- **Count** : L'opération *count* prend en entrée un motif  $P$  et retourne le nombre d'occurrences de ce motif dans le texte original.

Prenons le motif «  $GC$  » et recherchons ses occurrences dans la chaîne «  $AGCTGC\$$  ». Le calcul se fait en suivant les étapes suivantes :

- Le premier caractère à rechercher est le «  $C$  », le dernier caractère du motif. L'ensemble de lignes initiales est défini sur  $[C[C]+1 .. C[C+1]] = [1+1 .. 3] = [2.. 3]$ . Cet ensemble de lignes représente sur la chaîne, tous les suffixes commençant par «  $C$  ».

2	5	C	\$	A	G	C	T	G
3	2	C	T	G	C	\$	A	G

- Le «  $G$  » est le dernier caractère à analyser. Le nouvel ensemble est définie en appliquant la formule utilisée précédemment :

$$[[C[G]+ OCC (G, début-1)+ 1 .. C [G]+ OCC(G, fin)]] = [3+ 0+ 1 .. 3+ 2] = [4 .. 5].$$

Ce dernier ensemble de lignes représente sur la chaîne, tous les suffixes commençant par «  $GC$  ».

Comme le traitement du motif est achevé, le compte est égal à la taille de la plage.

Dans l'exemple précédent, la fonction *count* retourne le résultat suivant :  $5 - 4 + 1 = 2$ .

Si la plage est vide ou si les limites de l'ensemble de lignes se croisent mutuellement avant que l'ensemble du motif soit trouvé, cela signifie que le motif n'existe pas dans la chaîne.

- **Locate** : L'opération *locate* a en entrée un index d'un caractère dans la chaîne et retourne sa position  $i$  dans la chaîne. Afin de mapper un index dans  $L$  en un index dans la chaîne, un sous-ensemble des indices en  $L$  est associé à une position dans la chaîne.

Ainsi la recherche du motif « GC » dans la chaîne « AGCTGC\$ » donne deux occurrences aux positions 1 et 4.

4	1	G	C	\$	A	G	C	T
5	4	G	C	T	G	C	\$	A

#### 2.5.4. Les filtres de Bloom

Il s'agit d'une structure de données probabiliste qui sert à tester si un élément fait partie d'un ensemble de données ou non. Probabiliste car cette structure peut générer des faux positifs et ne peut donc pas s'adapter à n'importe quelle application ou type de données.

On évitera par exemple d'utiliser une structure de données pouvant donner des faux positifs pour un robot d'opérations à cœur ouvert ou de système de guidage de missiles. En revanche, dès lors qu'on travaille sur de grands ensembles de données et qu'on peut accepter, disons, 0,1% de faux positifs, comme dans le comptage de  $k$ -mers ou la correction d'erreurs dans de très gros jeux de données de séquençage, le filtre de Bloom est très intéressant car rapide en temps d'exécution et compact en mémoire. (Nicolas, 2012)

Exemple : Une visualisation de l'interrogation d'un filtre de Bloom de  $k$ -mers d'une séquence d'ADN. La première étape consiste à stocker les  $k$ -mers de la séquence « ACCGTAG » dans un filtre Bloom. L'interrogation est effectuée de la même manière lorsque la séquence de requête « CGTAT » est décomposée en ses  $k$ -mers correspondants, et les  $k$ -mers sont utilisés pour interroger le filtre Bloom avec  $k = 4$ .

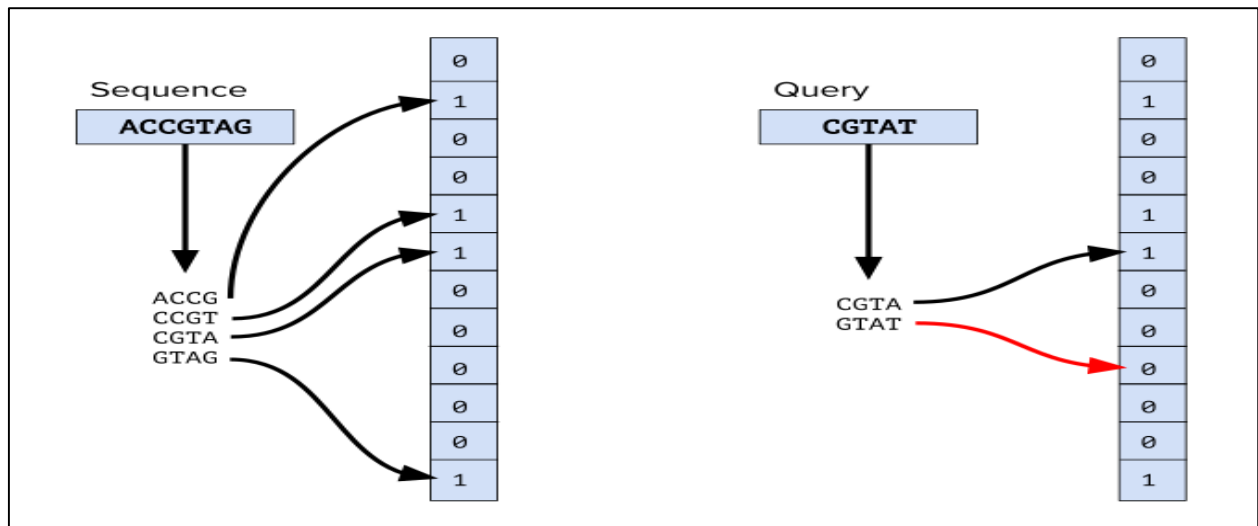


Figure 2.8: Exemple de filtre de Bloom.



## **2.6. Conclusion**

Nous avons vu dans ce chapitre quelques méthodes de recherche exacte d'un ou plusieurs motifs, issus de l'algorithmique du texte. Nous avons aussi présenté quelques algorithmes de recherche approchée et d'alignement de séquences. Et pour finir, afin d'optimiser le temps de recherche, nous avons donné quelques structures d'index les plus utilisées.

Dans le prochain chapitre nous présenterons notre contribution basée sur les k-mers.

# Chapitre 3

## Recherche de similarité à base de k-mers

La recherche de similarités -ou d'alignements- entre séquences génomiques est d'une importance fondamentale dans la recherche biologique, en particulier en biologie moléculaire et en génomique. Il s'agit de la première étape clé de l'analyse évolutive moléculaire, de la fonction des gènes et de la recherche d'homologie.

Deux types d'approches ont été mises en œuvre pour effectuer de telles comparaisons :

- La première répond au problème de manière très précise avec des outils classiques d'alignement de séquences tels que BLAST (*Altschul et al., 1990*) et Blat (*Kent, 2002*).
- La deuxième catégorie accélère fortement les calculs grâce à des méthodes de comparaison de séquences sans alignement basées sur des statistiques de mots ou des comparaisons de mots, tel que les k-mers (mots de taille  $k$ ) sont utilisés comme unité de mesure pour la recherche de similarité de deux séquences d'ADN et aussi avec de nouvelles techniques d'indexation tels que le FM-Index (*Ferragina & Manzini, 2000*) et les filtres de Bloom (*Nicolas, 2012*).

Dans ce chapitre, nous allons donc d'abord présenter une vue globale des méthodes de comparaison de séquences, avec et sans alignement. Ensuite, nous décrirons notre conception basée sur les k-mers (mots de taille  $k$ ) qui sont bien établies en bio-informatique dans la comparaison de séquences du génome entier et restent un défi permanent depuis plusieurs années.

Une mesure de similarité simple entre deux séquences peut être leur pourcentage de k-mers en commun. L'avantage des k-mers est qu'ils s'indexent et se comparent extrêmement rapidement.

Nous terminons notre conception par la méthode choisie pour la classification et l'analyse des résultats de la comparaison.

### 3.1. Comparaison des séquences d'ADN

La comparaison de séquences d'ADN est de loin la tâche informatique la plus fréquemment exécutée par les biologistes. Le but principal des opérations de comparaison est de quantifier la similitude entre les séquences pour ensuite déterminer l'information contenue dans ces portions de génomes.

Les différentes méthodes de comparaison d'ADN peuvent être regroupées en deux types d'approches :

- La comparaison par alignement.
- La comparaison sans alignement.

### 3.1.1. La comparaison par alignement

L'alignement est une manière de représenter deux ou plusieurs séquences d'ADN les unes sous les autres, de manière à en faire ressortir les régions homologues ou similaires (*Dardel & Képès, 2002*). Il permet de localiser et de mesurer la ressemblance entre deux séquences en prenant compte des mutations qui peuvent être dues à des erreurs de séquençage ou à des variantes génétiques des individus d'une même espèce.

Comme expliqué dans le premier chapitre, les séquences d'ADN évoluent au fil du temps, le but de l'alignement n'est pas seulement de mesurer le taux de caractères similaires mais, et surtout, de représenter l'ensemble des opérations permettant de transformer une séquence  $U$  en une séquence  $V$ .

Pour le faire ceci nécessite en général l'introduction des gaps (notés "—") à certaines positions dans les séquences, ces gaps correspondent soit à une insertion du caractère dans la première séquence, soit à une délétion d'un caractère dans la seconde.

Trois Situations sont possibles pour une position donnée de l'alignement :

- Les caractères sont les mêmes : Identité.
- Les caractères ne sont pas les mêmes : Substitution.
- L'une des positions est un gap : Insertion/ délétion.

Pour déterminer les endroits d'insertion et de délétion, on attribue un coût à chacune des mises en correspondance possibles. Le but sera donc de refléter les meilleures similitudes entre les séquences en choisissant la distance minimale entre les paires de séquences.

Afin de trouver l'alignement optimal entre deux séquences et retourner un score de similarité, on utilise la programmation dynamique. Il existe deux types d'alignement : global et local.

L'alignement local peut se limiter à un fragment de chaque séquence, il sert à trouver la plus forte similitude entre les deux séquences, en ignorant les différences en dehors de la région la plus similaire, c'est généralement plus approprié pour rechercher des bases de données de protéines et d'ADN, tandis que les algorithmes de comparaison globale sont plus appropriés quand l'homologie a été établie, comme lors de la construction d'arbres évolutifs (*Pearson, 2001*).

La comparaison par alignement est utilisée par plusieurs programmes connus dans le domaine de la bio-informatique, tels que FASTA (*Pearson & Lipman, 1988*), un logiciel conçu pour la recherche de similarités dans les séquences protéiques et nucléiques, ou le logiciel

ClustalW (Thompson et al., 1994) qui permet de générer un alignement multiple en se basant sur des algorithmes de programmation dynamique.

### 3.1.2. La comparaison sans alignement

La comparaison de séquences sans alignement (alignment-free) est un domaine à part entière qui date des années 80 dont la popularité a particulièrement augmenté ces dernières années tant la quantité de données générée en bio-informatique est importante (Vinga & Almeida, 2003).

Les approches sans alignement de la comparaison de séquences peuvent être définies comme toute méthode de quantification de la similarité/dissimilarité des séquences qui n'utilise pas ou ne produit pas d'alignement, la plupart de ces méthodes sont basées sur des statistiques de mots ou des comparaisons de mots, et leur évolutivité leur permet d'être appliquées à des ensembles de données beaucoup plus volumineux que les méthodes classiques basées sur l'alignement et qui reposent sur la programmation dynamique (Zielezinski et al., 2017).

Les approches sans alignement sont divisées en deux groupes :

#### a) Les méthodes basées sur l'évaluation de l'information :

Elles reconnaissent et calculent la quantité d'informations partagées entre deux séquences biologiques analysées (Zielezinski et al., 2017). En effet, les séquences nucléiques sont des chaînes de symboles, leur organisation numérique est donc interprétable avec des outils de théorie de l'information, tels que la complexité et l'entropie.

À titre d'exemple, parmi les outils de comparaison utilisant cette approche on trouve le projet « *decaf + py* » (Höhl, 2006), basé sur la mesure de complexité de « *Lempel–Ziv* » (Lempel & Ziv, 1976) et la distance W-metric (Clark & Rae, 1984). Il y a également le logiciel mBKM (Wei et al., 2012), basé sur l'entropie de Shannon et la distance euclidienne (Brillouin, 1988).

#### b) Les méthodes basées sur les fréquences de sous-séquences d'une longueur définie :

Ces algorithmes sont disponibles dans différentes formes, avec des variations méthodologiques. Leur principe général est de calculer la fréquence d'apparition de chaque k-mer dans une séquence donnée.

Ils sont très utilisés pour l'analyse des données issues du séquençage de la nouvelle génération, à titre d'exemple :

- Sailfish (Patro et al., 2014), un logiciel conçu pour l'estimation de l'abondance des isoformes à partir des séquences de référence et des données ARN-seq.
- Salmon (Patro et al., 2017), une méthode de quantification de l'abondance des transcriptions à partir de lectures d'ARN – seq.

Cette approche est également utilisée pour résoudre certains problèmes bio-informatiques, tel que l'analyse phylogénique du génome entier, le projet andi (*Haubold et al., 1999*), ainsi que dans d'autres branches du domaine.

Dans cette étude, nous allons utiliser la dernière approche en se basant sur le nombre de k-mers partagés entre les séquences car celle-ci est généralement d'une complexité linéaire dépendant uniquement de la longueur de la séquence de requête (*Pearson & Lipman, 1988*).

## 3.2. Notre stratégie de comparaison

L'objectif principal de notre travail est d'étudier les similarités entre les séquences d'ADN contenus dans un jeu de données, dans le but de démontrer les liens entre les différentes séquences et d'analyser les méthodes de leur comparaison.

Nous présenterons dans cette partie les étapes de notre contribution et nous expliquerons en détails comment nous nous sommes servis des deux approches citées précédemment (comparaison avec alignement et la comparaison sans alignement) pour sortir une matrice de scores finale contenant les résultats de comparaison (exemple dans le tableau de la section [3.2.2.1.3](#)).

### 3.2.1. Méthode de programmation dynamique

Par cette méthode, nous tentons d'effectuer un alignement global entre les paires de séquences en utilisant l'algorithme de Needleman-Wunsch, (présenté dans la section [2.4.3](#)).

Nous pouvons résumer cette approche en trois étapes : l'initialisation, le remplissage de la matrice et l'alignement.

- **L'initialisation** : Cette étape commence par la définition d'un système de score pour distinguer chaque cas possible. Cela va être réalisé par l'affectation d'un score à chaque cas (identité, substitution, indel).

Le système de score que nous avons choisi est:  $\delta(indel) = 0$ ,  $\delta(Identité) = 0$  et  $\delta(substitution) = 1$ .

Après la définition du système de score, les deux séquences ( $U$ ,  $V$ ) à comparer sont organisées dans une matrice à deux dimensions ( $n * m$ ). Où  $n$  et  $m$  représentent la taille de deux séquences respectivement.

La phase d'initialisation se termine par le remplissage de la première ligne et la première colonne. Où chaque case ( $[0,i]$  ou  $[i,0]$ ) contient le score  $s = i * indel$ .

- **Le remplissage** : Dans la matrice de score, chaque case contient un score qui correspond au score maximal de l'un des trois scénarios possibles (identité /substitution, insertion,

suppression). Et cela est implicitement calculé à partir de la formule suivante :

$$s(i, j) = \max \begin{cases} s(i-1, j-1) + \text{score}(U[i], V[j]) \\ s(i, j-1) + \text{indel} \\ s(i-1, j) + \text{indel} \end{cases} \quad \text{avec} \quad \begin{cases} \text{score}(U[i], V[j]) = \begin{cases} +2 & \text{si } U[i] = V[j] \\ -1 & \text{si } U[i] \neq V[j] \end{cases} \\ \text{et } \text{indel} = -1 \end{cases}$$

- **L'alignement** : Après le calcul de la matrice de score, chaque chemin entre la case  $(n, m)$  et la case  $(0, 0)$  est une façon d'aligner les deux séquences. On démarre de la case  $(n, m)$  et on choisit à chaque itération la case adjacente avec le score le plus élevé.

Pour la construction d'un alignement, un déplacement en diagonal correspond à une substitution ou correspondance, déplacement horizontal ou vertical constitue une insertion ou suppression dans la première séquence.

### 3.2.2. Méthode heuristique

Avec cette approche, nous allons estimer la distance entre les séquences en se basant sur les k-mers principalement utilisé dans le contexte de la génomique computationnelle, l'analyse et l'assemblage de séquences d'ADN mais peuvent également être utilisés dans l'alignement de séquences (Erki et al, 2018).

Le principe général que nous allons suivre avec cette méthode est de transformer les séquences de notre jeu de données en un vecteur qui représente les occurrences des k-mers dans deux séquences.

Afin de construire ce vecteur nous allons considérer deux approches d'indexation, une indexation globale de tout le jeu de données et une indexation par séquence, chacune se basant sur une structure d'index différente. Le résultat final sera une matrice de score contenant les taux de similarité entre les séquences.

#### **Définition d'un k-mer :**

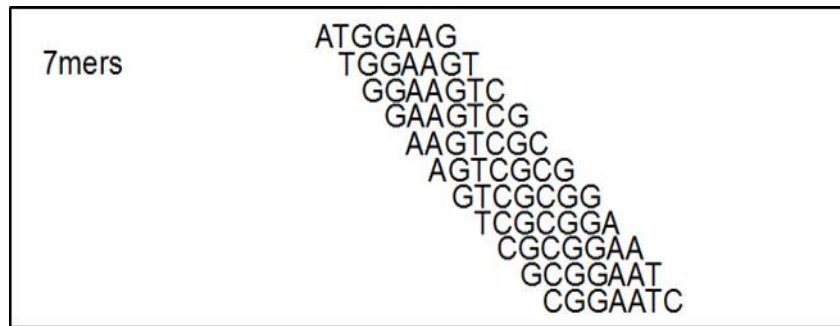
Une séquence  $s$  de  $n$  caractères appartenant à l'alphabet  $\Sigma = \{A, C, G, T\}$  est notée  $s[0], s[1] \dots s[n-1]$ . La notation  $|s|$  désigne la taille de la séquence  $s$ .

Soit  $k \in [0, n-1]$  et  $i \in [0, n-k-1]$ , le mot  $s[i], s[i+1] \dots s[i+k-1]$  est appelé un k-mer apparaissant à la position  $i$ .

Notons qu'il existe  $|s| - k + 1$  k-mers sur une séquence  $s$ . (Peterlongo et al., 2016)

**Remarque :** En règle générale, des k-mers de petites tailles doivent être utilisés lorsque les séquences sont manifestement différentes, tandis que des k-mers plus longs peuvent être utilisés pour des séquences ayant une forte similarité (Wu et al., 2005).

Exemple : Pour une séquence d'ADN  $s = \text{"ATGGAAGTCGCGGAATC"}$  et pour  $k = 7$ , l'ensemble de k-mers contenus dans cette séquence est le suivant :



**Figure 3.1:** Exemple de décomposition d'une séquence d'ADN en 7-mers.

Le grand intérêt des techniques de comparaison sans alignement est qu'elles peuvent être couplées à une étape d'indexation qui va réduire le nombre quadratique d'opérations à effectuer. Similairement à un annuaire, l'index permet de trouver facilement les informations qui nous intéressent. Dans le contexte des séquences d'ADN, indexer les k-mers d'une manière structurée est une façon de minimiser l'usage de la mémoire.

On distingue deux types d'index, ceux issus des structures discrètes de l'algorithmique du texte (voir les sections [2.5.1](#) et [2.5.2](#)), et les autres basés sur des fonctions de hachage.

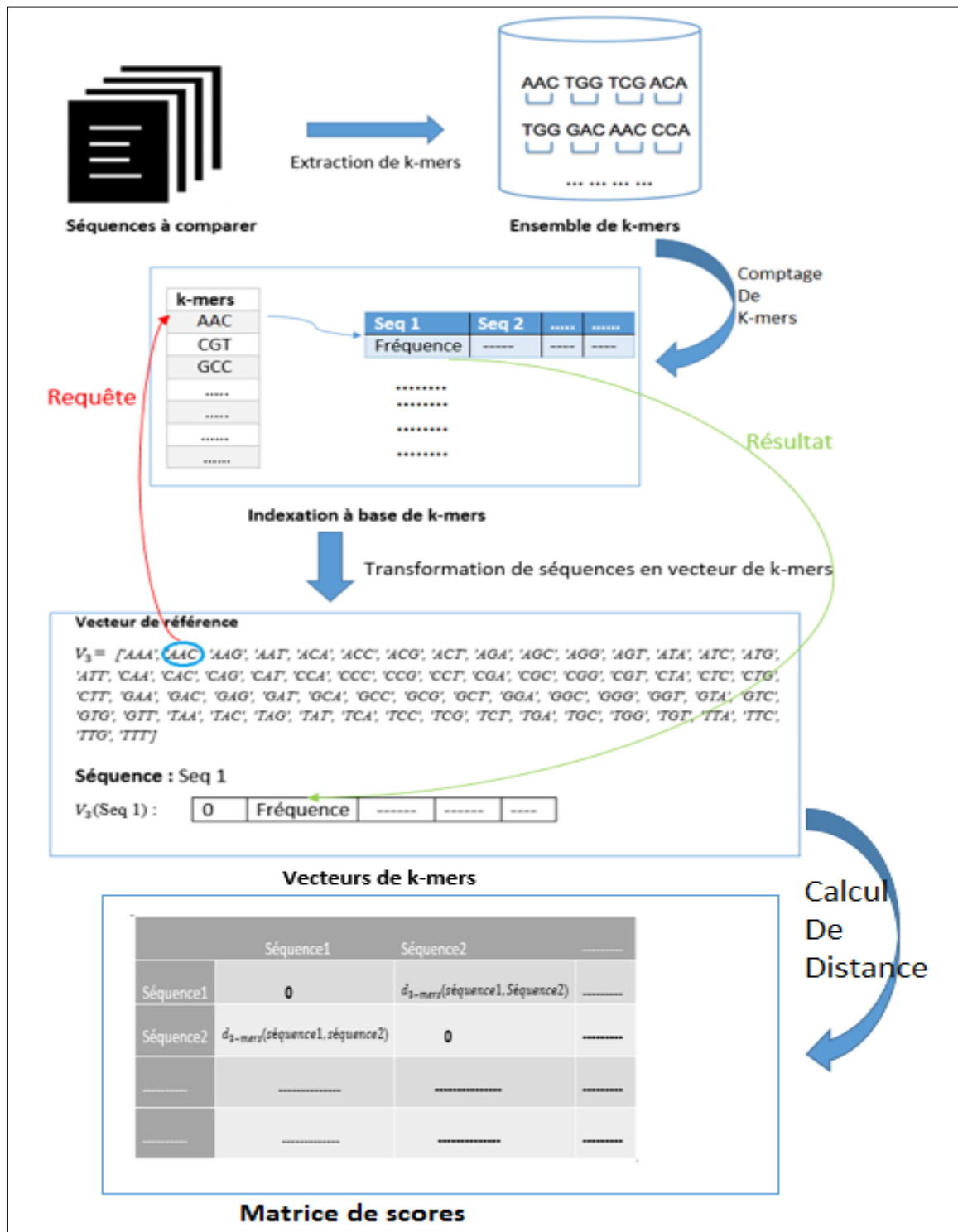
Les techniques issues des structures discrètes de l'algorithmique du texte ont l'avantage de garantir des propriétés sur les séquences recherchées. Cependant elles s'adaptent mal à la notion de recherche avec erreurs et s'avèrent donc plus coûteuses en temps et en mémoire. Pour remédier à ce problème nous allons utiliser dans notre comparaison une indexation appartenant à la deuxième famille (basée sur des fonctions de hachage).

Nous proposons deux approches d'indexation :

- Une indexation globale de tout le jeu de données, basée sur les dictionnaires.
- Une indexation par séquence, basée sur les filtres de Bloom.

### 3.2.2.1. Comparaison et indexation des k-mers avec dictionnaires

Le principe général de cette approche que nous proposons est de collecter l'ensemble de tous les k-mers contenus dans le jeu de données et de générer ensuite un score de similitude entre les séquences à partir de la comparaison de ces k-mers. La figure ci-dessous (**Figure 3.2**) schématise les phases de cette comparaison. Les étapes de cette comparaison sont les suivantes :



**Figure 3.2:** Schématisation de la comparaison et indexation des k-mers avec dictionnaires.

### 3.2.2.1.1. Comptage de k-mers

Une fois les k-mers de toutes les séquences de notre jeu de données sont collectées, nous allons effectuer un comptage afin de quantifier la fréquence de chaque k-mer et de définir les k-mers communs entre deux séquences.



Le but de cette étape est d'associer chaque k-mer aux identifiants des séquences dont il fait partie et de compter en même temps le nombre de ses répétitions. En d'autres termes, il s'agit de stocker les associations k-mers/séquences tout en permettant de définir le nombre de ces k-mers partagés entre les séquences.

Il faut noter que notre procédure de comparaison ignore complètement l'ordre des k-mers à l'intérieur des séquences. En effet, notre but est de construire un vecteur de k-mers et pour cela nous n'avons besoin que de la fréquence de chaque k-mer. Donc nous n'avons pas gardé les indices de position des k-mers.

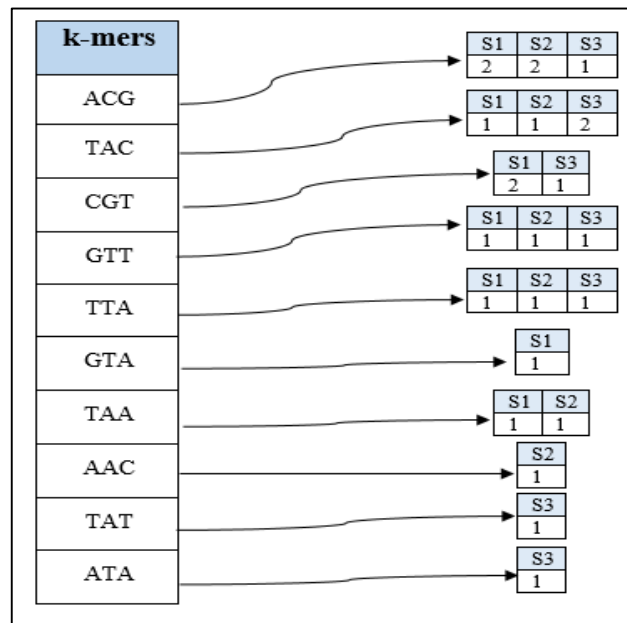
L'exemple suivant décrit formellement l'étape du comptage.

Exemple : supposons qu'on cherche à comparer 3 séquences : S1, S2 et S3, tels que :

S1= "ACGTTACGTAA"    S2= "TAACGTTACG"    S3= "TATACGTTAC"

Indexation de k-mers :

La structure de données que nous avons proposée est un dictionnaire global étiqueté par les k-mers appartenant aux trois séquences, et chaque clé (k-mer) renvoie à un dictionnaire secondaire qui contient l'indice de la séquence et le nombre de répétition du k-mer dans cette dernière.



**Figure 3.3:** Exemple d'indexation à base de k-mers.

L'algorithme suivant (**Algorithme 3.1**) résume les étapes de la construction du dictionnaire illustré dans ce qui précède.

## Fin ;

## 37

Maintenant que nous avons une représentation vectorielle pour chaque séquence avec un comptage précis de k-mers, nous pouvons comparer deux séquences en utilisant une distance classique, dans ce travail nous allons utiliser la distance euclidienne.

$$d_{k-mers}(S_1, S_2) = \sqrt{\sum_{i \in [0, \alpha^k - 1]} (V_k(S_1)[i] - V_k(S_2)[i])^2}$$

### 3.2.2.2. Comparaison et indexation des k-mers avec les filtres de Bloom

Les filtres de Bloom, déjà présentés dans la section (2.5.4) représentent une structure de données permettant de savoir si un élément est présent ou non dans une liste.

Un filtre de Bloom consiste en un tableau de  $m$  bits, tous initialisés à 0, et un ensemble de fonctions de hachage. Pour stocker un élément dans ce tableau, tous les bits associés à cet élément via les fonctions de hachage passent à 1. Pour tester la présence d'un élément, il suffit de hacher cet élément par les mêmes fonctions de hachage et de vérifier ensuite si tous ses bits sont à 1.

Dans l'exemple ci-dessous (Figure 3.4),  $m$  représente la taille du filtre,  $k$  le nombre de fonctions de hachage et  $n$  le nombre d'éléments à stocker ( $x$ ,  $y$  et  $z$ ).

Pour vérifier l'existence de l'élément  $w$ , il faut vérifier si toutes les cases qui lui correspondent dans le filtre de Bloom sont égales à 1.

Pour cela, il faut le hacher par les 3 fonctions du hachage utilisées lors du remplissage du filtre. Nous remarquons que le résultat de l'une des fonctions de hachage (celui correspondant à la case 30) est égale à 0, ce qui signifie que  $w$  ne fait pas partie des éléments stockés dans le filtre.

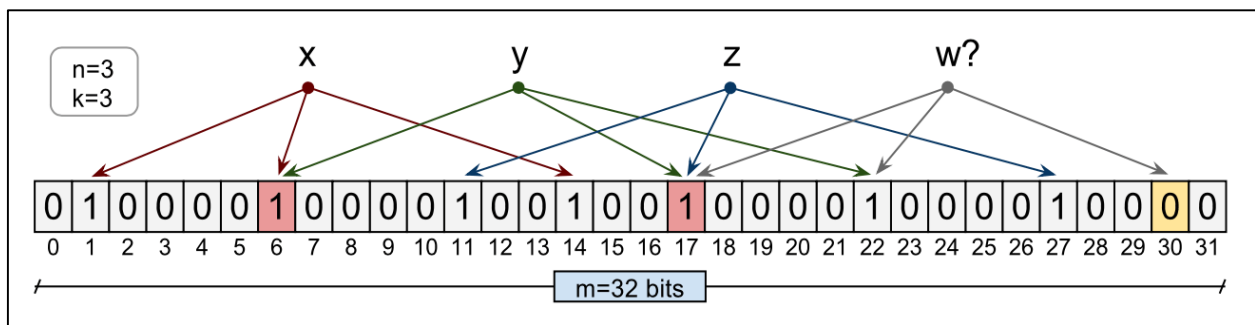


Figure 3.4: Exemple d'un filtre de Bloom.

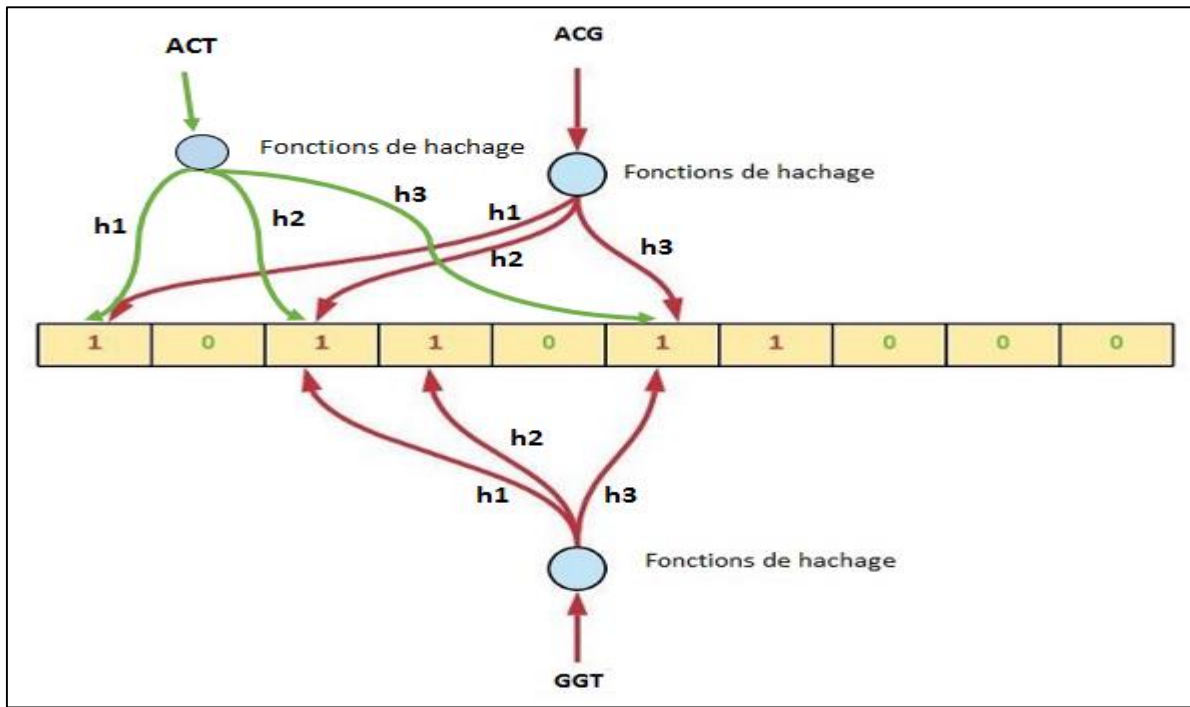
Nous avons choisi les filtres de Bloom comme structure d'indexation car ils ont l'avantage d'être très économe en mémoire et très rapide en exécution grâce à leur taille fixe et indépendante du nombre d'éléments contenus.

L'inconvénient de cette structure est l'existence des faux positifs, c'est à dire que la fonction renvoie "Vrai" alors que le mot n'est pas dans la liste, mais c'est possible de créer le filtre de façon à minimiser le nombre de faux positifs. Nous présentons dans ce qui suit les faux positifs en détails ainsi que la méthode de leur gestion.

- **Les faux positifs :**

Un faux positif est le résultat d'une prise de décision dans un choix à deux possibilités (*positif* et *négatif*), déclaré positif, là où il est en réalité négatif (Saporta, 2006).

**Exemple :** Dans le filtre suivant nous utilisons trois fonctions de hachage ( $h1$ ,  $h2$  et  $h3$ ). Nous avons stocké deux mots, "ACG" et "GGT" et on veut vérifier la présence du mot "ACT". Le résultat de la requête représente un faux positif, car tous les bits correspondants à "ACT" sont à 1 malgré que l'élément soit absent. Ceci est dû à une collision lors du hachage, c'est-à-dire que deux données ont un résultat identique avec la même fonction de hachage.



**Figure 3.5:** Exemple d'un faux positif dans un filtre de Bloom.

- **La gestion des faux positif :**

Le taux des faux positifs dépend de trois paramètres :

$m$  : La taille du vecteur booléen (taille du filtre).

$n$  : Le nombre d'éléments à insérer.

$k$  : Le nombre de fonctions de hachage.

Notre but est de trouver une taille  $m$  adéquate au taux d'erreur souhaité et aux nombres d'éléments du filtre. Ce paramètre peut être fixé à partir d'une formule qui détermine la proportion des faux positifs (*Christensen et al., 2010*).

Supposons qu'une fonction de hachage sélectionne chaque position du filtre avec une probabilité égale. La probabilité qu'une fonction de hachage fasse passer la valeur du filtre de 0 à 1 est de:  $\frac{1}{m}$ .

L'inverse, c'est-à-dire la probabilité que la valeur de  $j$  ne change pas, est donc de :  $f = 1 - \frac{1}{m}$ .

Si l'on généralise ceci à  $n$  élément et  $k$  fonctions de hachage, cette probabilité devient :

$$f = \left(1 - \frac{1}{m}\right)^{kn}$$

Nous savons que :  $\lim_{m \rightarrow \infty} \left(1 + \frac{1}{m}\right)^m = e$ .

On peut conclure donc que :  $\left(1 - \frac{1}{m}\right)^k = \left(\left(1 - \frac{1}{m}\right)^m\right)^{\frac{k}{m}} \approx e^{-\frac{k}{m}}$ .

Avec  $n$  élément à insérer, la probabilité qu'un élément ne soit pas dans le filtre ( $bit = 0$ ) est :

$$\left(1 - \frac{1}{m}\right)^{kn} \approx e^{-\frac{kn}{m}}$$

La probabilité que l'élément soit présent est donc :  $1 - \left(1 - \frac{1}{m}\right)^{kn} \approx 1 - e^{-\frac{kn}{m}}$ .

Au final, la probabilité d'avoir des faux positifs équivaut à la probabilité d'avoir toutes les positions du vecteur booléen à 1, pour les  $k$  fonctions de hachage.

On obtient ainsi la formule finale suivante :

$$\varepsilon = \left(1 - \left[1 - \frac{1}{m}\right]^{kn}\right)^k \approx \left(1 - e^{-\frac{kn}{m}}\right)^k$$

Le nombre de fonctions de hachage  $k$ , qui minimise la probabilité de faux positifs est :

$$k = \frac{m}{n} \ln 2. \text{ (Starobinski et al., 2003)}$$

Le nombre requis de bits  $m$ , étant donné  $n$  (le nombre d'éléments insérés), et une probabilité désirée  $\varepsilon$ , de faux positifs (et en supposant que la valeur optimale de  $k$  est utilisée), peuvent être calculées en remplaçant  $k$  par sa valeur optimale dans l'expression de probabilité mentionnée ci-dessus :

$$\varepsilon = \left(1 - e^{-\left(\frac{m}{n} \ln 2\right) \frac{n}{m}}\right)^{\frac{m}{n} \ln 2}$$

Qui peut être simplifié à :  $\ln \varepsilon = -\frac{m}{n} (\ln 2)^2$ .

Ainsi, pour  $n$  éléments à insérer, la taille du filtre  $m$  nécessaire pour atteindre le taux d'erreur souhaité  $\varepsilon$  est :  $m = -n \frac{\ln \varepsilon}{(\ln 2)^2}$ .

- **Les fonctions de hachage :**

Une fonction de hachage est une fonction qui, à partir d'une entrée renvoie une « *empreinte* » ou encore une « *signature* » permettant d'identifier l'entrée. Les fonctions de hachage sont utilisées dans de nombreux cas, notamment en cryptographie ou dans les structures de données de type dictionnaire (Schutz, 2016).

Dans le cas des filtres de Bloom, une fonction de hachage renvoie un unique entier compris entre 0 et  $n$ , choisi de façon uniforme. Et on peut créer autant de fonction de hachage qu'on le désire.

Les fonctions de hachage utilisées dans un filtre Bloom doivent être indépendantes, reproductibles et uniformément distribuées. Pour ces raisons nous allons utiliser les fonctions de hachage de SHA-1 (National Security Agency, 1995).

SHA-1(Secure Hash Algorithm) : une fonction de hachage cryptographique. Tous les algorithmes de cette famille génèrent des nombres pseudo-aléatoires, répartis uniformément sur l'intervalle de

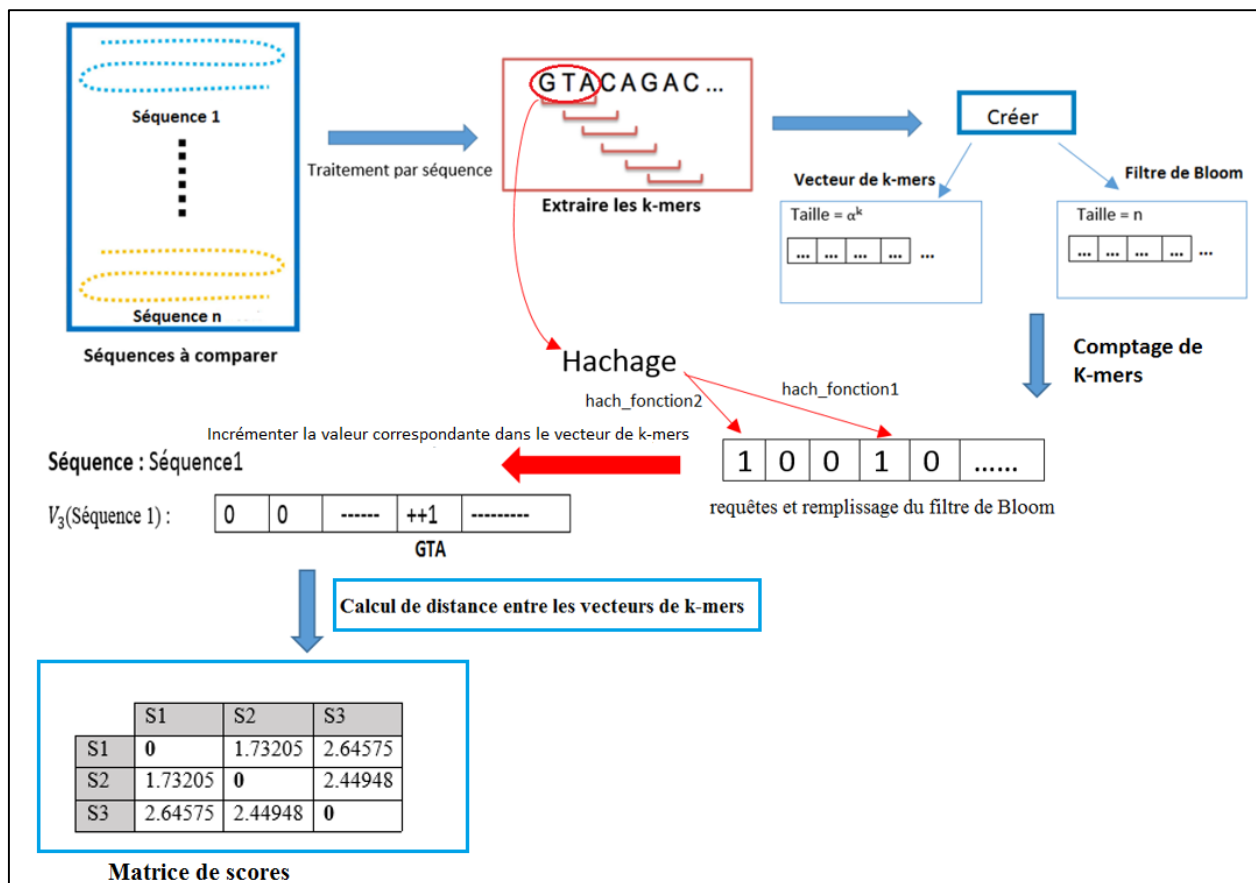
sortie, et donnent la garantie que deux clefs différentes produiront des valeurs indépendantes, au sens statistique.

### 3.2.2.2.1. Comptage de k-mers par les filtres de Bloom

Nous avons précédemment évoqué la notion de vecteur de k-mers utilisé pour calculer la distance entre deux séquences. La méthode que nous proposons dans cette section est d'utiliser les filtres de Bloom pour compter les k-mers d'une séquence et de la transformer ensuite en vecteur.

Le comptage s'effectue en deux étapes : la phase d'indexation et la phase de requête. Les deux phases vont se dérouler en parallèle.

Pour chaque séquence d'ADN  $s$ , on crée un filtre de Bloom de taille  $m$  et un vecteur de k-mers de taille  $\alpha^k$ . Ensuite, pour chaque k-mer, appartenant à la séquence  $s$ , on effectue la phase de requête sur le filtre de Bloom. Si le k-mer est déjà stocké dans le filtre, on incrémente la valeur correspondante dans le vecteur de k-mers. Sinon, on ajoute le k-mer au filtre de Bloom. Le principe général de cette approche est résumé dans la figure qui suit (Figure 3.6).



**Figure 3.6:** Schématisation de la comparaison et l'indexation des k-mers avec filtres de Bloom.

Exemple : Soit la séquence  $S = \text{"ACGACG"}$  en choisissant  $k = 3$ , le vecteur de k-mers  $V_3$  contenant tous les 3-mers possibles est le suivant :

$V_3 = [\text{'AAA'}, \text{'AAC'}, \text{'AAG'}, \text{'AAT'}, \text{'ACA'}, \text{'ACC'}, \text{'ACG'}, \text{'ACT'}, \text{'AGA'}, \text{'AGC'}, \text{'AGG'}, \text{'AGT'}, \text{'ATA'}, \text{'ATC'}, \text{'ATG'}, \text{'ATT'}, \text{'CAA'}, \text{'CAC'}, \text{'CAG'}, \text{'CAT'}, \text{'CCA'}, \text{'CCC'}, \text{'CCG'}, \text{'CCT'}, \text{'CGA'}, \text{'CGC'}, \text{'CGG'}, \text{'CGT'}, \text{'CTA'}, \text{'CTC'}, \text{'CTG'}, \text{'CTT'}, \text{'GAA'}, \text{'GAC'}, \text{'GAG'}, \text{'GAT'}, \text{'GCA'},$

'GCC', 'GCG', 'GCT', 'GGA', 'GGC', 'GGG', 'GGT', 'GTA', 'GTC', 'GTG', 'GTT', 'TAA', 'TAC', 'TAG', 'TAT', 'TCA', 'TCC', 'TCG', 'TCT', 'TGA', 'TGC', 'TGG', 'TGT', 'TTA', 'TTC', 'TTG', 'TTT']

Le vecteur  $V_3(S)$  correspondant à la séquence  $S$  est donc de taille 64.

Avant de créer le filtre de Bloom nous devons d'abord calculer sa taille ainsi que le nombre des fonctions de hachage à utiliser pour le remplir, ceci en utilisant la formule de gestion des faux positifs expliquée précédemment.

Les paramètres d'entrée pour obtenir la taille  $m$  et le nombre de fonctions de hachage  $k$  sont : le nombre d'éléments à stocker  $n$  et le taux de faux positifs  $\epsilon$ .

Rappelons que les filtres de Bloom sont une structure probabiliste. Pour garantir des résultats presque exacts nous allons accepter 0,1% de faux positifs, c'est-à-dire un taux  $\epsilon = 0.1$ .

- Le choix du nombre d'éléments à stocker :

Les éléments à stocker dans le filtre sont les k-mers de la séquence, c'est-à-dire que dans le pire des cas (où tous les k-mers de la séquence sont distincts),  $n = \text{la taille de la séquence} - k + 1$ , mais dans le cas où la taille est trop longue (une séquence de taille 120 par exemple) nous savons bien que le nombre de k-mers distincts ne va pas dépasser le nombre de k-mers possibles ( $\alpha^k$ ).

Pour éviter ce dernier cas nous allons, pour chaque séquence, prendre la taille minimale entre le nombre de k-mers possibles et le nombre de k-mers de la séquence.

Donc,  $n = \min(\alpha^k, \text{la taille de la séquence} - k + 1)$ .

- La création du filtre de Bloom et du vecteur de k-mers :

Revenons à notre exemple,  $n = \min(64, 4) = 4$ .

La taille  $m$  du filtre de Bloom pour la séquence  $S$  est donc :  $m = -n \frac{\ln \epsilon}{(\ln 2)^2} = -4 \frac{\ln 0.1}{(\ln 2)^2} = 20$

Le nombre de fonctions de hachage est :  $k = \frac{m}{n} \ln 2 = \frac{20}{4} \ln 2 = 4$ .

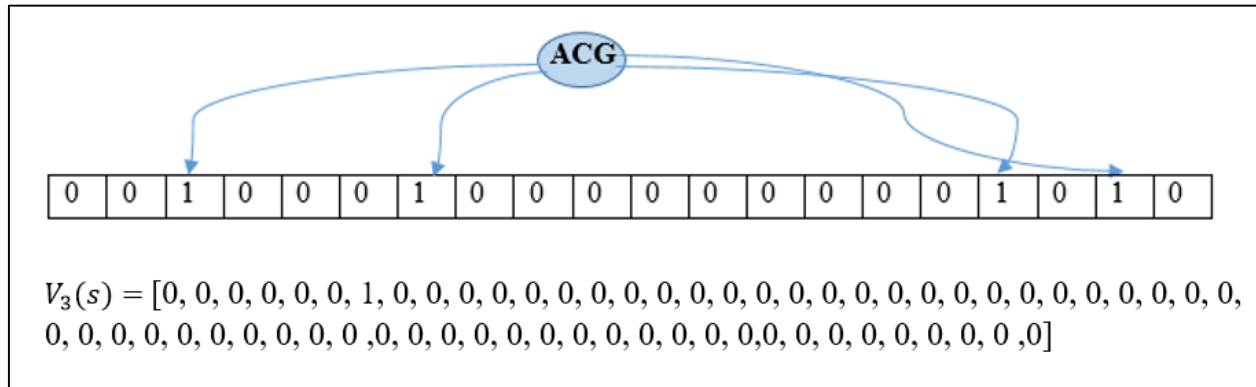
Le filtre de Bloom correspondant est :

0	1	1	0	1	1	1	0	0	0	0	1	0	0	1	1	1	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Lors de l'ajout des éléments au filtre nous remplissons en même temps le vecteur de k-mers.



Exemple de la 1ère itération :



**Figure 3.7:** Exemple de la 1ère itération lors d’une création d’un vecteur de k-mers.

Dans l'exemple illustré ci-dessus, en utilisant les fonctions de hachage SHA-1 générées automatiquement par le module « *hashlib* », nous avons obtenus les résultats suivants pour le premier k-mer ("ACG") de notre séquence :

## h0 ("ACG") = 18

# h1 ("ACG") = 16

$$h_2(\text{"ACG"}) = 2$$

### h3 ("ACG") = 6

Après avoir hacher et compter tous les k-mers de la séquence S, le résultat final et le vecteur de k-mers suivant :

[illegible]

#### 3.2.2.2.2. Calcul de distance

Maintenant que nous avons effectué le comptage et obtenus un vecteur de k-mers le calcul de la distance sera le même expliqué dans la section (3.2.2.1.3), c'est-à-dire que nous calculons la distance euclidienne entre les vecteurs pour obtenir à la fin une matrice de distance.

Comme nous pouvons constater, bien que les deux approches suivent le même principe de calcul de distance, les résultats peuvent changer, en terme de complexité et de précision, en fonction des deux différentes manières de comptage de k-mers qu'elles comportent.

La méthode basée sur les dictionnaires nous garantit des taux de similarité exactes à 100% contrairement à celle basée sur les filtres de Bloom où nous gardons une légère probabilité d'erreur. Cependant, l'indexation de tout le jeu de données peut s'avérer coûteuse en espace mémoire surtout dans le cas de bases de données volumineuses.

Pour profiter pleinement des avantages de l'approche basée sur l'indexation par filtres de Bloom, le cas idéal serait de traiter des séquences longues ayant des k-mers assez distincts, de manière à ce que le nombre d'éléments à stocker dans le filtre ne soit pas très différent du nombre d'éléments que nous considérons le pire des cas pour l'estimer.

Après avoir exposé les différentes approches de comparaison, nous allons utiliser les résultats obtenus pour les appliquer au domaine biologique de la phylogénie, et ce dans le but de trouver les relations de parenté entre les séquences comparées.

### 3.3. Analyse phylogénétique

Dans cette partie nous allons inférer des connaissances biologiques à partir des résultats de la comparaison que nous avons effectuée. En partant du principe que les séquences reliées d'un point de vue évolutif sont plus susceptibles de présenter une certaine forme de similarité, nous allons utiliser la matrice de similarité pour construire un arbre phylogénétique dont les feuilles représentent les séquences comparées (voir section [1.4](#)).

Il existe plusieurs méthodes de clustering<sup>8</sup> pour classer les séquences selon leur ressemblance. Pour notre projet, nous allons utiliser la méthode UPGMA.

#### 3.3.1. Méthode UPGMA

UPGMA (Unweighted Pair Group Method with Arithmetic Mean)<sup>9</sup> est une méthode agglomérative<sup>10</sup>, de classification hiérarchique attribuée à Sokal et Michener (Cleuziou, 2004). C'est l'une des méthodes les plus utilisées pour construire un arbre phylogénétique enraciné, elle est basée sur la méthode de liaison moyenne. L'algorithme fonctionne par itérations successives, qui réduisent progressivement la taille de la matrice.

À chaque étape nous regroupons les deux éléments les plus proches, ces éléments sont associés dans l'arbre et sont remplacés par un seul cluster.

Les nouvelles distances entre ce cluster et les éléments restants dans la matrice sont recalculées par la moyenne arithmétique des deux éléments regroupés. En d'autres termes, à chaque étape de regroupement, la distance est mise à jour entre les éléments joints et un nouveau cluster est donnée par la moyenne des distances.

L'arbre est construit « *de bas en haut* » c'est-à-dire qu'on part des feuilles et à chaque étape on rajoute un nœud au-dessus des précédents jusqu'à arriver à la racine.

- L'algorithme UPGMA :

Initialisation :

---

<sup>8</sup> Le clustering : classification non supervisée.

<sup>9</sup> Méthode des groupes de paires non pondérés avec moyenne arithmétique.

<sup>10</sup> Classification ascendante.

- Assigner chaque  $x_i$  à son propre cluster  $C_i$  ;
- Définir une feuille par espèce, chacune de hauteur 0 ;

Itération :

- Trouver deux clusters  $C_i$  et  $C_j$  tels que  $d_{ij}$  soit minimale ;
- Soit  $C_k = C_i \cup C_j$  // (le nouveau cluster) ;
- Ajouter un noeud qui connecte  $C_i$ ,  $C_j$  et le placer à une hauteur  $d_{ij}/2$  ;
- Effacer  $C_i$  et  $C_j$  ;
- Calculer la distance entre le nouveau cluster  $C_k$  et tous les autres comme une moyenne pondérée des distances entre les composantes ;

Terminaison :

- Quand on obtient un seul cluster ;

Exemple : La construction d'un arbre phylogénétique avec la méthode UPGMA :

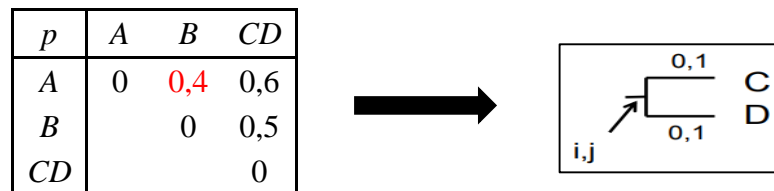
Soit une matrice de scores  $M$  contenant les résultats de comparaison entre 4 séquences d'ADN :  $A, B, C$  et  $D$ .

M :

$P$	$A$	$B$	$C$	$D$
$A$	0	0,4	0,6	0,6
$B$		0	0,5	0,5
$C$			0	0,2
$D$				0

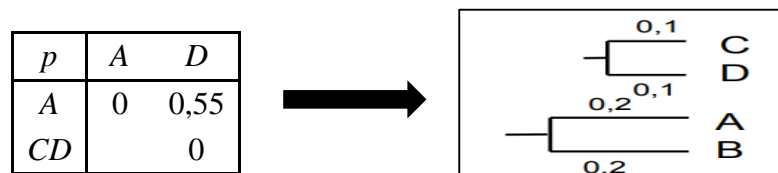
**Tableau 3.2:** Exemple d'application de la méthode UPGMA (itération 0).

Itération 1 : Connecter  $C$  et  $D$  et attribuer aux branches  $L_i$  et  $L_j$  la longueur  $D_{ij}/2$ , calculer ensuite la distance entre le nouveau groupe ( $C, D$ ) et tous les autres groupes.



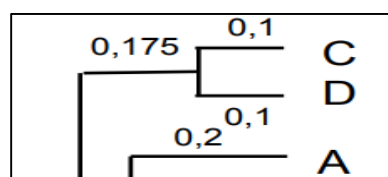
**Figure 3.8:** Exemple d'application de la méthode UPGMA (itération 1).

Itération 2 : De la même façon que l'étape précédente, relier  $A$  et  $B$  et mettre à jour la matrice de scores.



**Figure 3.9:** Exemple d'application de la méthode UPGMA (itération 2).

L'arbre phylogénétique final :



**Figure 3.10:** Exemple d'un arbre phylogénétique.

Le dendrogramme présenté ci-dessus, représentant l'arbre phylogénétique, permet de figurer la distance génétique entre les séquences étudiées de manière à pouvoir en déduire les liens de parenté entre elles. Dans le cas de données réelles, de tels résultats pourront être utilisés dans divers domaines tel que l'épidémiologie, la génomique fonctionnelle, ... etc., et permettrons donc de résoudre plusieurs questions débattues de longue date.

### **3.4. Conclusion**

La comparaison de séquences est un problème à part entière qui ne cesse de se développer. Les différentes méthodes développées pour ce problème peuvent être résumées en deux grandes catégories : une approche de programmation dynamique basée sur l'alignement de séquences, et une autre approche plus récente qui tient à utiliser de nouvelles méthodes basées principalement sur la fréquence des mots. Nous avons présenté deux démarches à suivre, une de chaque catégorie. Nous avons également décrit comment traduire les liens entre les séquences dans un arbre phylogénétique.

Dans le chapitre suivant nous allons présenter les grands axes de la réalisation de notre application et les spécifications matérielles et logicielles requis dans le développement de notre application ainsi que les tests et résultats obtenus.

# 4

## Chapitre

### Mise en œuvre et expérimentation

Maintenant que nous avons une analyse approfondie des méthodes de comparaisons d'ADN que nous souhaitons réaliser, le présent chapitre consistera à exposer l'application que nous avons développée, ainsi que les résultats obtenus après son implémentation.

Puis, nous allons effectuer plusieurs expériences sur des données biologiques réelles dans le but d'étudier les performances de chaque méthode et d'aboutir à une analyse comparative finale.

#### 4.1. Ressources matérielles et logicielles

##### 4.1.1. Ressources matérielles

Pour la réalisation de notre application et les tests, nous avons utilisé les machines dont les caractéristiques sont présentées dans le tableau qui suit :

	Rôle	Composante	Description
<b>1<sup>ère</sup> machine</b>	Utilisée pour l'implémentation	Processeur	Intel Core 2 Duo CPU 3,07 GHz
		RAM	4 Go
<b>2<sup>ème</sup> machine</b>	Utilisée pour la réalisation des tests	Processeur	Intel Core i7 CPU 2,69 GHz
		RAM	8 Go

**Tableau 4.1:** Les caractéristiques techniques des machines utilisées.

##### 4.1.2. Ressources logicielles

###### 4.1.2.1. Langage de programmation Python

Python est un langage de programmation aux sources ouvertes, généraliste et orienté objet. Il est couramment utilisé aussi bien pour écrire des applications complètes autonomes que des scripts, cela dans de nombreux domaines et par des milliers de développeurs.

Les programmes python fonctionnent sur la plupart des plates-formes couramment utilisées, des macros-ordinateurs aux super calculateurs, sous les systèmes Unix et linux, Windows et Macintosh, et bien d'autre encore (*Lultz & Bailly, 2005*).

Python est un langage polyvalent qui admet de multiples applications. L'une d'entre elles est la bio-informatique, grâce aux bibliothèques de traitement de données. Le module qui nous a servi le plus dans le développement de l'application est Biopython.

- **Biopython :**

Un ensemble d'outils écrits en Python dédiés à la manipulation de données biologiques dans le cadre de l'analyse complète d'une séquence d'ADN.

La principale force de Biopython sont ses « parsers », des modules capables de lire et manipuler les formats standards de données biologiques les plus répandus. La possibilité d'accéder automatiquement aux bases de données en ligne et d'en utiliser les outils permet d'intégrer facilement des traitements dans des workflows d'analyse automatisés (*Dameron & Farrant, 2014*).

Nous présentons dans le tableau ci-dessous (**Tableau 4.2**) le reste des modules que nous avons utilisé pour la programmation de l'application.

Module	Intérêt
Biotie	Pour le traitement des arbres phylogénétiques.
Bokeh	Pour la visualisation des alignements et des graphes.
Hashlib	Génère des fonctions de hachage.
Itertools	Fournit de nombreuses briques d'itérateurs pour le traitement des données.
Math	Pour le calcul des équations.
Numpy	Pour effectuer des calculs sur des vecteurs ou des matrices.
Os	Pour utiliser les fonctionnalités dépendantes du système d'exploitation.
Werkzeug	Pour la gestion sécurisée de formulaires HTML.

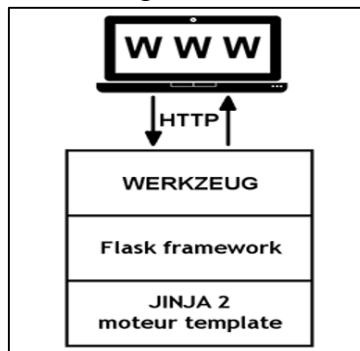
**Tableau 4.2:** Modules utilisés pour le développement de l'application.

Python est également utilisé dans le développement web, notamment grâce à ses frameworks, entre autre, le framework Flask.

#### 4.1.2.2. Framework Flask

Flask est un micro framework de développement web écrit en Python. Allant à contre-pied d'autres solutions de développement web, Flask est livré avec le strict minimum (*Meurisse, 2018*), à savoir :

- Un moteur de template (Jinja 2).
- Un serveur web de développement (Werkzeug).
- Un système de distribution de requête compatible REST (dit RESTful).
- Un support de débogage intégré au serveur web.
- Un micro framework doté d'une très grande flexibilité.



**Figure 4.1:** Structure du micro framework Flask.

## 4.2. Architecture de l'application

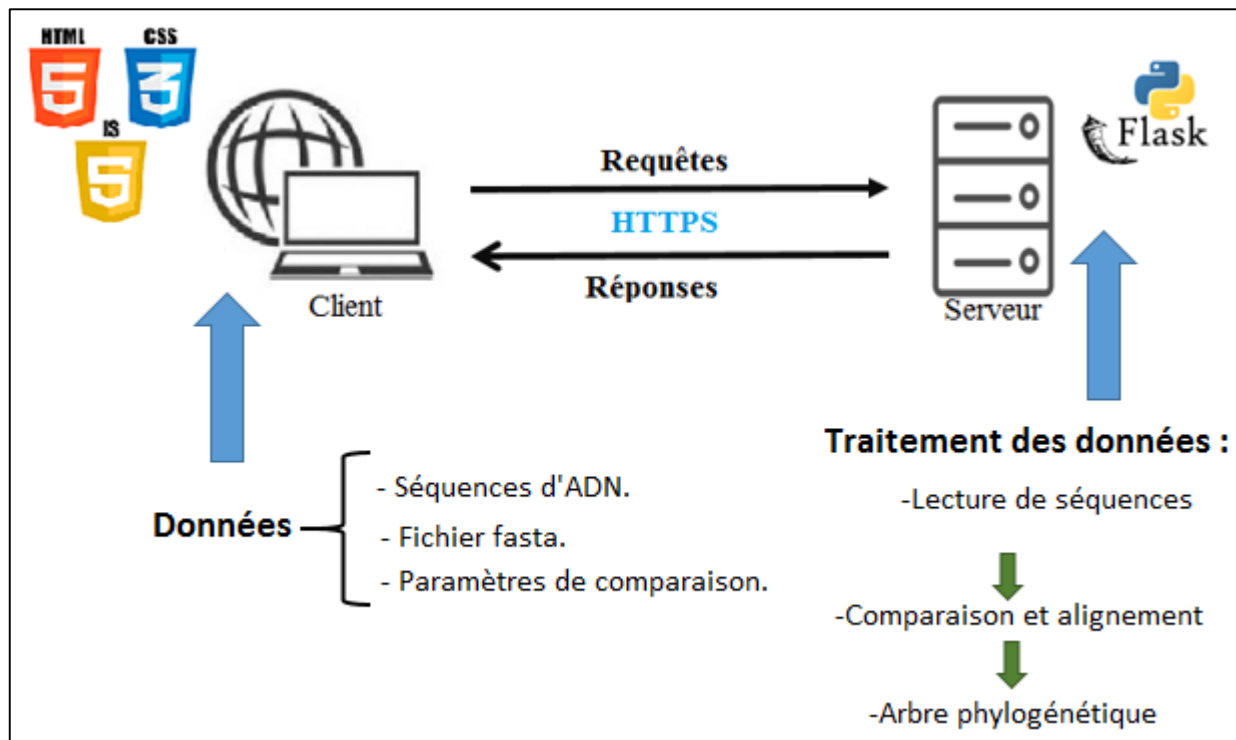
L'architecture de notre application est une architecture de client-serveur. Cette dernière est un modèle de fonctionnement logiciel qui se réalise sur tout type d'architecture matérielle (petites à grosses machines), à partir du moment où ces architectures peuvent être interconnectées.

On parle de fonctionnement logiciel dans la mesure où cette architecture est basée sur l'utilisation de deux types de logiciels, à savoir un logiciel serveur et un logiciel client s'exécutant normalement sur 2 machines différentes.

L'élément important dans cette architecture est l'utilisation du mécanisme de communication entre les 2 applications. Le dialogue entre les applications peut se résumer par :

- Le client demande un service au serveur.
- Le serveur réalise ce service et renvoie le résultat au client. (*Omari, 2010*)

Ceci est schématisé dans la figure qui suit (**Figure 4.2**) dans laquelle nous présentons de manière globale notre application.



**Figure 4.2:** Architecture de l'application.

### 4.3. Présentation de l'application

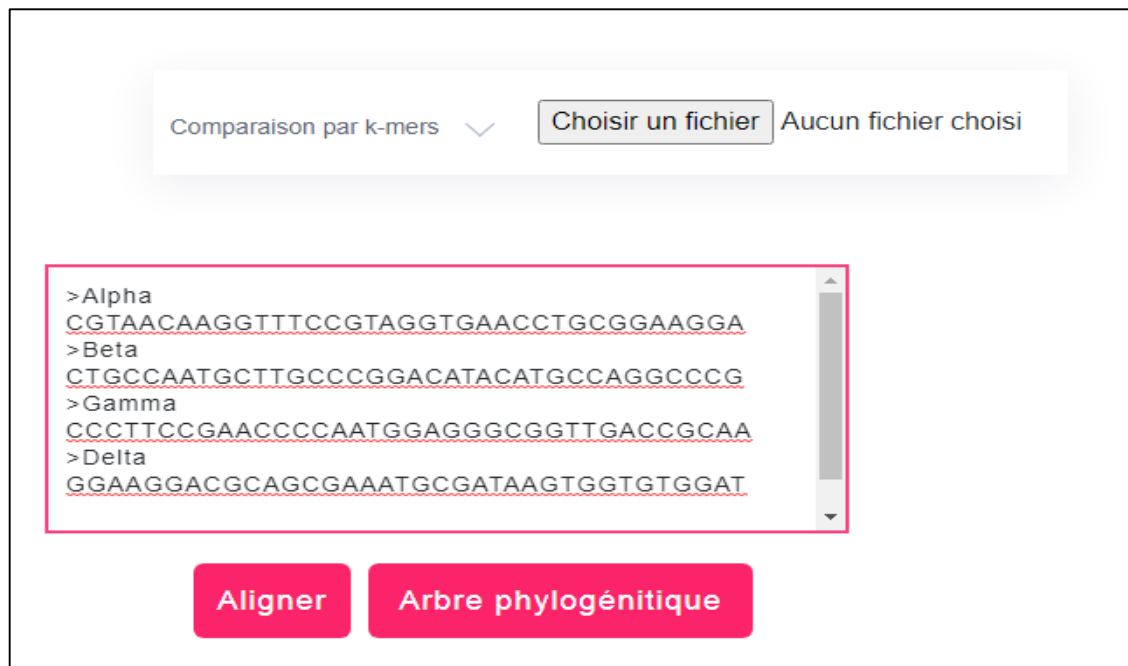
Nous présentons dans ce qui suit les principales fonctionnalités de notre application.

#### 4.3.1. Lecture de données

Pour pouvoir bénéficier des fonctionnalités de l'application, l'utilisateur doit introduire les séquences d'ADN qu'il souhaite comparer. Ces séquences peuvent être contenues dans un fichier Fasta, comme elles peuvent être saisies directement dans la zone du texte.

La figure suivante (**Figure 4.3**) représente l'interface correspondante.



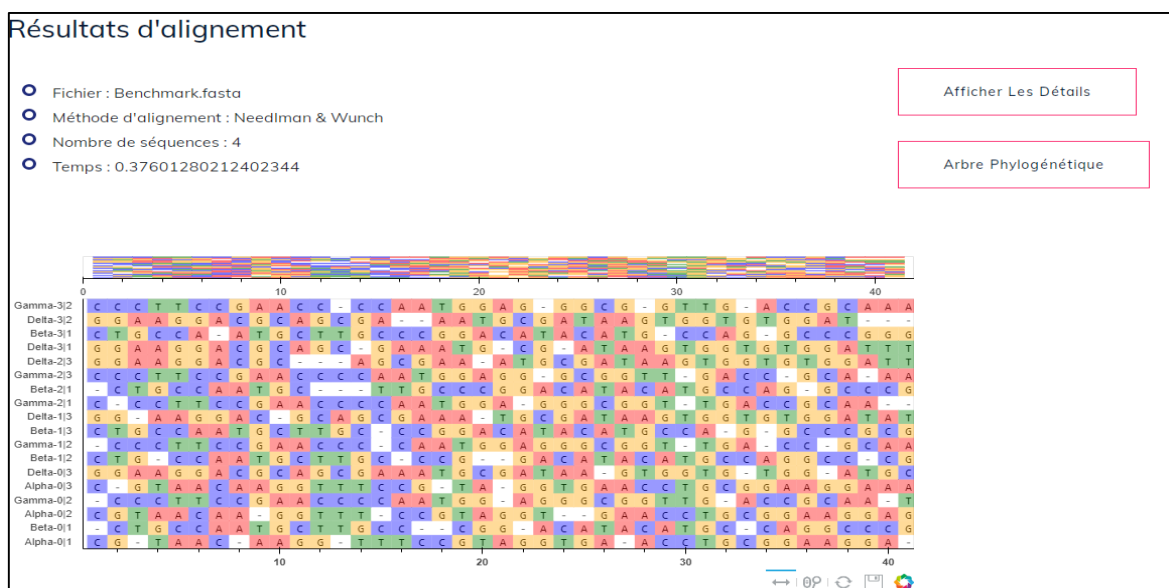


**Figure 4.3:** Lecture de données via l'application.

Tel que l'interface l'indique, l'utilisateur a également la possibilité de choisir la méthode de comparaison qu'il souhaite parmi les trois méthodes que nous avons implémentées. Pour les résultats à afficher, l'utilisateur a le choix d'afficher directement l'arbre phylogénétique sans passer par les détails de la comparaison ou bien de visualiser l'alignement avant.

#### 4.3.2. Affichage de l'alignement

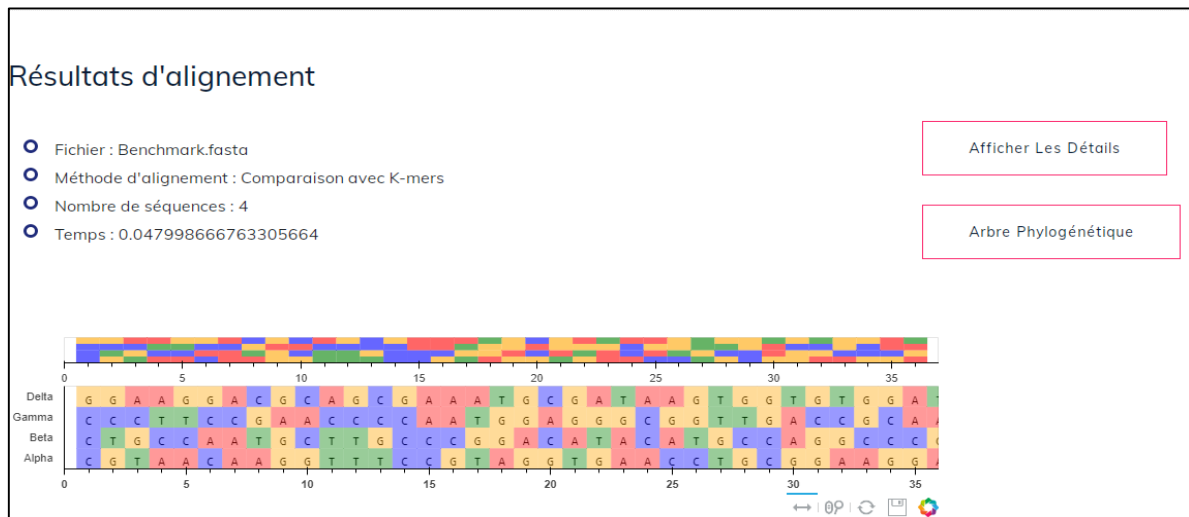
Une fois la comparaison effectuée, on peut visualiser le taux de similarité entre les paires de séquences, ainsi que les transformations apportées aux séquences dans le cas de l'alignement par Needleman-Wunsch.



**Figure 4.4:** Résultat d'un alignement par paire en utilisant Needleman-Wunsch.

L'interface présentée dans la figure précédente (**Figure 4.4**) illustre le résultat de comparaison de 4 séquences d'ADN. Puisque nous effectuons un alignement par paires (entre chaque 2 séquences), nous avons obtenus 4 paires d'alignement pour notre exemple.

Dans le cas des méthodes de comparaison basées sur les k-mers, il n'y aura pas de modifications apportées aux séquences (pas d'insertion de gaps). Nous allons donc nous contenter d'afficher les séquences à comparer juste pour visualiser la ressemblance grâce aux couleurs affectées à chaque base nucléique.



**Figure 4.5:** Visualisation d'une comparaison de séquences à base de k-mers.

Ces résultats illustrant la comparaison peuvent être sauvegardés sous forme d'image, si l'utilisateur le souhaite.

#### 4.3.3. Détails de la comparaison

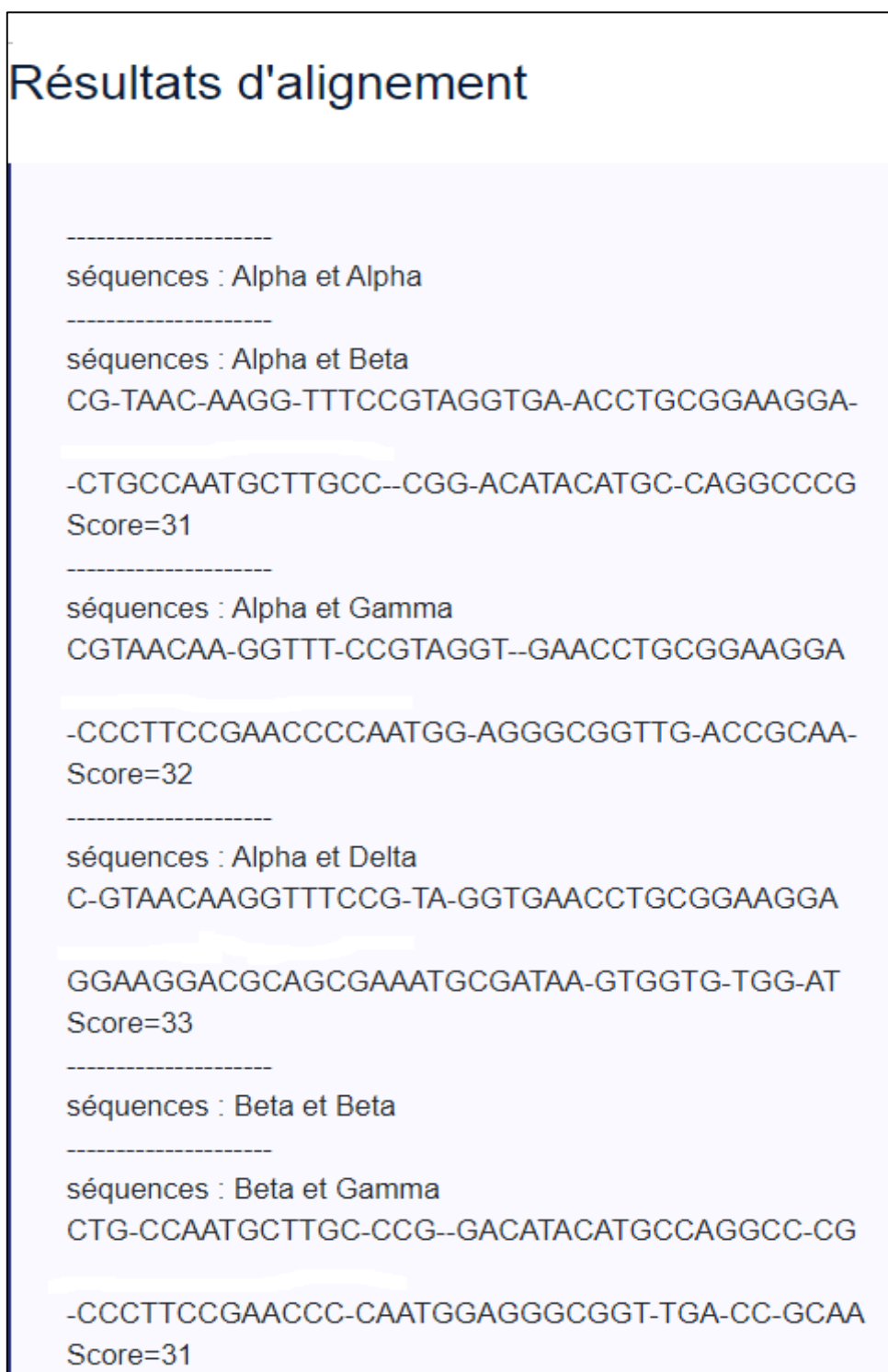
Une autre fonctionnalité de l'application est l'affichage des détails de la comparaison, à partir de la matrice de scores qui détermine le taux de similarité entre les paires de séquences.

Matrice de scores

	Alpha	Beta	Gamma	Delta
Alpha	0.0	9.5916	7.2111	7.7459
Beta	9.5916	0.0	8.2462	10.0
Gamma	7.2111	8.2462	0.0	8.2462
Delta	7.7459	10.0	8.2462	0.0

**Figure 4.6:** Matrice de scores résultante de la comparaison de 4 séquences d'ADN.

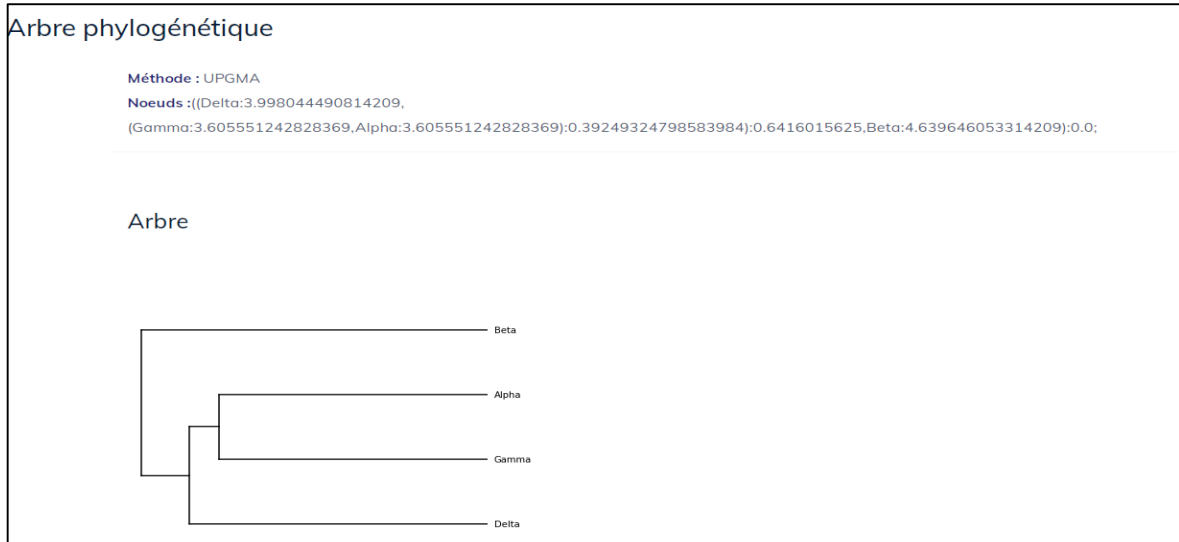
Dans le cas d'un alignement par Needleman-Wunsch, un fichier contenant les détails de l'alignement par paires sera automatiquement sauvegardé à chaque alignement. Les données de ce fichier peuvent également être consultées à partir de l'application telle que le démontre l'interface illustrée ci-dessous.



**Figure 4.7:** Présentation d'une partie des détails d'un alignement de séquences.

#### 4.3.4. Affichage de l'arbre phylogénétique

À partir de la page de l'application, présentée ci-dessous, on peut visualiser l'arbre phylogénétique issu des résultats de la comparaison ainsi que les nœuds (qui illustrent les séquences) et les distances qui les séparent.



**Figure 4.8:** Présentation de l'arbre phylogénétique.

#### 4.3.5. Les tests de fonctionnement

Afin d'atteindre le niveau de qualité souhaité, nous avons testé dans cette phase les cas des erreurs qui peuvent être générées par les données saisies par l'utilisateur.

Nous présentons dans ce qui suit les conditions nécessaires pour la validité de l'insertion des données ainsi que les messages d'erreurs générés dans le cas où ces conditions ne sont pas respectées.

1. Il faut saisir des séquences ou bien insérer un fichier avant de confirmer le formulaire d'envoi.

Fichier ou séquences introuvables

Comparaison par k-mers

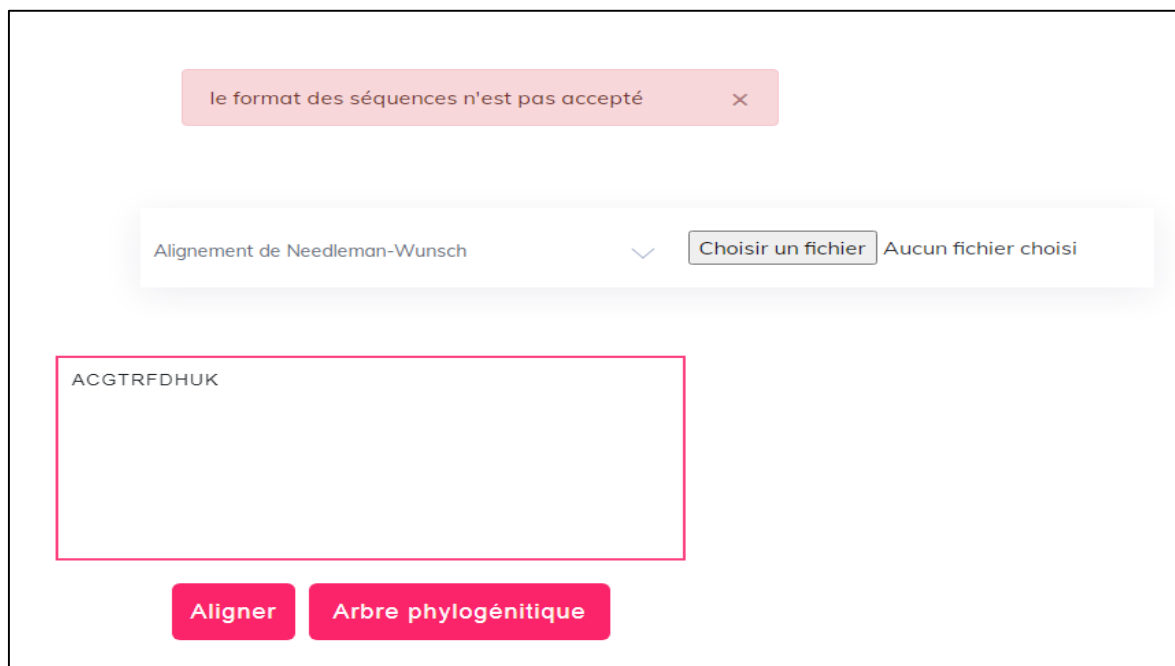
Choisir un fichier

Aucun fichier choisi

Aligner

Arbre phylogénétique

**Figure 4.9:** Erreur générée lorsque aucun fichier ni aucune séquence ne sont introduits.

**2. Le fichier doit être sous format FASTA.****Figure 4.10:** Erreur générée si le format du fichier n'est pas autorisé.**3. Les séquences saisies doivent respecter le format requis (avoir une clé et appartenir à l'alphabet {A, C, G, T})****Figure 4.11:** Erreur générée si le format de séquences n'est pas valide.

4. Il doit y avoir au moins deux séquences à comparer.

**Figure 4.12:** Erreur générée si le nombre de séquences saisies est insuffisant.

## 4.4. Expérimentations

Afin d'évaluer les performances des méthodes que nous avons conçues et implémentées, nous testons dans cette section le temps d'exécution de chaque méthode en variant les paramètres et la taille des jeux de données.

### 4.4.1. Benchmarks

Pour discuter la performance des méthodes implémentées, il est nécessaire d'appliquer des tests sur des données réalistes. Nous avons donc effectué nos expérimentations sur des séquences biologiques réelles que nous avons téléchargées à partir de la banque de données NCBI.

Les informations de ces fichiers FASTA sont présentées dans le tableau ci-dessous :

Nom du fichier	Nombre de séquences d'ADN	Nombre de lignes	Taille du fichier	Taille moyenne des séquences
Influenza_H1N1	28	794	51 ko	1621 bases
Orchid	94	1196	76 ko	718 bases
Hantavirus	88	3060	185 ko	1997 bases
HIV	200	6592	395 ko	1893 bases
Human_Viruses	2600	1932841	115009 ko	44513 bases

**Tableau 4.3:** Description de fichiers utilisés.

#### 4.4.2. Discussion

Notre étude expérimentale sera sur les trois méthodes de comparaison d'ADN, la comparaison par alignement avec l'algorithme de Needleman-Wunsch, la comparaison avec k-mers via l'indexation par dictionnaires et la comparaison avec k-mers en utilisant l'indexation par filtres de Bloom.

Dans un premier temps, nous procéderons à une étude comparative entre l'alignement par l'algorithme de Needleman-Wunsch et nos deux méthodes basées sur les k-mers. Ceci est dans l'objectif de répondre à la question suivante : est-il vraiment plus avantageux d'utiliser des méthodes de comparaison basées sur les k-mers plutôt que des algorithmes d'alignement classiques basées sur la programmation dynamique ?

Nous ferons également une étude comparative entre les performances des deux méthodes basées sur les k-mers, en fonction du paramètre  $k$  de 2 à 8 et de la taille des jeux de données.

K	Influenza_ H1N1				Orchid			Hantavirus			HIV			Human_Viruses		
	Méthode	k-mers indexés par dictionnaires	k-mers indexés par filtres de Bloom	Needleman-Wunsch	k-mers indexés par dictionnaires	k-mers indexés par filtres de Bloom	Needleman-Wunsch	k-mers indexés par dictionnaires	k-mers indexés par filtres de Bloom	Needleman-Wunsch	k-mers indexés par dictionnaires	k-mers indexés par filtres de Bloom	Needleman-Wunsch	k-mers indexés par dictionnaires	k-mers indexés par filtres de Bloom	Needleman-Wunsch
2		0,1199	1,5695	290,3570068			1287,818618			5436,570159			28995,53729			
3		0,2148	4,6949													
4		0,6710	14,770													
5		1,6777	41,054													
6		6,5581	72,626	290,3570068			1287,818618			5436,570159			28995,53729			
7		22,974	105,85													
8		82,889	35751													
		19425	35751													
		167,77	167,77	290,3570068			1287,818618			5436,570159			28995,53729			
		35751	35751													
		1011,6	1011,6													
		22556	22556													
		497,09	497,09	290,3570068			1287,818618			5436,570159			28995,53729			
		89633	89633													
		1151,3	1151,3													
		52663	52663													
		750,56	750,56	290,3570068			1287,818618			5436,570159			28995,53729			
		04551	04551													
		404,00	404,00													
		81074	81074													
		5062,5	5062,5	290,3570068			1287,818618			5436,570159			28995,53729			
		72934	72934													
		2250,5	2250,5													
		87365	87365													
		1510,7	1510,7	290,3570068			1287,818618			5436,570159			28995,53729			
		19934	19934													
		1036,0	1036,0													
		93559	93559													
		607,59	607,59	290,3570068			1287,818618			5436,570159			28995,53729			
		47676	47676													
		270,54	270,54													
		44031	44031													
		95823,92617	95823,92617	290,3570068			1287,818618			5436,570159			28995,53729			
		36497,38823	36497,38823													
		11277,9871	11277,9871													
		15805,14038	15805,14038													
		4603,5	4603,5	290,3570068			1287,818618			5436,570159			28995,53729			
		63132	63132													
		2257,2	2257,2													
		67073	67073													
		1331,4	1331,4	290,3570068			1287,818618			5436,570159			28995,53729			
		2744	2744													
		2763,6	2763,6													
		78642	78642													

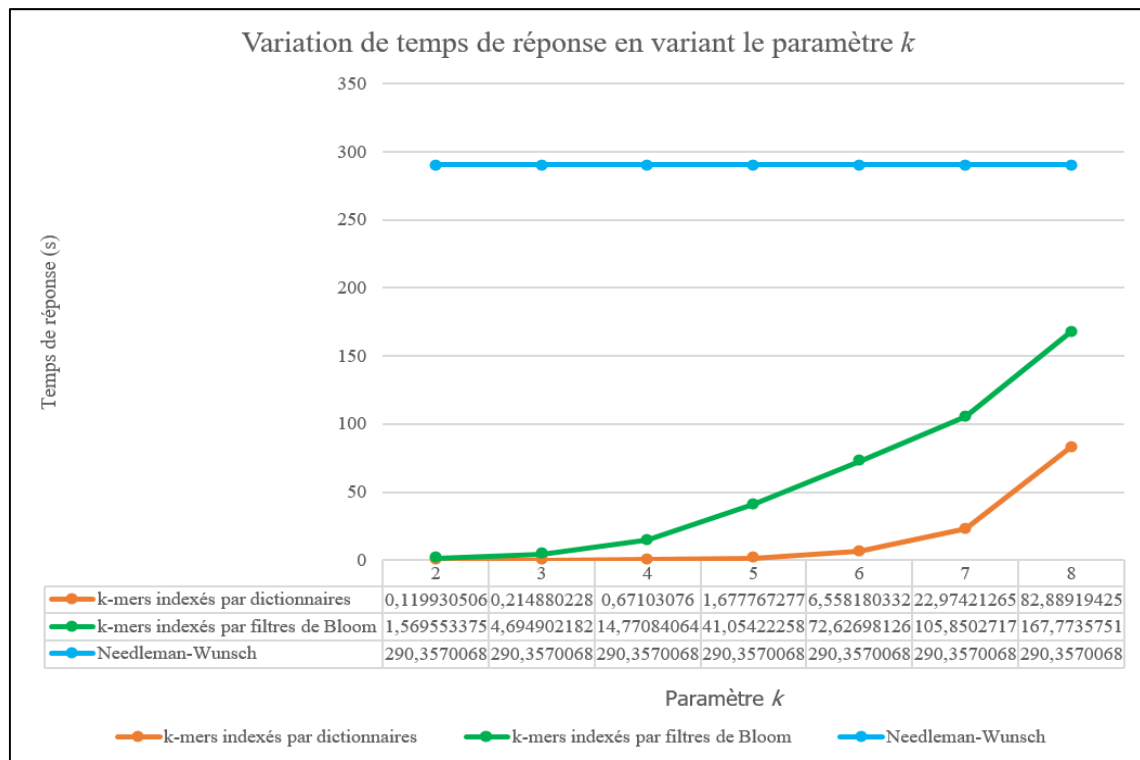
Tableau 4.4: Performances des méthodes de comparaison en fonction du paramètre k et de la taille des jeux de données.



Le tableau ci-dessus (**Tableau 4.2**) résume les temps d'exécution mesurés sur les 5 fichiers de tests obtenus avec les trois méthodes. Pour chacun de ces jeux de données, nous avons fait varier la valeur de  $k$  pour les deux méthodes basées sur les k-mers.

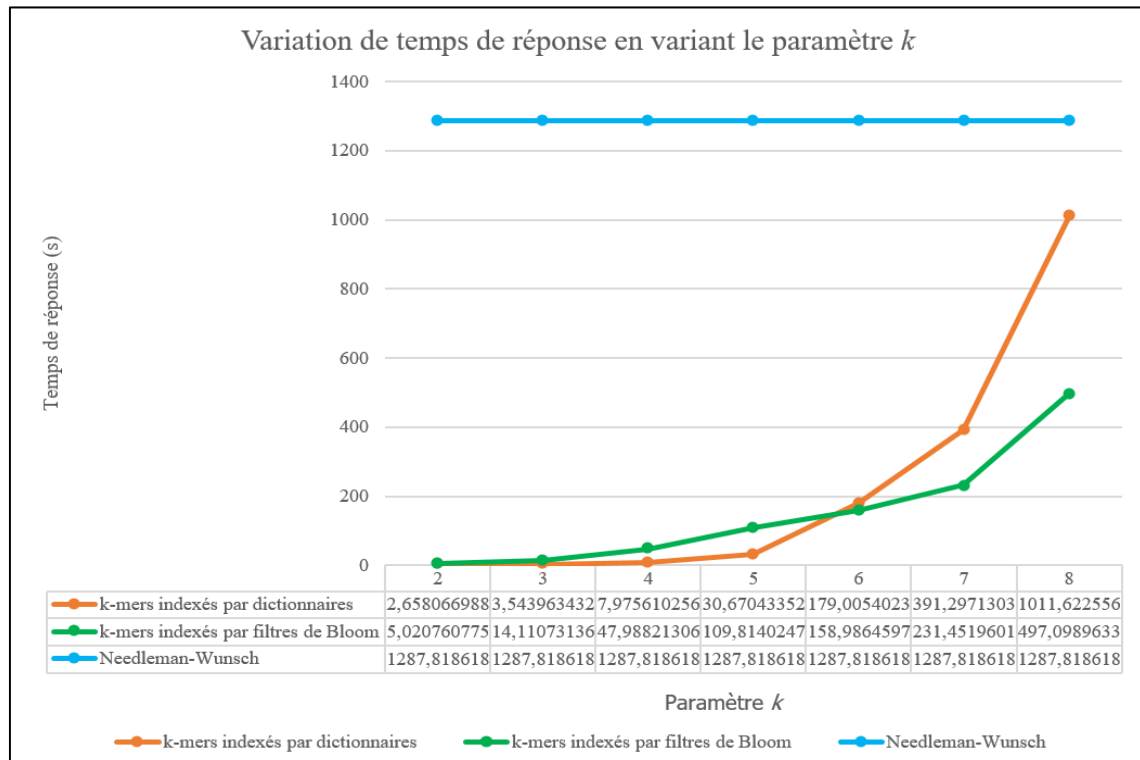
Afin d'obtenir une représentation plus claire, nous avons opté pour la réalisation de graphes correspondants à chaque fichier, en utilisant les trois méthodes. L'axe des ordonnées représente les temps d'exécution, tandis que l'axe des abscisses représente les valeurs de  $k$  (le paramètre  $k$  ne va pas influencer la méthode basée sur l'algorithme de Needleman-Wunsch) ; les couleurs correspondent aux méthodes de comparaison.

- **Fichier Influenza\_H1N1 :**



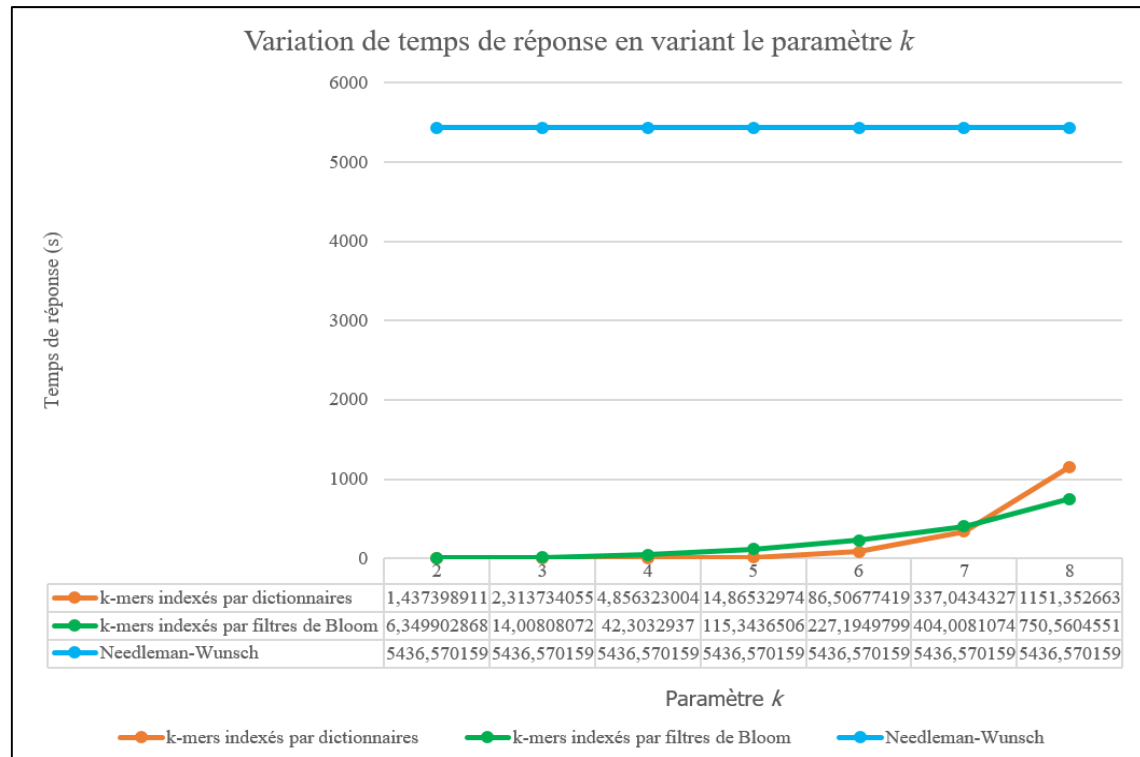
**Figure 4.13:** Variation du temps de réponse selon le paramètre  $k$  pour le fichier d'Influenza\_H1N1 avec les 3 méthodes de comparaison.

- **Fichier Orchid :**



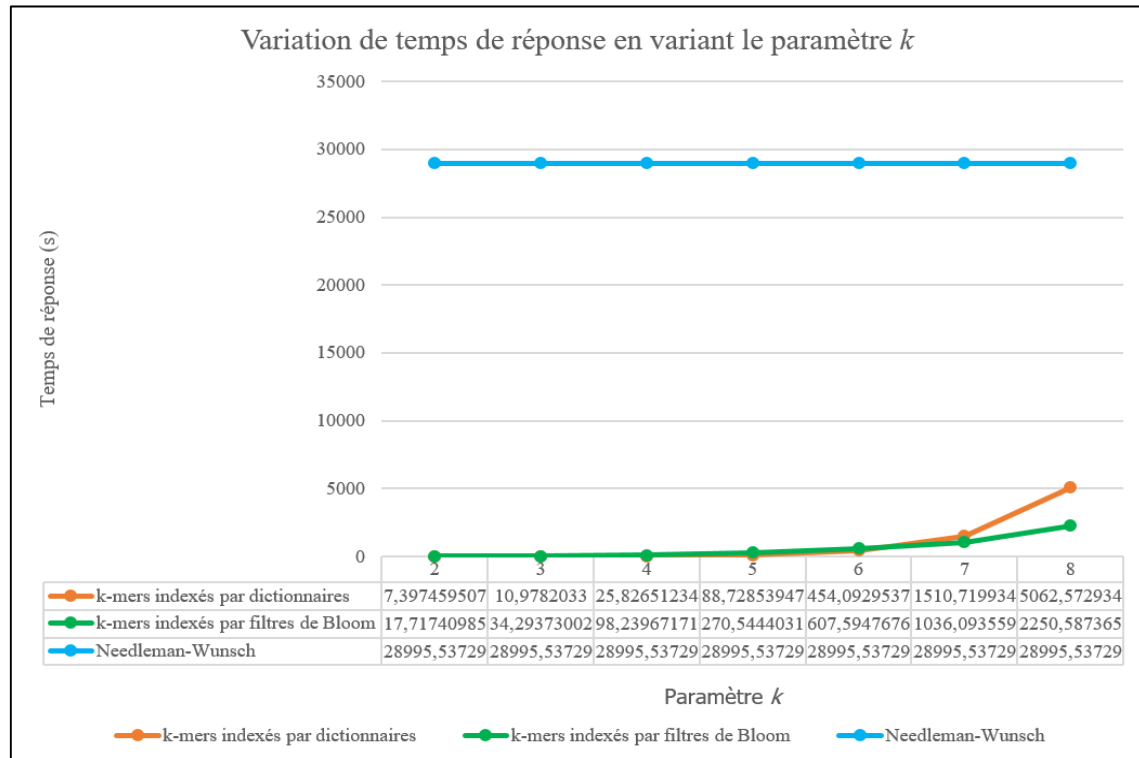
**Figure 4.14:** Variation du temps de réponse selon le paramètre  $k$  pour le fichier d'Orchid avec les 3 méthodes de comparaison.

- **Fichier Hantavirus :**



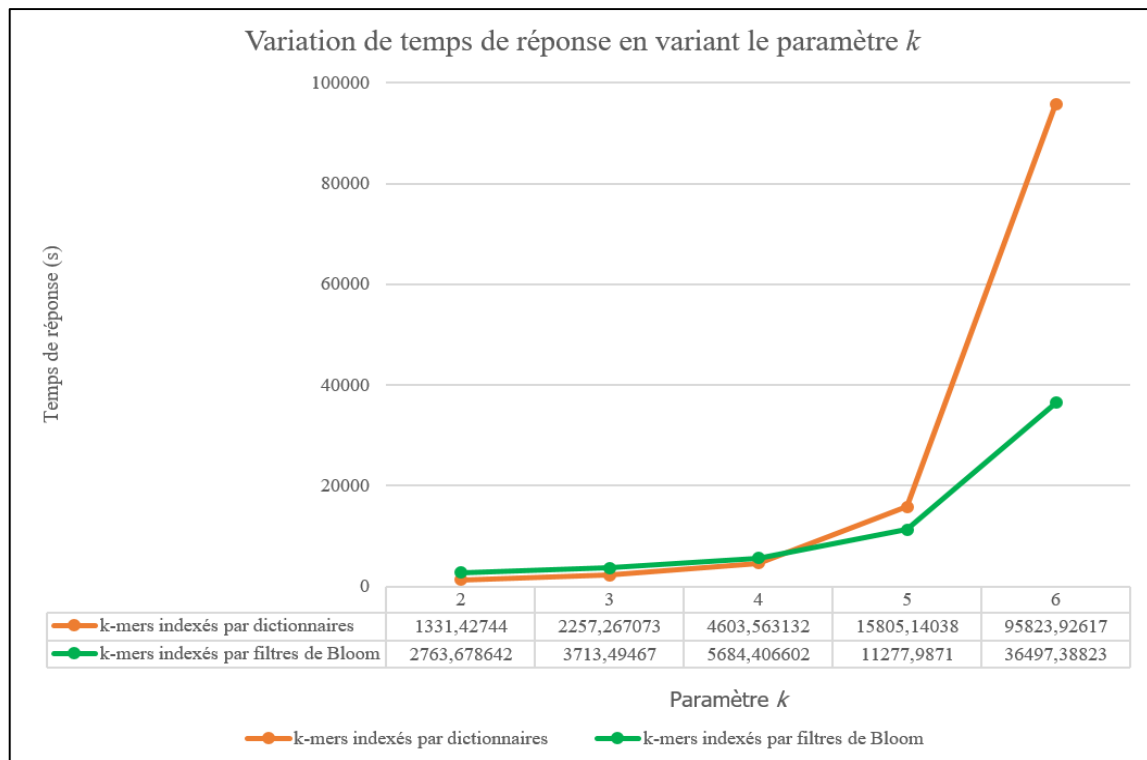
**Figure 4.15:** Variation du temps de réponse selon le paramètre  $k$  pour le fichier d'Hantavirus avec les 3 méthodes de comparaison.

- **Fichier HIV :**



**Figure 4.16:** Variation du temps de réponse selon le paramètre  $k$  pour le fichier d'HIV avec les 3 méthodes de comparaison.

- **Fichier Human\_Viruses :**



**Figure 4.17:** Variation du temps de réponse selon le paramètre  $k$  pour le fichier d'Human\_Viruses avec les 2 méthodes de comparaison

Comme on peut le constater dans les graphes précédents des figures **4.13**, **4.14**, **4.15**, **4.16**, **4.17**, quelle que soit la taille des k-mers, les performances de comparaison, en termes de temps, sont considérablement améliorées lors de l'utilisation des méthodes basées sur les k-mers. Nous remarquons également que la complexité des algorithmes dépend principalement de la taille des jeux de données, puisque les algorithmes se comportent quasiment de la même façon pour les cinq fichiers.

Nous pouvons donc conclure, à partir des résultats obtenus, que les nouvelles méthodes basées sur les k-mers apportent une grande amélioration, en terme de complexité temporelle, par rapport aux méthodes classiques basées sur l'alignement.

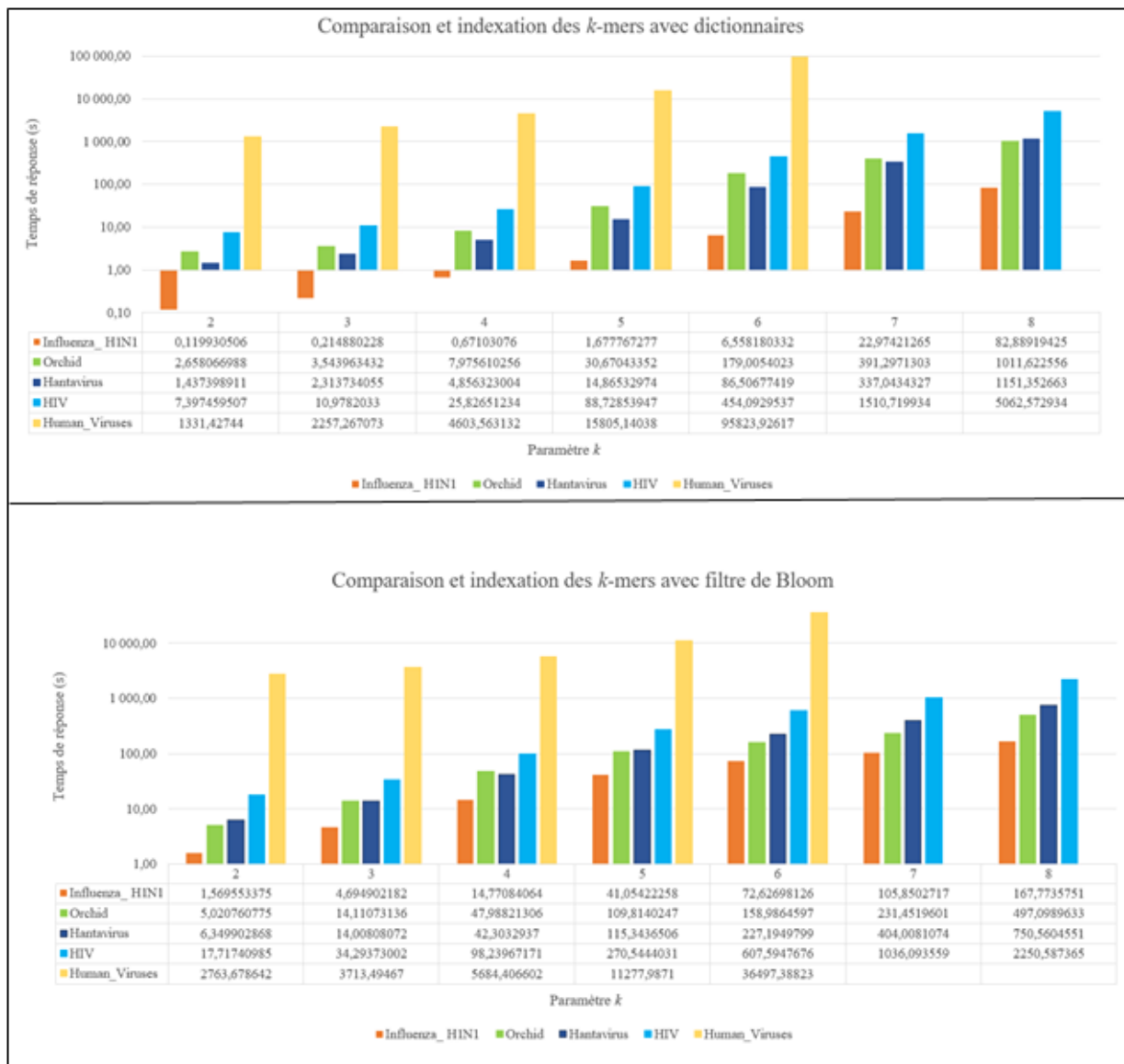
Pour ce qui est de la comparaison des deux approches d'indexation utilisées dans la comparaison à base de k-mers, l'indexation par dictionnaire et l'indexation par filtre de Bloom, nous allons toujours estimer la performance en se basant sur le temps d'exécution selon la taille des k-mers et la taille des fichiers.

Pour le fichier nommé « Influenza\_H1N1 » (voir la figure **4.13**) contenant le jeu de données le plus petit, les résultats de l'indexation par dictionnaires sont bien meilleurs que les résultats obtenus en utilisant l'indexation par filtres de Bloom pour toutes les valeurs de  $k \in [2; 8]$ .

Cependant, lorsque nous augmentons la taille du fichier (voir la figure **4.14**), les courbes montrent que l'avantage de l'indexation par dictionnaires par rapport à l'indexation par filtres de Bloom n'est visible qu'à partir de  $k = 4$ . Tandis qu'à partir de  $k = 6$  l'indexation par filtre de Bloom s'avère plus efficace.

Nous pouvons donc déduire que plus la taille augmente plus il est approprié d'utiliser les filtres de Bloom comme structure d'indexation.

L'influence de taille des k-mers et des fichiers est plus remarquable avec des jeux de données plus volumineux, car nous constatons dans la figure (**4.15**) et la figure (**4.16**) que la courbe représentant l'indexation par filtre de Bloom décroît dès que nous faisons augmenter la valeur de  $k$  à partir de  $k = 6$ . Ceci se confirme très clairement avec le fichier le plus volumineux qu'a pu supporter notre machine, le fichier « Human\_Viruses », où la valeur de  $k$  à partir de laquelle l'indexation par filtre de Bloom est plus efficace s'est diminué jusqu'à  $k \approx 4$ .



**Figure 4.18:** Impact de la taille des  $k$ -mers et des jeux de données sur les performances.

Concernant l'impact de  $k$  sur les performances de chacune des méthodes sur nos jeux de données. La figure 4.18 montre que le temps d'exécution augmente de manière proportionnelle avec  $k$  et avec la taille des fichiers.

## 4.5. Conclusion

Nous avons présenté dans ce chapitre l'implémentation des méthodes que nous avons conçues ainsi que la description de notre application.

Les résultats des tests que nous avons effectués sur des jeux de données de tailles différentes nous ont permis de comparer les performances des méthodes de comparaison sur lesquelles se repose notre étude.

Nous pouvons conclure des expérimentations réalisées que les méthodes de comparaison basées sur les k-mers apportent une grande amélioration des performances et permettent par ce fait, le traitement des bases de données biologiques, qui ne cessent de s'étendre, dans des durées beaucoup plus optimales.

Ceci nous mène aussi à constater le grand intérêt des structures d'index, grâce à nos expérimentations nous avons prouvé l'efficacité des filtres de Bloom pour les bases de données volumineuses.

Cependant, le traitement des fichiers de grandes tailles nécessite un espace de mémoire plus large. Les résultats obtenus pourraient se généraliser avec plus de certitude si nous pouvions effectuer ces expérimentations sur des bases de données plus larges, mais nos machines ne peuvent les supporter avec leur actuelles performances.

## Conclusion générale et perspectives

L'ADN représente le support de l'hérédité de l'information génétique. De ce fait, son analyse constitue l'une des tâches les plus importantes dans le domaine de la bio-informatique.

Dans ce mémoire, nous nous sommes intéressés à un problème fondamental de la bio-informatique, à savoir la comparaison de séquences d'ADN.

Après avoir présenté les concepts biologiques nécessaires à l'analyse et à l'étude de notre problème, nous avons trouvé que, d'un point de vue informatique, on peut considérer une séquence génétique comme un mot fini formé par un alphabet de quatre caractères : {A, C, G, T}. Cela signifie que son traitement nécessite des algorithmes spécialisés appartenant au domaine de l'algorithmique du texte.

Notre problématique repose essentiellement sur la détermination de la similarité et le calcul de distance entre deux séquences d'ADN. Les méthodes que nous avons exposées font toutes partie du domaine des algorithmes de recherche de motif, en prenant en compte ses deux catégories, à savoir la recherche exacte et la recherche approchée de motif.

Il existe deux grandes familles d'approches de comparaison d'ADN, la comparaison avec alignement et la comparaison sans alignement. Notre intérêt s'est porté sur la comparaison sans alignement qui permet d'être appliquées à des ensembles de données beaucoup plus volumineux que les méthodes classiques basées sur l'alignement. La plupart de ces méthodes sont basées sur des statistiques de mots ou des comparaisons de mots tel que les *k-mers* (mots de taille *k*) qui sont bien établies en bio-informatique dans la comparaison de séquences du génome.

Aussi, le grand intérêt des techniques de comparaison sans alignement est qu'elles peuvent être couplées à une étape d'indexation qui va réduire le nombre quadratique d'opérations à effectuer. Dans le contexte des séquences d'ADN, indexer les *k-mers* d'une manière structurée est une façon de minimiser l'usage de la mémoire.

Deux approches de comparaison ont été étudiées et présentées, basées sur la fréquence des *k-mers*, dont chacune a été réalisée avec une structure d'index différente. La première, le jeu de données a été indexé en utilisant les dictionnaires, pour construire ensuite le vecteur de *k-mers* à partir des fréquences stockées dans la structure d'index. Tandis que dans la seconde approche, nous avons utilisé les filtres de Bloom comme structure d'index, où nous avons appliqué l'indexation au fur et à mesure de la transformation en vecteur.

Ainsi, afin de donner plus de crédibilité à notre travail, nous avons implémenté la méthode de Needleman-Wunsch qui est basée sur l'alignement global classique (comparaison avec alignement). Cette implémentation nous a servi à vérifier l'apport des deux méthodes que nous avons proposées et qui appartiennent à la deuxième famille d'approche de comparaison sans alignement basées sur les *k-mers*. Le résultat de ces implémentations est une matrice de scores contenant les taux de similarité entre les séquences d'ADN comparées.

Afin d'illustrer l'usage de la comparaison d'ADN en pratique, nous avons utilisé les résultats obtenus dans un domaine d'application biologique, à savoir : la phylogénie. À partir de la matrice de scores obtenue, nous avons construit un arbre phylogénétique qui met en exergue l'évolution des séquences génétiques comparées. Cet arbre peut être utilisé pour résoudre divers problèmes ayant un rapport avec les liens de parenté entre les différentes familles de gènes.

À cet effet, nous nous sommes servis des connaissances en programmation acquises durant notre cursus universitaire, pour mettre en œuvre les méthodes développées et réaliser une application web pouvant être utilisée par les biologistes.

Pour tester les performances de nos approches, nous avons effectué des expérimentations sur un ensemble de séquences issues de 5 fichiers de données biologiques réelles, dont chacun se caractérise par une taille et un nombre de séquences différents.

Les résultats de nos expérimentations ont démontré que les méthodes de comparaison basées sur les k-mers assurent une grande amélioration des performances en terme de temps de calcul. Ceci est principalement dû à la possibilité de l'indexation durant la phase du calcul de la distance entre deux séquences. D'autre part, nous avons constaté que le choix de la structure d'index adéquate aux données dépend fortement de la taille du jeu de données à étudier.

Globalement, nous pensons que les résultats obtenus de nos expérimentations sont à la fois satisfaisants et prometteurs, surtout pour les bases de données biologiques volumineuses.

Comme perspectives, nous jugeons qu'il serait intéressant d'effectuer la comparaison de séquences de protéine et d'ARN, en gardant le même principe que nous avons implémenté pour comparer les séquences d'ADN.

Nous proposons également d'étendre les résultats que nous avons obtenus en utilisant un alignement multiple.

Il serait aussi bénéfique que l'introduction de séquences à comparer se fasse également en ligne (directement à partir des banques de données biologiques), sans que l'on soit obligé de les télécharger au préalable.



## Références bibliographiques

### A

- Aho, A. V., & Corasick, M. J. (1975). Efficient string matching. *Communications of the ACM*, 18(6), 333–340. Consulté à l'adresse <https://doi.org/10.1145/360825.360855>
- Altschul, S. F., Gish, W., Miller, W., Myers, E. W., & Lipman, D. J. (1990). Basic local alignment search tool. *Journal of molecular biology*, 215(3), 403-410.
- Aun, E., Brauer, A., Kisand, V., Tenson, T., & Remm, M. (2018). A k-mer-based method for the identification of phenotype-associated genomic biomarkers and predicting phenotypes of sequenced bacteria. *PLOS Computational Biology*, 14(10), e1006434.

### B

- Bernard, N., & Leprévost, F. (2012). Hardened Bloom Filters, with an Application to Unobservability. *Annales UMCS, Informatica*, 12(4).
- Bolser, D. M., Chibon, P.-Y., Palopoli, N., Gong, S., Jacob, D., Angel, V. D. D., Swan, D., Bassi, S., Gonzalez, V., Suravajhala, P., Hwang, S., Romano, P., Edwards, R., Bishop, B., Eargle, J., Shtatland, T., Provart, N. J., Clements, D., Renfro, D. P., ... Bhak, J. (2011). MetaBase-the wiki-database of biological databases. *Nucleic Acids Research*, 40(D1), D1250-D1254.
- Brillouin, L. (1988–1959). *La science et la théorie de l'information* (Vol. 1). Sceaux.
- Brochier-Armanet, C. (2018). Bioinformatique : alignement de séquences. Université de Claude Bernard, Lyon 1 Laboratoire de Biométrie et Biologie évolutive (UMR 5558), 26-29. Consulté à l'adresse <https://fdocuments.fr/document/bioinformatique-alignement-de-squences-alignement-de-squences-cline-brochier-armanet.html>
- Burrows, M., & Wheeler, D. J. (1994). A block sorting lossless data compression algorithm. *Technical Report*, p. 124.

### C

- Chegrane, I. (2016). *La recherche approchée de motifs : théorie et applications*. PhD thesis. Université des Sciences et de la Technologie Houari Boumédiène, p. 2. Consulté à l'adresse <https://arxiv.org/pdf/1701.08688.pdf>

- Christensen, K., Roginsky, A., & Jimeno, M. (2010). A new analysis of the false positive rate of a Bloom filter. *Information Processing Letters*, 110(21), 944-949.
- Claverie, J., & Notredame, C. (2006). *Bioinformatics For Dummies* (2nd ed.). For Dummies.
- Clark, R.G., Rae, M.S. (1984). A class of wasserstein metrics for probability distributions. *The Michigan Mathematical Journal*, 31(2), 231-240.
- Cleuziou, G. (2004, December). Une méthode de classification non-supervisée pour l'apprentissage de règles et la recherche d'information (Thèse de doctorat). Université d'Orléans.
- Coissac, E. (2005). La fluidité des génomes. Université de Pierre et Marie Curie, Paris, p. 40.  
Consulté à l'adresse [https://www.researchgate.net/publication/281989102\\_Genome\\_plasticity](https://www.researchgate.net/publication/281989102_Genome_plasticity)
- Courrèges, A. (2014). Les notions-clés de la génétique médicale. L'Agence de Biomédecine Sur La Génétique Médicale. Consulté à l'adresse <https://genetique-medicale.fr/la-genetique-l-essentiel/les-notions-cles-de-la-genetique/article/les-notions-cles-de-la-genetique-medicale>
- Crochemore, M., Hancart, C., & Lecroq, T. (2001). *Algorithmique du texte*, Editions Vuibert, Paris.

## D

- Dameron, O., & Farrant, G. (2014). La bioinformatique avec Biopython. *GNU/Linux Magazine*, 73(13), 84-97.
- Dardel, F., & Képès, F. (2002). *Bioinformatique: Génomique et post-génomique (ECOLE POLYTECHNIQUE)* (French Edition) (0 ed.). ECOLÉ POLYTECH.
- Delarue, M., & Furelaud, G. (2004, May 1). Le séquençage d'un ADN. Planet-Vie. Consulté à l'adresse <https://planet-vie.ens.fr/thematiques/cellules-et-molecules/le-sequencage-d-un-adn>

## E

- El-Mabrouk, N. (2013). Recherche de motifs, Université de Montréal, p. 24. Consulté à l'adresse <http://www.iro.umontreal.ca/~mabrouk/IFT3295/Recherche-exacte.pdf>
- Erki, A., Age, B., Maido, R., Tanel, T., & Veljo, K. (2018). A k-mer-based method for the identification of phenotype-associated genomic biomarkers and predicting phenotypes of sequenced bacteria. Consulté à l'adresse <https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1006434>

## F

- Federal Information Processing. (1993). Secure Hash Standard. National Institute of Standards and Technology, Standards Publication FIPS PUB 180.
- Ferragina, P., & Manzini, G. (2000). "Opportunistic Data Structures with Applications". Proceedings of the 41st Annual Symposium on Foundations of Computer Science, p. 390.
- Ferragina, P., Manzini, G., Mäkinen, V., & Navarro, G. (2004). An Alphabet-Friendly FM-Index. String Processing and Information Retrieval, 150-160.
- Fondrat, C. (1997). Les banques de séquences biologiques. Cours de Bioinformatique.

## G

- Gueddah, H., Aoureg, S. L., & Yousfi, A. (2014). Adaptation de la distance de Levenshtein pour la correction orthographique contextuelle, p. 2. Consulté à l'adresse [https://www.researchgate.net/publication/271906755\\_Adaptation\\_de\\_la\\_distance\\_de\\_Levenshtein\\_Pour\\_la\\_Correction\\_Orthographique\\_Contextuelle](https://www.researchgate.net/publication/271906755_Adaptation_de_la_distance_de_Levenshtein_Pour_la_Correction_Orthographique_Contextuelle)

## H

- Hamel, S. (2012). Arbre des suffixes. Université de Montréal. Consulté à l'adresse <http://www.iro.umontreal.ca/~hamelsyl/ArbresSuffixes2012.pdf>
- Haubold, B., Klötzl, F., & Pfaffelhuber, P. (2015). andi: fast and accurate estimation of evolutionary distances between closely related genomes. Bioinformatics (Oxford, England), 31(8), 1169-1175.
- Höhl, M., Rigoutsos, I., & Ragan, M. A. (2007). Pattern-based phylogenetic distance estimation and tree reconstruction. Evolutionary bioinformatics online, 2, 359-375.

## I

- Iovleff, S. (2018). Algorithme de Knuth-Morris-Pratt. Consulté à l'adresse [https://nanopdf.com/download/algorithme-de-knuth-morris-pratt-departement-informatique-de-liut\\_pdf](https://nanopdf.com/download/algorithme-de-knuth-morris-pratt-departement-informatique-de-liut_pdf)

## K

- Kent W. J. (2002). BLAT--the BLAST-like alignment tool. Genome research, 12(4), 656-664.

## L

- Lamoril, J., Ameziane, N., Deybach, J.-C., Bouizegarène, P., & Bogard, M. (2008). Les techniques de séquençage de l'ADN : une révolution en marche. Première partie. Immuno-Analyse & Biologie Spécialisée, 23(5), 260-279.
- Lempel, A., & Ziv, J. (1976). On the complexity of finite sequences, IEEE Transactions on Information Theory, 22 (1) ,75-81.
- Levenshtein, V. I. (1966). Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. Soviet Physics Doklady Vol.10, 707-710.
- Lopez, P., Casane, D., & Philippe, H. (2002). Phylogénie et évolution moléculaires. Médecine/Sciences, 18(11), 1146-1154.
- Lultz, M., & Bailly, Y. (2005). Python. O'Reilly Editions.

## M

- Manber, U., & Myers, G. (1990). Suffix arrays: a new method for on-line string searches. First Annual ACM-SIAM Symposium on Discrete Algorithms, 319-327.
- Meurisse, D., & Mocq, F. (2018). Python, Raspberry Pi et Flask. Editions ENI.

## N

- Nicolas, (2012). Filtre de Bloom. <https://bioinfo-fr.net/filtre-de-bloom>

## O

- Omari, K. (2010). La réalisation d'une application de contrôle total des processus d'un ordinateur distant. Université pédagogique nationale (UPN). Consulté à l'adresse [https://www.memoireonline.com/05/12/5812/m\\_La-realisation-dune-application-de-contrle-total-des-processus-dun-ordinateur-distant4.html](https://www.memoireonline.com/05/12/5812/m_La-realisation-dune-application-de-contrle-total-des-processus-dun-ordinateur-distant4.html)

## P

- Parreaux, J. (2018). Leçon 907 : Algorithmique du texte. Exemples et applications, p. 5. Consulté à l'adresse [http://perso.eleves.ensrennes.fr/people/julie.parreaux/fichiers\\_agreg/info\\_lecons/907\\_AlgoTexte.pdf](http://perso.eleves.ensrennes.fr/people/julie.parreaux/fichiers_agreg/info_lecons/907_AlgoTexte.pdf)

- Patro, R., Duggal, G., Love, M. I., Irizarry, R. A., & Kingsford, C. (2017). Salmon provides fast and bias-aware quantification of transcript expression. *Nature methods*, 14(4), 417-419.
- Patro, R., Mount, S. M., & Kingsford, C. (2014). Sailfish enables alignment-free isoform quantification from RNA-seq reads using lightweight algorithms. *Nature biotechnology*, 32(5), 462-464.
- Pearson W. R. (2001). Training for bioinformatics and computational biology. *Bioinformatics* (Oxford, England), 17(9), 761-762.
- Pearson, W. R., & Lipman, D. J. (1988). Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences of the United States of America*, 85(8), 2444-2448.
- Peterlongo, P., Lecompte, L., Limasset, A., Bittner, L., & Marchet, C. (2016). A resource-frugal probabilistic dictionary and applications in (meta)genomics. *Discrete Applied Mathematics*, 274, 92-102.

## R

- Robin, D. (2019). ADN - Le code source et l'origine de la vie (French Edition) (1<sup>ère</sup> ed.). Le Mercure Dauphinois.

## S

- Saldaña, F. (2018). The Boyer-Moore-Horspool Algorithm. Consulté à l'adresse <https://medium.com/@fsaldana/the-boyer-moore-horspool-algorithm-51b785afde67>
- Saporta, G. (2011). Probabilités, analyse des données et statistique. TECHNIP.
- Schutz, S. (2016). Le filtre de bloom. Consulté à l'adresse <https://dridk.me/bloom-filter.html>.
- Sèverine. B. (2016). Recherche exacte de motifs "Algorithme de Boyer-Moore".
- Shannon C. E. (1948). The mathematical theory of communication. *The Bell System Technical Journal*, Vol. 27, pp. 379-423, 623-656.
- Smith F., & Waterman, M. S. (1981). Identification of Common Molecular Subsequences. *Journal of Molecular Biology*, vol. 147: 443-453.
- Starobinski, D., Trachtenberg, A., & Agarwal, S. (2003). Efficient PDA synchronization. *IEEE Transactions on Mobile Computing*, 2(1), 40-51.  
<https://doi.org/10.1109/tmc.2003.1195150>

## **T**

Tatiana, R. (2018). Compression et indexation de séquences annotées, 37-38. Consulté à l'adresse <https://tel.archives-ouvertes.fr/tel-01758361/document>

Thompson, J. D., Higgins, D. G., & Gibson, T. J. (1994). CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic acids research*, 22(22), 4673-4680.

## **V**

Vinga, S., & Almeida, J. (2003). Alignment-free sequence comparison-a review. *Bioinformatics* (Oxford, England), 19(4), 513-523.

Vroland, C. (2016). Algorithmique pour la recherche de motifs approchée et application à la recherche de cibles de microARN. Bio-informatique [q-bio.QM]. Thèse de Doctorat de l'Université Lille 1 Sciences et Technologies. Consulté à l'adresse <https://tel.archives-ouvertes.fr/tel-01576433/document>

## **W**

Wei, D., Jiang, Q., Wei, Y., & Wang, S. (2012). A novel hierarchical clustering algorithm for gene sequences. *BMC bioinformatics*, 13, 174.

Wu, T.-J., Huang, Y.-H., & Li, L.-A. (2005). Optimal word sizes for dissimilarity measures and estimation of the degree of dissimilarity between DNA sequences. *Bioinformatics*, 21(22), 4125-4132.

## **Z**

Zielezinski, A., Vinga, S., Almeida, J., & Karlowski, W. M. (2017). Alignment-free sequence comparison: benefits, applications, and tools. *Genome biology*, 18(1), 186.