Search engine for genomic sequencing data

Téo Lemane

Institut National de Recherche en Informatique et en Automatique

Institut de Recherche en Informatique et Systèmes Aléatoires

Equipe GenScale

(Scalable, Optimized and Parallel Algorithms for Genomics)

Search engine for genomic sequencing data

Rapport de Stage

Master 2 Bio-informatique

2018-2019

Auteur :

Téo LEMANE

Encadrant :

Pierre PETERLONGO

Master bioinformatique

UNIVERSITÉ DE RENNES 1

ENGAGEMENT DE NON PLAGIAT

Je, soussigné(e) ..Téa..Lemasle.....
étudiant(e) en..M2..Bioinformatique.....
déclare être pleinement informé que le plagiat de documents ou d'une partie de document publiés sur toute forme de support, y comprit l'internet, constitue une violation des droits d'auteur ainsi qu'une fraude caractérisée.

En conséquence, je m'engage à citer toutes les sources que j'ai utilisées pour la rédaction de ce document.

Date : 14/06/2019

Signature :

Document à compléter de manière manuscrite et à insérer obligatoirement en première page du rapport de stage.

Remerciements

Je tiens tout d'abord à remercier Pierre Peterlongo pour m'avoir accordé ce stage, pour sa disponibilité et ses précieux conseils, ainsi que pour les multiples relectures de ce rapport.

Aussi, j'adresse mes remerciements à l'équipe Genscale, et plus généralement à l'ensemble de Symbiose pour leur accueil et le cadre de travail très agréable.

Enfin, je remercie mes collègues stagiaires pour leur bonne humeur au quotidien qui a pleinement contribué au bon déroulement de mon stage.

Résumé

Depuis plusieurs années, l'évolution des techniques de séquençage associée à une importante réduction des coûts à conduit à une utilisation massive de ces technologies. Ceci se traduit par une accumulation de données brutes, uniquement utilisées dans le cadre de projets isolés. Ces données n'étant pas indexées, elles ne sont pas exploitables d'un point de vue global. Des méthodes d'indexation répondent à ce problème en permettant de cibler un ensemble de jeu de données à partir de la requête d'une ou plusieurs séquences. Cependant, bien que très optimisée, ces méthodes ne permettent pas encore le passage à l'échelle nécessaire à l'indexation des bases de données actuelles qui sont constituées de plusieurs pétaoctets de données.

Dans ce cadre, nous proposons une approche d'indexation avec l'idée d'améliorer les performances et d'apporter de nouvelles fonctionnalités. Celle-ci est basée sur deux outils existants dont les implémentations ont été modifiées pour répondre au problème. Simka, un outil de comparaison de métagénomes, utilisé ici comme un compteur de k-mers particulier. Et HowDeSBT, un outil d'indexation utilisant des filtres de Bloom.

La méthode présentée apporte de nouvelles fonctionnalités avec notamment un traitement spécifique des k-mers rares (uniques dans un jeu de données), qui sont simplement ignorés par les méthodes d'indexations actuelles. Ces nouvelles possibilités n'ont que très peu d'impact sur les performances qui sont améliorées en terme de mémoire et de temps de calcul. Néanmoins, la réduction des temps de calcul est loin d'être suffisante pour espérer une indexation global des bases de données. Pour le permettre, les temps de calcul proposés par les méthodes actuelles devront être réduit d'au minimum un facteur 100.
Mots clés : Indexation, structure de données, données NGS, passage à l'échelle, k-mers rares.

Abstract

For several years, the evolution of sequencing techniques combined with a significant reduction in costs has led to a massive use of these technologies. This results in an accumulation of raw data, which is only used in isolated projects. As these data are not indexed, they cannot be used from a global point of view. Indexing methods address this problem by allowing a set of data sets to be targeted from the request of one or more sequences. However, although very optimized, these methods do not yet allow the necessary scaling to index the current databases, which are composed of several petabytes of data.

In this context, we propose an approach with the idea of improving performance and of bringing new features. This is based on two existing tools whose implementations have been modified to address the problem. Simka, a metagenome comparison tool, used here as a particular k-mer counter. And HowDeSBT, an indexing tool using Bloom filters.

The method presented brings new functionalities with in particular a specific treatment of rare k-mers (uniques in a data set), which are simply ignored by current indexing methods. These new possibilities have very little impact on performance, which is improved in terms of memory and computing time. Nevertheless, the reduction in computing time is far from sufficient to hope for a global indexing of databases. To allow this, the computation times proposed by current methods will have to be reduced by at least a factor of 100.
Keywords : Indexing, data structure, NGS data, scaling, rare k-mers.

Sommaire

I Introduction

I.1 Contexte

Depuis des années l'évolution des techniques de séquençages apporte de nouvelles connaissances dans différents domaines liés à la biologie. Dans les années 90, la course au séquençage a débuté avec le projet génome humain[1] suivi par d'autres projets sur des espèces modèles[2]. L'objectif était l'assemblage de génomes d'un maximum d'espèces dans un but comparatif afin de révéler les variations génétiques qui sont une des sources d'informations majeures en biologie. Ces données ont apporté énormément de nouvelles connaissances avec un impact dans divers domaines. Aujourd'hui, les coûts de plus en plus bas du séquençage permettent l'utilisation du séquençage en routine dans de nombreux projets mais aussi au sein de projets spécifiques dédiés au séquençage massif. Avec cette utilisation massive[3], de nouveaux problèmes ont vu le jour. En effet, les données de séquençages s'accumulent plus vite que la loi de Moore[4] et la puissance de calcul seule ne suffit plus pour exploiter cette mine d'information.

A l'heure actuelle, les avancées matériels et logiciels permettent le traitement de données de séquençage au sein de projet à petite échelle. La majorité des données stockées dans les bases de données comme la Sequence Read Archive[5] (SRA) ont seulement été utilisée dans le cadre de projets isolés. Certaines de ces données proviennent de projet de séquençage "à la chaîne" et restent inutilisées. Ces bases de données ne peuvent pour le moment pas indexer les données brutes (non assemblées) dont elles disposent, faute d'outils suffisamment performant. Il est impossible de rechercher une expérience autrement que par son numéro d'accession ou en utilisant les métadonnées qui lui sont associées. La recherche d'une séquence particulière dans l'ensemble des expériences demanderait de nombreuses heures, jours ou mois de calcul en fonction de la base de données concernée. L'indexation de ces bases de données pourrait permettre un traitement plus global avec tous les avantages que cela confère. Il serait alors possible de requêter des millions de jeux de données avec, par exemple, un transcrit d'intérêt, un SNP[6] particulier ou encore un génome d'intérêt dans le cadre de jeux de données métagénomiques. Bien sûr, l'indexation ne se limite pas à de grands ensembles de données, et peut être utilisée sur de plus petits jeux de données comme une cohorte de patients ou une population d'individus. Dans ce cas, l'exploration est facilitée avec des temps de requêtes extrêmement rapides.

Une méthode d'indexation efficace peut jouer le rôle d'un filtre qui permettrait de cibler les expériences en liens avec une séquence avant d'utiliser des méthodes plus précises mais plus coûteuses dans le but d'obtenir des informations supplémentaires ou de réaliser des analyses sur ces expériences.

I.2 Objectifs

Depuis 2016, de nombreuses méthodes ont vu le jour pour répondre aux problèmes de passage à l'échelle inhérent à l'indexation de grandes quantités de données. À l'heure actuelle, ces méthodes ne permettent toujours pas l'étude de très gros jeux de données qui peuvent atteindre plusieurs dizaines voir centaines de teraoctets. Dans notre cas, la mission d'une méthode d'in-

dexation est simplement la localisation d'une séquence dans une énorme banque d'expériences de séquençages. Tout cela avec des temps de calcul raisonnables du point de vue de l'indexation en elle-même et de la requête.

Le premier objectif du stage est donc une analyse précise de l'état de l'art afin de comprendre précisément les méthodes et algorithmes mis en jeu. Cette étape est primordiale pour identifier les limitations et améliorations possibles des méthodes existantes.

Dans un second temps, nous avons envisagé les implémentations possibles avec l'idée de proposer un nouvel outil disposant de nouvelles fonctionnalités avec une réduction des temps de calcul.

II État de l'art

II.1 Compression de texte sans perte

Une première famille de méthodes consiste en la compression de texte sans perte. Un index construit à partir de ces méthodes est capable de répondre au problème de localisation d'une séquence dans un ensemble de données. Les premiers outils destinés à l'indexation de données de séquençage utilisent ce type d'algorithmes comme par exemple BEETL-fastq[7] ou pop-BWT[8]. Ces deux outils sont basés sur le FM-index[9] et la transformé de Burrows-Wheeler[10] qui permettent une compression sans perte des données avec la possibilité de déterminer le nombre d'occurrences d'un motif ainsi que la position de ces occurrences.

Ces méthodes, bien que très efficaces sur de petits jeux de données, ne permettent pas le passage à l'échelle et ne sont donc pas adaptées dans le cas présent. La totalité de l'information est conservée, c'est-à-dire que les séquences d'origines peuvent être entièrement retrouvées à partir de l'index. C'est une fonctionnalité, non recherchée ici, qui a un impact au niveau des taux de compression et des temps d'exécution en comparaison avec les méthodes avec perte qui sont plus adaptées au passage à l'échelle.

II.2 Méthodes basées sur l'utilisation de k-mers

Une expérience de séquençage peut être représentée par une collection de k-mers, c'est-à-dire l'ensemble des sous-séquences de longueur k contenues dans l'expérience. Cette approche est utilisée dans de nombreuses branches de la bio-informatique. L'approche k-mer est, par exemple, très efficace dans le cadre de l'assemblage avec l'utilisation de graphes de De Bruijn[11] pour l'assemblage *de novo*[12-14]. On la retrouve aussi dans l'alignement de séquences[15] ou encore la détection de variants[16].

Un ensemble d'expériences de séquençage peut donc être représenté par une structure associant chaque k-mer à une liste d'expériences. La question de la présence ou de l'absence d'une séquence dans une expérience est alors simplifiée et ne demande plus d'alignement. Il suffit simplement de compter les k-mers partagés par la séquence et les expériences, une approche beaucoup plus rapide que l'alignement dynamique[17].

Le nombre de k-mers contenus dans un ensemble de jeux de données peut vite atteindre plusieurs milliards. Il s'agit donc de stocker et d'indexer efficacement ces associations k-mers/expériences tout en permettant des requêtes rapides.

Plusieurs algorithmes et méthodes qui tentent de répondre à ce problème sont présentés dans la suite de cette partie.

II.2.1 Filtre de Bloom

Un grand nombre de méthodes utilisent comme structure de base le filtre de Bloom[18]. Un filtre de Bloom B est une structure de données permettant de représenter un ensemble d'éléments de manière efficace en terme d'espace et de temps. Cette structure supporte deux opérations qui sont l'insertion et la requête : $insert(B, x)$ et $contain(B, x)$. En revanche, le comptage n'est pas supporté par les filtres de Bloom, on ne peut pas connaître le nombre d'occurrences d'un élément.

C'est un tableau de bits $B[0, ..., m-1]$ associé à k fonctions de hachage avec $h_i : U \rightarrow \{0, ..., m-1\}$, où $1 \leq i \leq k$ et U est l'univers des objets qui peuvent être insérés dans le filtre. Une fonction de hachage est une fonction qui va, à partir d'une entrée, renvoyer une signature qui correspond ici à une position dans le filtre, c'est-à-dire un entier p tel que $0 \leq p \leq m-1$.

Pour insérer un élément x, le filtre règle $B[h_i(x)] \leftarrow 1$ pour $i = 1, ..., k$. Pour vérifier la présence d'un élément, on vérifie les valeurs des bits correspondant à chaque valeurs de hachage : $CONTAIN(B, x) = \bigwedge_{i=1}^{k} B[h_i(x)]$.

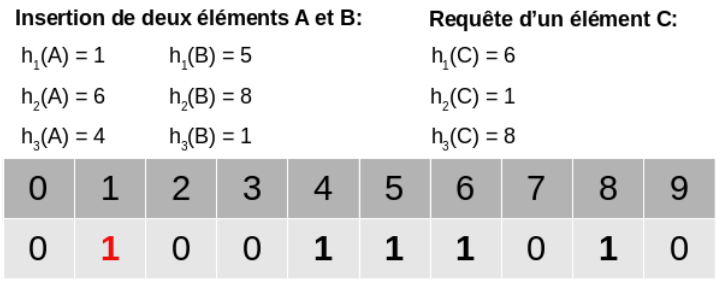

FIGURE 1 – Exemple d'insertion de deux éléments dans un filtre de Bloom de taille 10 avec 3 fonctions de hachage. Pour chaque élément, 3 positions sont retournées par les 3 fonctions de hachage et les bits correspondant à ces positions sont réglés à 1.
Collision entre $h_1(A)$ et $h_3(B)$. Les collisions peuvent aboutir à des faux positifs lors de la requête comme c'est le cas pour l'élément C. Les positions correspondantes aux valeurs de hachage de C sont à 1 alors que cet élément n'a pas été indexé.

Les filtres de Bloom font partie de la famille des structures de données probabilistes. Les fonctions de hachage n'étant pas parfaites, il est possible d'avoir plusieurs éléments avec la même signature (Figure 1). Cette particularité entraîne plusieurs inconvénients dont la possibilité d'avoir des faux positifs. Le taux de faux positifs d'un filtre de Bloom B de taille m après l'insertion de n éléments est d'environ $(1 - e^{-nk/m})^k$. On peut optimiser ce taux en choisissant $k = \frac{m}{n} ln2$ fonctions de hachages. Cela implique qu'un espace m suffisant doit être alloué dès le départ afin que l'insertion de nouveaux éléments est peu d'impact sur le taux de faux positifs.

La faible localité des données rend aussi difficile l'utilisation sur disque. Par exemple, un filtre de Bloom utilisé sur disque avec dix fonctions de hachage peut insérer environ 20 éléments par seconde contre plusieurs milliers lorsqu'il est utilisé en mémoire.

II.2.2 Indexation des k-mers sans compression de la redondance

Deux grands types d'approches distinguent les méthodes d'indexation. Cette partie concerne l'approche sans compression de la redondance. Ici, le contenu de chaque expérience de séquençage est indexé de manière indépendante. Ainsi les k-mers partagés entre les expériences sont indexés plusieurs fois. La seconde famille de méthodes correspond à une indexation avec gestion de la redondance de ces k-mers. Dans ce cas, ce n'est pas le contenu de chaque expérience qui est indexé mais l'origine de chaque k-mer. L'index contient alors la liste des expériences relatives à chacun des k-mers. Les méthodes associées à cette approche sont présentées à la section II.2.3.

II.2.2.a BItsliced Genomic Signature Index (BIGSI)

BIGSI[19] est une méthode récente qui utilise les filtres de Bloom comme structure de base. Dans cet index, chaque expérience est représentée par un filtre de Bloom. Tous ces filtres sont construits à partir des mêmes paramètres et peuvent donc être regroupés sous la forme d'une matrice de filtre de Bloom avec n lignes pour les n expériences. Ceci en fait une structure de données dynamique qui permet l'intégration de nouveaux jeux de données simplement par l'ajout de filtres de Bloom supplémentaires à la matrice. La représentation sous forme de matrice est un avantage pour les requêtes. Pour une requête donnée, l'élément en entrée est haché une seule fois par chaque fonction de hachage. Aussi, cela permet un traitement par colonne où l'intersection des colonnes correspondantes aux valeurs de hachages détermine les expériences qui contiennent probablement la requête (figure 2). L'intersection de deux vecteurs binaires peut être facilement calculée à l'aide de l'opération bit à bit *AND*.

Cela peut aussi être un problème lorsque les jeux de données à indexer sont hétérogènes, c'est-à-dire lorsque le nombre de k-mers est très différent selon les expériences. Avec une taille de filtre élevée, donc adaptée au jeu de données avec un grand nombre de k-mers, les filtres représentant les plus petits jeux de données sont clairsemés et utilisent de l'espace de stockage inutilement. Si on considère une taille de filtre adaptée aux plus petits jeux de données, les filtres indexant beaucoup de k-mers se retrouvent surchargés et donc très sensibles aux faux positifs.

Une amélioration de BIGSI a été présentée lors de la conférence DSB 2019. Il s'agit du Compact Bit-Sliced Signature Index[20] (COBS) . C'est une implémentation C++ de BIGSI qui répond à certaines limitations évoquées précédemment avec par exemple différentes classes de taille de filtre de Bloom possibles. COBS exploite aussi les nouvelles spécifications des SSD, rendant l'utilisation sur disque beaucoup plus efficace. Les temps de calcul et la taille de l'index sont nettement améliorés.

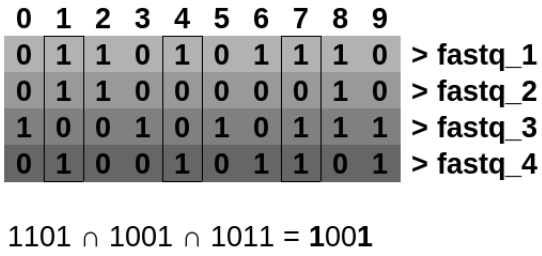

FIGURE 2 – Exemple de requête BIGSI. Ici dans un cas à 3 fonctions de hachage qui retournent les positions 1, 4 et 7 pour les trois k-mers qui constituent la requête. L'intersection de ces colonnes donne les expériences qui contiennent probablement la requête : fastq_1 et fastq_4.

II.2.3 Indexation des k-mers avec compression de la redondance

II.2.3.a Mantis

Mantis[21] est une méthode d'indexation construit autour de Squeakr[22], un compteur de k-mers basé sur des Counting Quotient Filter[23] (CQF) qui offrent plus de possibilités que les filtres de Bloom. Ce type de filtre supporte le comptage, contrairement au filtre de Bloom, et peut être utilisé en mode exact, sans possibilité de faux positifs. Ils permettent aussi d'autres opérations comme la suppression, le redimensionnement ou la fusion.

Mantis utilise des graphes de De Bruijn colorés[24]. Ceux-ci ont une structure identique aux graphes de De Bruijn classiques mais dans laquelle chaque noeud du graphe (k-mer) est associé à un label. Pour un ensemble de n jeux de données, il existe un ensemble C de i couleurs $c_1, ..., c_2$ où c_i correspond a un jeu i et l'ensemble des k-mers contenus dans i ont pour label c_i.

Pour représenter ce graphe, Mantis utilise deux structures de données. La première est un CQF qui permet d'associer chaque k-mer a un identifiant de couleur. La seconde associe chaque classe de couleur a un vecteur binaire avec un bit par expérience (figure 3). Ici l'utilisation de la fonction de comptage du CQF est détournée pour permettre le stockage des identifiants. Pour associer un k-mer X à un identifiant i, i copies de X sont insérées dans le filtre. Chaque identifiant i correspond à une combinaison précise d'expériences.

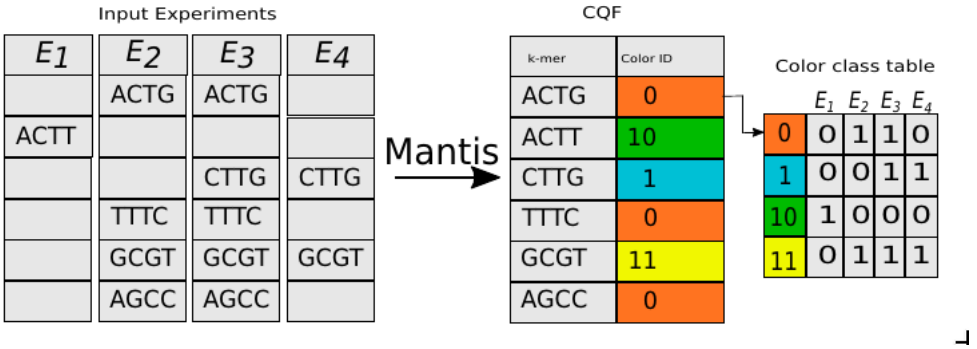

FIGURE 3 – Structure de l'index Mantis. D'après Pandey *et al.*
Le CQF associe chaque k-mer a un identifiant de couleur lui même associé à une combinaison d'expériences dans la table d'association.

Pour construire cet index, le contenu en k-mer de chaque expérience est d'abord déterminé par Squeakr avec en sortie un CQF par expérience contenant les k-mers et leurs nombres d'occurrences dans chaque jeu de données. L'ensemble de ces CQFs sont ensuite fusionnés dans le but d'obtenir un unique CQF en sortie dans lequel chaque k-mer est unique et est associé à un identifiant spécifique, lui même associé à une combinaison précise de jeux de données dans la table d'association.

Pour requêter l'index, la séquence est représentée par son ensemble de k-mers Q. On considère qu'une expérience contient la requête si une fraction suffisante θ de k-mers est retrouvée dans l'expérience. Pour chaque k-mer $k \in Q$, on interroge le CQF pour obtenir les classes de couleurs C_x. La table d'association classe-vecteur est ensuite interrogée pour obtenir le nombre de k-mers de la requête dans chacun des jeux de données.

II.2.3.b Othello hashing et SeqOthello

SeqOthello est une méthode d'indexation basée sur la structure de données Othello[25,26] qui fait partie de la famille des fonctions de hachage minimales et parfaites[27]. Ce type de fonction permet d'indexer n clés sur n entiers consécutifs sans aucune collision. Othello est plus compact qu'une table de hachage classique tout en conservant une recherche clé-valeur en temps constant. Cette méthode exploite cette structure de données afin de stocker des cartes d'occurrences qui associent les k-mers à leurs expériences. SeqOthello utilise une structure hiérarchique avec plusieurs étages de structures Othello (figure 4.a).

Les cartes d'occurrences sont divisées en trois groupes correspondant à trois types d'encodage en fonction de la taille nécessaire à leur stockage. On distingue deux type principaux, le *Value-list encoding* pour les k-mers rares et le *Delta-list encoding* pour les k-mers plus fréquents. Le dernier, *Bitmap encoding*, réserve d'avantage d'espace et est utilisé seulement lorsque les deux autres méthodes ne permettent pas un encodage plus efficace (figure 4.a).

Lors de la requête, une première structure Othello identifie le type d'encodage pour chaque k-mer, après quoi d'autres structures Othello permettent de récupérer les carte d'occurrences. On obtient alors une carte d'occurrences générale pour toutes les expériences pour l'ensemble des k-mers de la requête comme présenté sur la figure 4.b. Ce résultat sous forme de carte d'occurrences permet de rendre compte de la répartition des k-mers le long de la requête.

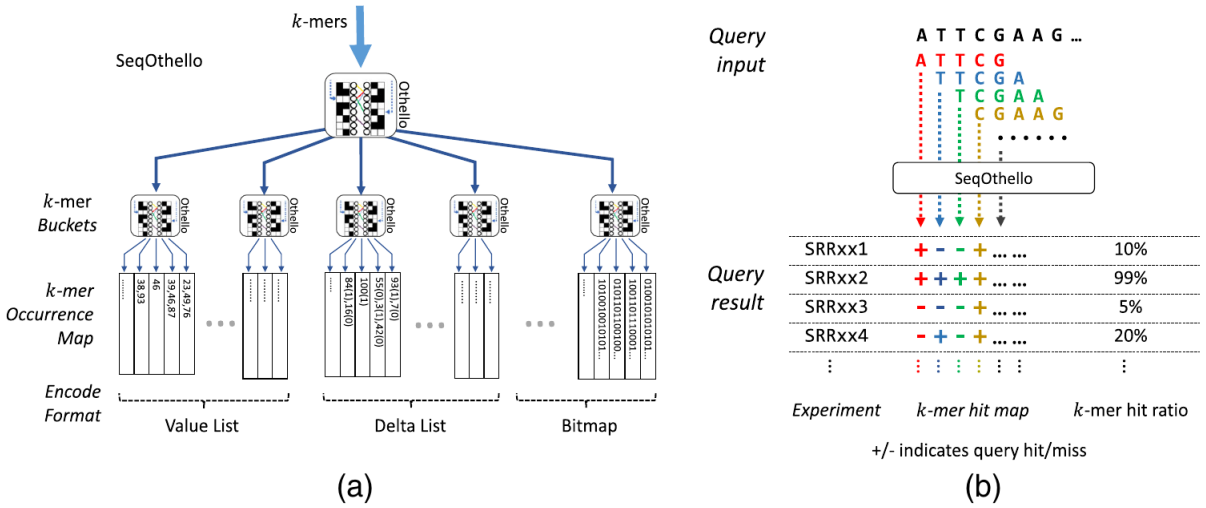

FIGURE 4 – Fonctionnement de SeqOthello. D'après Yu *et al.*
(a) L'association des k-mers à leurs cartes d'occurrences est assurée par une hiérarchie de structure Othello. La racine fait le lien entre un k-mer et son type d'encodage puis une autre structure Othello associe chaque k-mer à sa carte d'occurrences. (b) Exemple de requête. La structure est interrogée pour chacun des k-mers de la requête avec en sortie une carte d'occurrences générale.

II.2.3.c Sequence Bloom Tree (SBT)

Le Sequence Bloom Tree[28] est un index qui utilise comme structure de base des filtres de Bloom. A la différence de BIGSI, ici les filtres sont hiérarchisés sous la forme d'un arbre binaire. Chaque expérience est représentée par un filtre de Bloom à une fonction de hachage contenant l'ensemble des k-mers contenus dans le jeu de données.

Un SBT est construit par l'insertion successive d'expérience de séquençage. Pour insérer une expérience x, on calcule d'abord son filtre de Bloom $B(x)$. L'arbre est ensuite parcouru de la racine vers les feuilles pour insérer $B(x)$. Cette méthode d'insertion permet l'ajout de nouveaux jeux de données a *posteriori* sans recalculer l'ensemble de l'index.

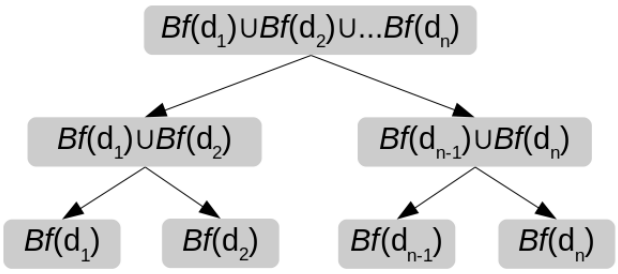

FIGURE 5 – Structure d'un SBT. Chaque noeud est un filtre de Bloom qui correspond à l'union des filtres enfants et la racine est l'union de l'ensemble des filtres de l'arbre. Chaque feuille (d_1...d_n) correspond à une expérience de séquençage indexée.

L'insertion d'un filtre x dans l'arbre se fait récursivement de la manière suivante :

— Pour un noeud u, si u a un seul enfant, le filtre x est inséré comme second enfant de u.

— Si u à deux enfants, x est comparé aux deux filtres de Bloom correspondant. L'enfant avec le filtre le plus similaire par rapport à la distance de Hamming[29] devient le noeud courant et le processus est répété. La distance de Hamming permet de quantifier le degré de similarité entre deux séquences. Pour des tableaux de bits, elle sera égale au nombre de bits qui diffèrent entre les deux tableaux.

— Si u n'a pas d'enfant, c'est une feuille de l'arbre et il représente une expérience de séquençage. Dans ce cas, un nouveau noeud z est créé comme enfant du parent de u. Ce noeud z a deux enfants u et un nouveau noeud représentant x.

La représentation sous forme d'arbre binaire permet un gain de temps non négligeable lors des requêtes. En effet, les filtres de Bloom n'étant pas soumis aux faux négatifs, l'absence d'un k-mer à noeud n arrête immédiatement la requête pour le sous-arbre enraciné à n.

Dans cet arbre, chaque feuille correspond à un filtre de Bloom et représente une expérience séquençage et chaque noeud v est un filtre de Bloom contenant l'ensemble des k-mers de toutes les expériences du sous-arbre enraciné à v. La racine représente l'ensemble des k-mers indexés, c'est l'union de tout les filtres de Bloom de l'arbre. Cette propriété implique qu'une taille suffisante de BF soit utilisée dès le départ pour ne pas saturer les noeuds, en particulier dans le cas d'un index "dynamique" auquel de nouvelles expériences pourraient être ajoutées. En effet, plus la taille des BFs est faible, plus il y aura saturation des noeuds, surtout pour les noeuds proches de la racine. Cette saturation entraîne des temps de requête plus élevés mais aussi un plus grand nombre de faux positifs.

D'autres méthodes sont basées sur SBT : SSBT[30], AllSome[31] et HowDeSBT[32]. Ces méthodes ont pour point commun l'utilisation de deux filtres par noeud de l'arbre. Avec cette représentation, l'information contenue à un noeud v est plus importante comparée à SBT. Cela permet d'une part des requêtes plus rapides, et d'autre part, un index plus efficace en terme d'espace.

HowDeSBT est la méthode la plus récente et la plus efficace. Comme pour SBT, on représente d'abord l'ensemble des expériences à indexer par leurs filtres de Bloom. Puis, à partir de ces filtres, on détermine la topologie de l'arbre à l'aide d'une méthode de clustering utilisant la distance de Hamming.

Cette étape est cruciale puisque l'efficacité de la méthode dépend d'un clustering efficace. En effet, plus les expériences qui se ressemblent en terme de k-mers sont proches dans l'arbre plus l'information peut être regroupée avec un impact à la fois sur l'espace de stockage et sur les temps de requête. Pour cette étape de clustering, HowDeSBT utilise la méthode introduite par Sun et al.[31].

HowDeSBT utilise deux tableaux de bits par noeud : B_{det} et B_{how}. B_{det} va permettre de savoir si à un noeud v, il est possible d'avoir l'information pour les enfants de v, c'est-à-dire si tous les bits à la position i ont la même valeur dans l'ensemble du sous-arbre enraciné à v. Dans le cas où l'on peut déterminer l'information, celle-ci sera donnée par B_{how}.

Pour chaque noeud, $B_{det}(u)[i] = 1$ si tous les $BF[i]$ ont la même valeur pour toutes les feuilles du sous-arbre. $B_{how}(u)[i] = 1$ si $B_{det}(u) = 1$ et $BF[i] = 1$ pour toutes les feuilles du sous-arbre.

De même, $B_{how}(u)[i] = 0$ si $B_{det}(u) = 1$ et $BF[i] = 0$ pour toutes les feuilles du sous-arbre. Pour un noeud u, ces deux vecteurs sont défini comme suit :

$$B_{det}(u) \triangleq B_{\cap}(u) \cup \overline{B_{\cup}(u)} \qquad B_{how}(u) \triangleq B_{\cap}(u) \tag{1}$$

Il existe deux exceptions. $B_{det}(u)[i]$ est inactif si il est déjà déterminé par un ancêtre ($B_{det}(parent(u))[i] = 1$). $B_{how}(u)[i]$ est inactif si il ne peut pas être déterminé à ce noeud ($B_{det}(u)[i] = 0$).

Cette représentation a deux grands avantages. Le premier avantage concerne l'accélération et l'optimisation des requêtes. Contrairement à SBT, où la requête est arrêtée si un k-mer n'est pas présent dans le sous-arbre, ici, la requête est arrêtée dès lors que le bit correspondant peut être déterminé. Avec SBT, lorsqu'un k-mer est présent dans une feuille, il n'y a pas de condition d'arrêt, chaque noeud entre la racine et la feuille correspondante est requêté.

Pour requêter une séquence S (figure 6), celle-ci est représentée par son ensemble de k-mers Q, chaque k-mer est haché pour obtenir la liste des positions représentant Q. Cette liste permet d'obtenir un filtre de Bloom représentant la requête : $BF(Q)$. Une requête est aussi associée à un seuil θ qui définit la fraction minimale de k-mers que doit contenir une expérience pour être retournée par l'algorithme.

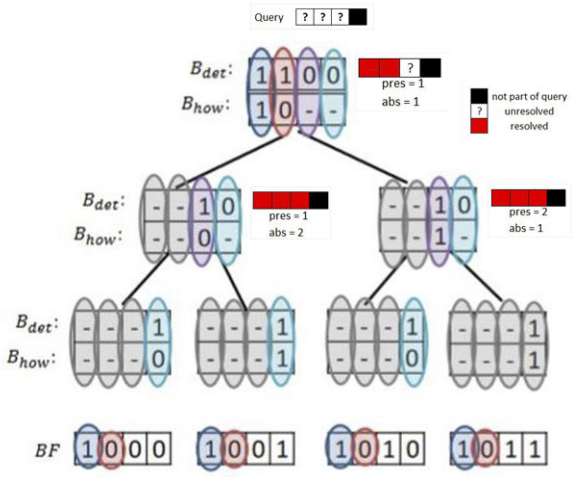

FIGURE 6 – Structure et requête d'un index produit par HowDeSBT. Ici la requête est constituée de trois k-mers. Au cours de la requête, deux compteurs, *pres* et *abs*, sont incrémentés en fonction du nombre de k-mers présents ou absents. Ils permettent l'abandon de la requête lorsqu'un nombre suffisant, ou insuffisant, de k-mers est déterminé à un noeud donné. Dans cet exemple, la requête est retrouvée dans la partie droite de l'arbre mais pas dans la partie gauche (avec un seuil $\theta = 0.5$. Modifié d'après P. Medvedev (DSB 2019).

Chaque requête utilise deux compteurs, l'un pour compter les positions déterminées à 1 (pres) et l'autre pour les positions déterminées à 0 (abs). La requête est ensuite effectuée de manière récursive, de la racine vers les feuilles. Chaque position $BF(Q) == 1$ est comparée à chaque noeud u, l'ensemble des positions déterminées sont retirées de la liste et chaque compteur est incrémenté. Pour un noeud donné v, si $\frac{pres}{|Q|} \geq \theta$ les expériences du sous-arbre enraciné à v sont immédiatement retournées. Au contraire, si $\frac{abs}{|Q|} < 1 - \theta$, le sous-arbre est directement abandonné.

Le second avantage est la réduction de l'espace nécessaire au stockage de l'index. Pour un noeud donné, si un bit peut être déterminé au niveau d'un ancêtre, ce bit est alors considéré

comme inactif et ne sera pas stocké. Cette notion de bit non informatif (figure 7) est introduit par Solomon et al. avec SSBT, un outil plus ancien que HowDeSBT utilisant lui aussi deux filtres par noeud. C'est un algorithme de *Rank-and-Select*[33] qui permet de ne pas de stocker ces bits inactifs. Chaque vecteur est ensuite compressé avec la méthode de Raman-Raman-Rao[34].

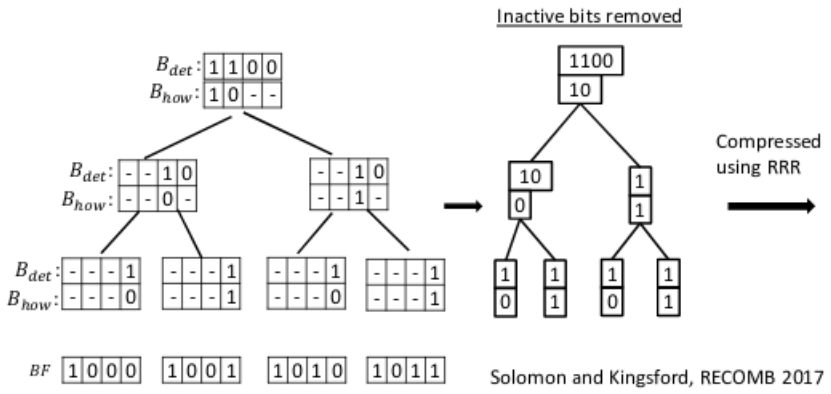

FIGURE 7 – Principe de bits non informatifs. Les tirets représentent les bits non informatifs déterminés au niveau d'un noeud ancêtre. Ceux-ci ne sont pas seulement considérés comme inactifs, ils ne sont jamais intégrés à l'index offrant ainsi un gain en terme d'espace. Ce sont ces vecteurs sans les bits inactifs qui sont ensuite compressés avec la méthode de Raman-Raman-Rao. D'après Solomon and Kingsford (RECOMB 2107).

Ici la construction de l'index est plus rapide par rapport à SBT mais l'insertion de nouveaux de jeux de données sans recalculer l'intégralité de l'index n'est pas aussi trivial qu'avec SBT. Bien qu'en théorie possible, cette fonctionnalité n'est pas encore implémentée.

III Bilan et apports du stage

Depuis SBT, la majorité des méthodes concernant l'indexation de données brutes de séquençage sont testées avec les mêmes données d'évaluations. Il s'agit d'un jeu de données composé de 2652 jeux de reads provenant de SRA pour une taille d'environ 15 teraoctets. Les résultats de ces différents tests sont présentés dans le tableau 1. Attention, ces résultats sont extrapolés à partir des différents articles des méthodes. Les conditions expérimentales sont variables, notamment au niveau des machines utilisées pour les tests.

	SBT	HowDeSBT	Mantis	SeqOthello
Comptage des k-mers (h)	NA	60	Plusieurs jours	NA
Temps de construction (h)	55	10	20	2
Espace intermédiaire (Go)	300	30	3500	190
Taille de l'index (Go)	200	15	32	21
Mémoire (Go)	25	NA	NA	15

TABLE 1 – Evaluation de HowDeSBT, Mantis et SeqOthello. Indexation de 2652 jeux de données. Attention, ces données sont extrapolées à partir des différents articles publiés sur ces méthodes.

BIGSI n'apparaît pas dans tableau puisqu'il a été testé sur des jeux de données différents. Les inconvénients majeurs de BIGSI sont l'utilisation de McCortex[35] comme compteur de k-mers qui pose des problèmes de passage à l'échelle, et l'indexation de l'ensemble des k-mers de chaque

expérience. Les apports très récents de COBS semblent dépasser certaines limitations de BIGSI avec une indexation plus efficace. Cependant, COBS est fortement dépendant l'architecture de la machine qui l'exécute, avec une utilisation de SSD spécifique.

Concernant leurs performances, on remarque tout d'abord que HowDeSBT produit l'index de plus petite taille avec une utilisation de disque intermédiaire aussi inférieure aux autres méthodes. La consommation de mémoire n'est pas renseignée mais au vu de l'implémentation de HowDeSBT, elle doit correspondre environ à la taille d'un filtre de Bloom. Au niveau du comptage des k-mers, Mantis est le moins efficace, à cause de l'utilisation de Squeakr. A terme, ces méthodes devront être capables d'indexer au moins aussi vite que la croissance des bases de données, un temps de comptage trop long est donc problématique. SeqOthello propose des résultats intéressants mais a montré, comme Mantis, des problèmes d'instabilité lors des premiers tests avec certains fichiers impossibles à indexer. Pour toutes ces raisons, nous avons choisi de nous concentrer sur HowDeSBT.

Nous proposons une version modifié de HowDeSBT qui utilise l'outil Simka (section IV.2.1), lui aussi modifié, comme compteur de k-mers dans le but de réduire la consommation de mémoire et les temps de calcul tout en proposant de nouvelles fonctionnalités.

Une première fonctionnalité correspond à une gestion particulière des k-mers rares (uniques dans un jeu de données), souvent associés à des erreurs de séquençages. Nous pensons que dans certains cas, ces k-mers peuvent faire partie intégrante de l'information. En particulier, dans le cas d'études métagénomiques où il est possible que certaines molécules d'ADN soient nettement moins représentées au sein d'un échantillon. Cette fréquence très faible pourrait se traduire par des k-mers présent une unique fois dans un jeu de données. Dans toutes les méthodes précédemment présentées, ces k-mers rares ne sont simplement pas indexés.

Nous proposons d'indexer les k-mers rares dès lors qu'ils existent dans plusieurs jeux de données. C'est-à-dire qu'un k-mer est conservé si son nombre d'occurrences global est strictement supérieur à 1. Toutefois, il convient d'utiliser ces k-mers rares dans un cadre biologiquement cohérent. Nous proposons une méthode compatible avec une gestion des expériences à indexer sous forme de groupe. Ainsi, les expériences avec des métadonnées cohérentes peuvent être regroupées et la vérification de k-mers rares est effectuée au sein de chaque groupe. En effet, il semble peu cohérent de vérifier la présence d'un k-mer rare dans des jeux de données avec un contexte biologique très différent. Par exemple, dans le cas d'études métagénomiques, des prélèvements peuvent être effectués dans diverses conditions (température, pH, luminosité, ...) avec une composition spécifique très différentes. Plus généralement, il s'agit d'éviter de conserver par hasard des k-mers correspondant réellement à des erreurs lors de l'utilisation de jeux de données trop différents pour leur validation. Dans ce cas, un k-mer est indexé si son nombre d'occurrences est strictement supérieur à 1 dans au moins un groupe.

Une seconde nouvelle fonctionnalité concerne la gestion du taux de faux positifs. La majorité des outils se basent sur des structures de données probabilistes et sont donc sensibles aux faux positifs. La méthode proposée ici permet un contrôle sur ces erreurs en précisant un taux de faux positifs attendus à partir duquel les paramètres seront calculés.

L'ensemble des modifications et les protocoles de tests sont détaillés à la section IV.

IV Matériel et méthodes

L'ensemble des scripts et données de tests ainsi que les sources du projet sont disponibles sur un dépôt github. Suite à de très récents changements au niveau des algorithmes, les scripts de test ainsi que le script permettant la gestion globale de l'outil ne sont pour le moment plus utilisables. La documentation doit aussi être modifiée. Une mise à jour sera effectuée dès que possible dans les prochains jours.

IV.1 Données

IV.1.1 Données synthétiques

Pour valider les modifications au cours du développement, un petit jeu de données de test à été constitué. Ce jeu de test est associé à des tests automatiques de non régression réalisés à chaque nouvelle implémentation. Ces tests consistent en la comparaison des filtres de Bloom produits par la version originale de HowDeSBT (Jellyfish-HowDeSBT) et la version proposée ici Simka-HowDeSBT. Dans des conditions similaires d'utilisation (paramètres identiques, sans gestion des k-mers rares), les filtres produits par les deux méthodes doivent être identiques.

Ce jeu de test synthétique est constitué de 5 fichiers fasta avec 400000 reads par fichiers. Pour chaque fichier, chaque k-mer distinct est présent au moins deux fois. Ce jeu de données est désigné par *synth_5* dans la suite du rapport.

IV.1.2 Données Tara

La fondation Tara Océan[36] organise des expéditions dans le but d'étudier la vie marine et les impacts des changements climatiques majeures sur celle-ci. Au cours de ces expéditions, de nombreux prélèvements de micro-organismes sont réalisés dans une optique de séquençage massif, générant ainsi d'énormes quantités de données métagénomiques. Ces données ont permis la découverte de plusieurs milliers d'espèces marines et de plusieurs millions de gènes[37]. L'immensité de cet ensemble de données, actuellement de 400 To, le rend difficile à exploiter. A l'heure actuelle, seul le métatranscriptome assemblé peut être requêté en utilisant la base de données Ocean Gene Atlas[38] (OGA). De plus, ce sont des données hétérogènes représentant un grand nombre d'expériences. Toutes ces expériences sont associées à des métadonnées qui peuvent être prises en compte, ou non, lors du processus d'indexation. Tout ceci fait de ces données un excellent exemple pour évaluer les outils d'indexations.

La structure des données Tara suit un schéma particulier. Chaque prélèvement est associé à une station qui correspond à une zone géographique précise. On retrouve aussi différentes classes de profondeurs et différentes tailles de filtres pour les différentes tailles d'organismes. En général, plusieurs réplicats sont disponibles pour un même triplet station, profondeur et taille de filtre. Dans le cas de Tara, ce sont ces triplets qui peuvent permettre le regroupement des différentes expériences à indexer.

Pour évaluer la méthode en termes de performance et de précision, nous avons utilisé les données publiés dans *Nitrogen-fixing populations of Planctomycetes and Proteobacteria are*

abundant in surface ocean metagenomes[39]. Parmi ces données on retrouve 93 métagénomes issus du projet Tara ainsi que 957 génomes bactériens. On connaît aussi la répartition de ces génomes, obtenue via une méthode de mapping, dans l'ensemble des métagénomes.

Pour la partie précision, seulement 10 métagénomes sur les 93 (figure 2) sont indexés pour des raisons de temps. Les 957 génomes bactériens sont utilisés comme requête pour évaluer la précision de l'index. Pour la partie performance, seulement 3 de ces fichiers sont utilisés. Les 7 autres se sont montrés inutilisables avec Jellyfish[40] (le compteur de k-mer de la version originale de HowDeSBT) à cause de leur taille trop importante qui provoque un bug lors du comptage. Ces jeux de données sont respectivement désignés par *meta_10* et *meta_3* dans la suite du rapport.

Identifiants	Taille fastq	Région	Profondeur	Filtre
ERR598942	112	Pacifique N	DCM	0.22-3
ERR598954	124	Pacifique S-O	SRF	0.22-3
ERR598967	118	Pacifique S-E	SRF	0.22-3
ERR598976	123	Atlantique N-E	SRF	0.22-3
ERR598986	115	Atlantique N-E	DCM	0.22-3
ERR598961	118	Pacifique S-E	DCM	0.22-3
ERR598965	106	Pacifique S-E	DCM	0.22-3
ERR598963	112	Atlantique N-O	SRF	0.22-3
ERR598968	106	Atlantique N-O	SRF	0.22-3
ERR598983	110	Atlantique N-O	SRF	0.22-3

TABLE 2 – Description des expériences à indexer. La taille des fastq est données en go avec un total de 1,2 To. Les séparations correspondent aux groupes constitués pour la recherche des k-mers rares (même région, profondeur et taille de filtre).

IV.2 Outils

IV.2.1 Simka

Simka[41] est un outil développé par l'équipe Genscale sur la base de la librairie GATB-core[42] qui implémente de nombreux algorithmes dédiés au traitement de données NGS. C'est un outil de métagénomique comparative permettant d'étudier la similarité entre jeux de données de séquençages. Pour cela, Simka représente chaque jeu de données par son ensemble de k-mers et calcule des distances écologiques entre ceux-ci. La phase de comptage est basée sur DSK[43], un compteur de k-mers utilisant la librairie GATB-core. En comparaison à d'autres outils (BFCounter, Jellyfish), cette méthode basée sur l'utilisation de disque offre de gros avantages en termes de consommation mémoire. Concernant les temps de calcul, les résultats obtenus avec DSK sont au moins équivalents, pour une utilisation sur SSD, en comparaison avec d'autres outils utilisant de la mémoire.

Simka peut donc être utilisé comme un compteur de k-mers efficace en terme de mémoire et de temps. Dans notre cas, la fonctionnalité la plus intéressante est le traitement simultané de

l'ensemble des jeux de données. Après le comptage, lors de la phase de calcul des distances écologiques, on dispose d'une matrice d'occurrences k-mer/expériences. Pour un k-mer donné, sa présence ou son absence dans chaque jeu de données est alors connu simultanément. C'est cette fonction, non retrouvée chez les autres compteurs de k-mers, qui va permettre le traitement des k-mers rares. Lors de la lecture et dans le cas d'un k-mer vu une seule fois dans un jeu de données, on peut vérifier sa présence ou son absence dans les autres jeux de données. De plus, Simka permet d'obtenir cette matrice d'occurrences k-mer/expériences ligne par ligne sans avoir à la stocker ou à la représenter dans son ensemble. C'est cette particularité qui permet l'utilisation des techniques de communications inter-processus présentées à la section IV.2.2.a.

IV.2.2 Modifications

IV.2.2.a Simka

Dans un premier temps Simka a été modifié afin de réduire les temps de calcul. Les calculs des distances écologiques, n'ayant pas d'intérêt dans notre cas, ont été retirés dans leur intégralité.

Dans un second temps, les modifications ont concerné l'obtention de la matrice avec gestion des k-mers rares et la transmission de celle-ci à HowDeSBT. Une fois cette matrice obtenue, l'idée est de la transmettre de Simka vers HowDeSBT sans stockage sur disque. En réalité, la matrice est obtenue par le calcul de plusieurs sous-matrices par un grand nombre de processus simultanés (plusieurs centaines dans certains cas). Il s'agit donc de faire le lien entre l'ensemble de ces processus et un unique processus de HowDeSBT. Pour cela, le code des deux outils a été modifié. Pour permettre la communication entre ces deux outils, on peut utiliser des techniques de communications inter-processus appelés *named pipe* : Lien documentation (figure 8).

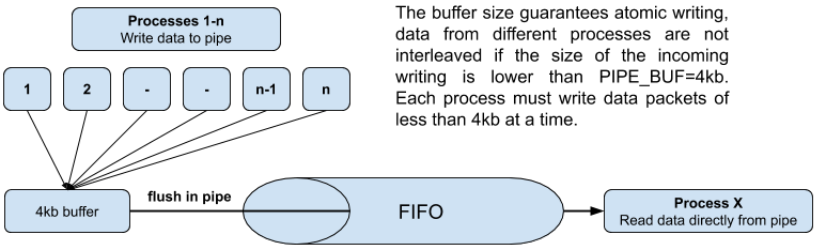

FIGURE 8 – Fonctionnement d'un *pipe*. Un ou plusieurs processus peuvent écrire dans un même *pipe*. Un autre processus est capable de lire directement les données en sortie du *pipe*. Si aucune données ne rentre, le processus en sortie est en attente. De la même manière, si le processus en sortie ne peut pas lire les données, l'écriture est en attente.

Ce sont des zones de données organisées en file d'attente utilisant le principe FIFO (first in, first out). Dès qu'une ligne de la matrice est calculée, elle est transmis au *pipe* et donc directement à HowDeSBT. Cette partie est gérée au niveau de simkaMerge (figure 9.[B]), la partie de Simka normalement en charge du calcul des distances écologiques et qui permet maintenant : le calcul de la matrice d'occurrences avec la prise en charge des k-mers rares et l'écriture de celle-ci dans un *pipe* ou sur disque. Pour l'écriture sur disque, il s'agît pour le moment d'une écriture au format texte dans un fichier compressé GZIP. Une méthode autre, plus efficace, qui utilise des vecteurs binaires compressés est en cours d'implémentation. L'écriture sur disque répond à

un problème rencontré avec *pipe*, celui-ci est détaillé à la section IV.2.2.d. Dans la suite de ce rapport, ces versions seront nommées : Simka-HowDeSBT-disk et Simka-HowDeSBT-pipe.

IV.2.2.b HowDeSBT

La première modification de l'implémentation de HowDeSBT réalisée assure la compatibilité avec la sortie de Simka. C'est-à-dire la prise en charge de la matrice k-mer/expériences comme entrée pour la création des filtres de Bloom. Ce type d'entrée a permis d'accélérer la création des filtres de Bloom. En utilisant Jellyfish, le compteur original de HowDeSBT, chaque jeu de données produit un fichier de sortie contenant les k-mers qui lui sont associés. Les filtres de Bloom sont ensuite créés les uns à la suite des autres à partir de ces fichiers. Avec Simka, chaque ligne de la matrice donne l'information pour les n jeux de données, on peut donc charger les n filtres en mémoire avant de les traiter. Toute cette partie est gérée par la partie *makebf* de HowDeSBT (figure 9.[C]).

Avec Simka, le nombre de k-mers total est estimé avant la création des filtres de Bloom. Cela nous a permis d'implémenter une nouvelle fonctionnalité. Il est maintenant possible d'utiliser un taux de faux positif attendu à partir duquel la taille des filtres de Bloom est calculée. Pour cela, nous avons vérifié que les taux de positifs avec HowDeSBT étaient bien cohérent avec les taux attendus en théorie. La méthode est présentée à la section IV.3.1.

L'utilisation de vecteurs binaires compressés comme entrée pour la création des filtres est en cours d'implémentation.

IV.2.2.c Simka-HowDeSBT

Le script Simka-HowDeSBT.py (figure 9.[A]) permet l'utilisation conjointe des versions modifiées de Simka et HowDeSBT. Il vérifie le bon format des entrées et crée l'arborescence de fichiers avant d'exécuter les deux outils. Il rend aussi possible la gestion par groupe présentée à la section III. Dans ce cas, Simka est exécuté une fois par groupe et chaque groupe produit les filtres de Bloom correspondant aux expériences qui le constitue. Ces filtres sont ensuite utilisés pour construire un seul et même index. La constitution de ces groupes dépend du contexte biologique dans lequel les expériences sont réalisées et est à effectuer manuellement par l'utilisateur.

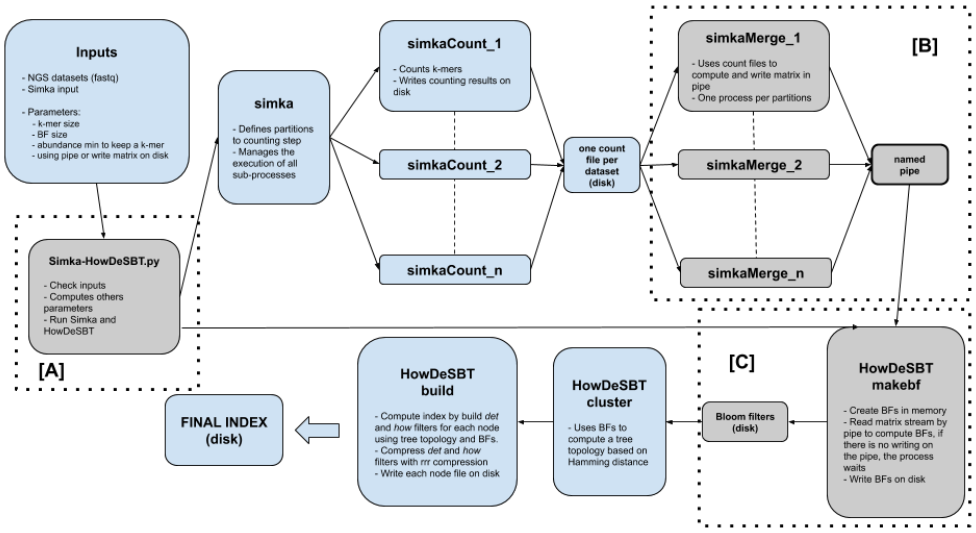

FIGURE 9 – Fonctionnement de HowDeSBT avec utilisation de Simka. Les entrées sont lues par le script Simka-HowDeSBT.py qui se charge de les vérifier et d'exécuter dans le même temps Simka et HowDeSBT qui est au départ en attente, tant qu'il n'y pas d'écriture dans le *named_pipe*. On trouve ensuite une étape de comptage des k-mers avec simkaCount. Celle-ci est suivie du calcul et de l'écriture de la matrice d'occurrence dans le *named_pipe* par un grand nombre de processus simkaMerge. La sortie de *named_pipe* est utilisée au fur et à mesure par HowDeSBT pour calculer les filtres de Bloom. Ces filtres sont ensuite écrits sur le disque avant d'être utilisés pour calculer la topologie de l'arbre. Enfin, les filtres de Bloom et la topologie permettent le calcul et la compression de l'index.

IV.2.2.d Utilisation de pipe et problème de passage à l'échelle

La solution Simka-HowDeSBT-pipe est fonctionnelle pour des jeux de données relativement petits avec peu d'expériences (≤3-4go par expérience). D'après la documentation, les données écrites par plusieurs processus dans un même *pipe* peuvent être intercalées si la taille de l'écriture dépasse la valeur de PIPE_BUF. C'est-à-dire que plusieurs lignes peuvent être mélangées. PIPE_BUF est une constante système qui défini le nombre maximum d'octets atomiques lors de l'écriture dans un *pipe*. Une opération atomique correspond à une opération qui s'exécute sans pouvoir être interrompue et garantie en théorie que les données écrites par plusieurs processus ne soient pas intercalées.

Avec un grand nombre de jeux de données volumineux, le nombre de processus simkaMerge (figure 9) atteint plusieurs centaines de processus simultanés. Dans ce cas, l'écriture de tous ces processus dans le *pipe* pose un problème. Certaines lignes de la matrice peuvent être erronées même si la quantité de données écrite en une seule fois est inférieure à PIPE_BUF.

Lorsque la quantité de données écrite est de taille très inférieure à PIPE_BUF, on observe une réduction du nombre de lignes erronées mais cela ne règle pas le problème.

Ce problème, toujours présent sur la structure de calcul utilisée pour les tests, a très récemment disparu avec les dernières mises à jour de la distribution sur une machine personnelle (debian 10.0, kernel 4.19.0-5). Des investigations plus poussées sont nécessaire pour déterminer les conditions exactes qui conduisent à ce problème. Toutefois, le nombre d'erreurs est très faible avec quelques milliers de lignes erronées pour plusieurs milliards de lignes ($\leq 10^{-6}\%$).

Une solution temporaire permet simplement d'écrire une sous-matrice par processus de simkaMerge au format compressé GZIP.

Si l'utilisation de *pipe* s'avère impossible, un autre type de sortie, plus efficace en terme d'espace que la simple écriture de fichier texte est en cours d'implémentation. Celle-ci utilise des vecteurs binaires pour représenter les sous-matrices. Le nombre de bits nécessaire pour stocker une ligne de la matrice correspond ici à $2k + n$ où k est la taille des k-mers et n le nombre d'expériences contre $8k + 8n$ en format texte. Une compression de Raman-Raman-Rao[34] est ensuite appliquée à ces vecteurs en utilisant l'implémentation disponible avec la librairie SDSL-lite (github).

IV.3 Benchmarks

IV.3.1 Taux de faux positifs

Dans cette partie, les taux de faux positifs concernent les requêtes par k-mer et non la précision lorsqu'une séquence de n k-mers est requêtée (section IV.3.2).

A partir du nombre de fonctions de hachage, une dans notre cas, du nombre d'éléments à indexer et de la taille du filtre de Bloom, on peut calculer un taux de faux positifs théorique. Pour cela, on utilise la formule : $p = (1 - e^{-kn/m})^k$ où p est le taux de faux positifs, k est le nombre de fonctions de hachage, n est le nombre d'éléments à indexer et m est la taille du filtre de Bloom. Afin de vérifier la correspondance des résultats de HowDeSBT avec les valeurs théoriques, des k-mers sont générés aléatoirement et attribués à n expériences. Il est possible que certaines expériences ne contiennent aucun k-mers. Pour chaque expérience, un fichier fasta avec les k-mers correspondants est créé et sera utilisé ensuite comme fichier de requête. Une matrice est aussi créée pour être utilisée comme fichier d'entrée dans HowDeSBT.

Une fois l'index crée par HowDeSBT, chaque fichier fasta est requêté avec une requête par k-mer. La liste des k-mers contenus dans les expériences étant connue, on peut déterminer le nombre de requêtes erronées à partir duquel on calcul ensuite le taux de faux positifs.

Ce protocole permet aussi de vérifier le bon fonctionnement de l'outil en s'assurant, durant la requête, que chaque k-mer indexé est bien retrouvé et qu'aucune requête ne renvoie une expérience sans k-mers.

IV.3.2 Performances et précision

Les tests sont effectués sur le cluster de calcul GenOuest sur un noeud disposant de 24 processeurs Intel Xeon E5-2640 (2.50GHz) et de 190Go de mémoire. Chaque test utilise un maximum de 16 coeurs. Les temps de calculs et la consommation mémoire sont déterminés à l'aide la commande système */usr/bin/time*.

La description des tests concernant les temps de calcul et l'utilisation de mémoire sont décrits dans le tableau 3. Ces tests permettent de comparer Jellyfish-HowDeSBT et Simka-HowDeSBT-disk avec le jeu de données *meta_3*. La version originale Jellyfish-HowDeSBT est utilisée comme décrit sur le dépôt du projet (HowDeSBT/reproduce).

Test	Fichiers indexés	Taille BF	K	Min
1	ERR598942 ERR598965 ERR598967	5x10^9	21	1
2			21	2
3			30	2

TABLE 3 – Paramètres des tests. **Taille BF** : nombre de bits par filtre de Bloom, **K** : longueur des k-mers, **Min** : nombre minimum d'occurrences d'un k-mer pour qu'il soit conservé. Ces 3 combinaisons de paramètres sont utilisées pour Jellyfish-HowDeSBT et Simka-HowDeSBT-disk

Le problème avec *pipe* étant toujours présent sur le cluster GenOuest, Simka-HowDeSBT-pipe n'est pas testé sur ces données. Un test avec de petits jeux de données est effectué dans le but de comparer les performances entre l'utilisation de *pipe* et Jellyfish-HowDeSBT. Ce test utilise le jeu de données *synth_5* présenté à la section IV.1.1.

La précision de la méthode est évaluée en indexant les 10 métagénomes (jeu de données *meta_10* qui ont des métadonnées compatibles avec la formation de groupes d'expériences. Les génomes bactériens sont ensuite requêtés sur l'index. Une matrice d'abondance génomes/expériences obtenue par mapping est disponible avec les données. Celle-ci est transformée en matrice de présence/absence et est utilisée comme base de vérité.

Ce jeu de données est utilisé pour trois tests : sans considération des k-mers rares, avec k-mers rares et avec k-mers rares et gestion par groupe. Lors du test sans k-mers rares, on conserve les k-mers avec un nombre d'occurrences au moins égal à 2 dans au moins un jeu de données. Pour le test avec groupe, un k-mer est conservé si il est présent au moins deux fois dans un jeu de données, où présent au moins 2 fois au sein d'un même groupe. Pour ces deux tests, la taille des filtres de Bloom est fixée à 20 milliards. Une fois les index obtenus, les 957 génomes sont requêtés et les résultats sont comparés avec la matrice de présence/absence.

V Résultats

Dans cette section, les résultats concernant Simka-HowDeSBT correspondent à une utilisation avec écriture sur disque, à l'exception des résultats présentées à la section V.2.2. Pour les performances, les résultats sont présentés en fonction des trois différentes étapes de l'indexation : le comptage des k-mers, le calcul des filtres de Bloom, et la construction/compression de l'index.

V.1 Taux de faux positifs en k-mer

Pour comparer les valeurs théoriques aux valeurs obtenues avec Simka-HowDeSBT, 100 000 k-mers sont attribués de manière aléatoire à 10 expériences avant d'être indexés. Les résultats des requêtes permettent d'observer des taux de faux positifs qui peuvent être comparés aux valeurs attendues en théorie. Les résultats sont présentés sur la figure 10.

On remarque que les taux de faux positifs observés avec Simka-HowDeSBT suivent bien les valeurs théoriques. Un taux de faux positif acceptable, de 1%, est atteint avec un filtre de 10 millions de bits, soit 100 fois le nombre d'éléments à indexer. Pour atteindre 0.1% de faux positifs, il faut encore multiplier la taille du filtre par un facteur 10 soit 100 millions de

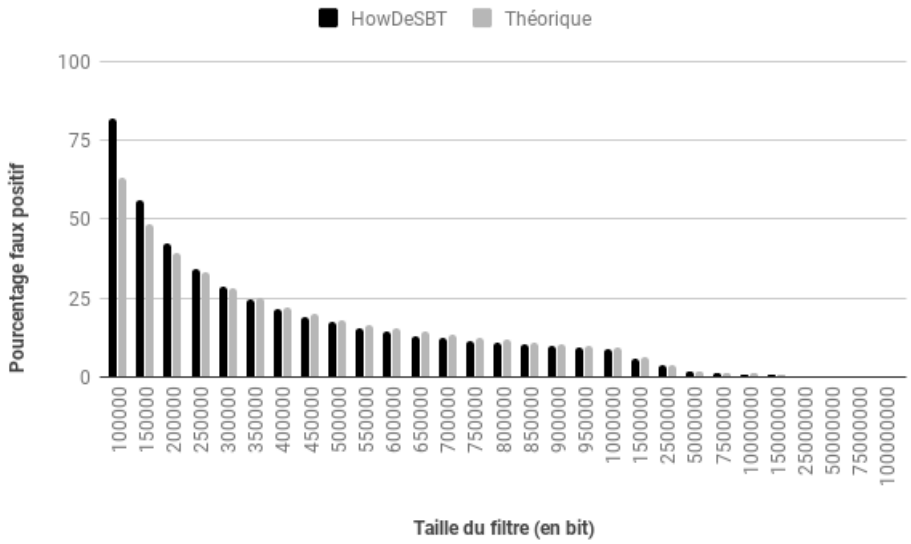

FIGURE 10 – Taux de faux positifs en fonction de la taille de filtre de Bloom pour une indexation de 100 000 éléments. Les valeurs théoriques sont comparées aux valeurs calculées avec Simka-HowDeSBT.

bits. On peut estimer la taille de filtre nécessaire pour un taux de faux positifs recherché avec l'approximation : $m = \frac{n}{p}$. Cette estimation grossière est seulement valable dans un cas à une fonction de hachage et pour des taux de faux positifs recherchés relativement faible ($\leq 10\%$).

V.2 Performances

V.2.1 Écriture sur disque

Cette section compare les performances de Jellyfish-HowDeSBT et Simka-HowDeSBT-disk et a pour objectif une comparaison sur des jeux de données réelles, Simka-HowDeSBT-pipe n'étant pour le moment pas utilisable sur ce type de données (jeu de données *meta_3*. Bien que l'écriture sur disque au format compressé GZIP influe négativement sur les performances, les résultats présentés ici permettent d'avoir une première idée des différences de performances entre ces deux méthodes. La figure 11 présente les résultats pour trois combinaisons de paramètres.

On remarque tout d'abord que le temps nécessaire à la construction (label *index* sur la figure) de l'index à partir des filtres de Bloom est équivalent pour l'ensemble des tests. Pour la partie comptage, Simka réduit les temps de calcul d'environ 11 à 22%. Pour le calcul des filtres de Bloom la différence est moins importante avec une réduction d'environ 8 à 11%. La mémoire utilisée par Simka-HowDeSBT est aussi moins importante et reste constante en fonction de la valeur de k. Pour Jellyfish-HowDeSBT, la taille des k-mers a une influence sur la consommation de mémoire avec une augmentation de 77% lorsque k passe de 21 à 30.

On observe aussi de grandes différences de performance entre J_k21_min1 et S_k21_min1. Ces deux tests ne sont pas vraiment comparables puisque dans un cas, Jellyfish garde l'ensemble des k-mers rares ($\approx$ 16 milliards) et dans l'autre cas Simka garde les k-mers rares si ils sont observés dans au moins un autre jeu de données ($\approx 7,9$ milliards).

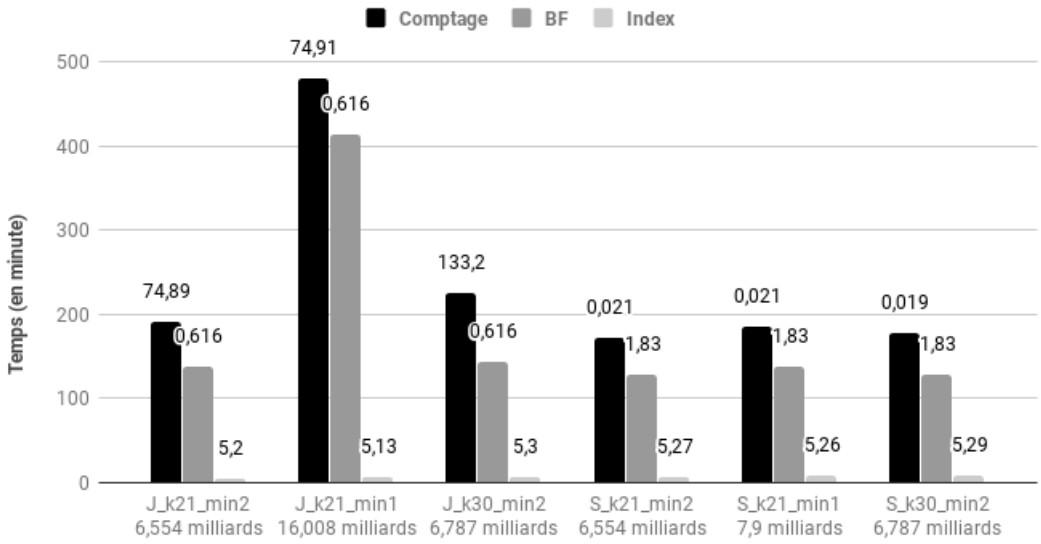

FIGURE 11 – Temps de calcul pour l'indexation de *meta_3* pour différents paramètres. **Comptage** correspond à la partie de comptage des k-mers. **BF** correspond à la création des filtres de Bloom. **Index** correspond à la construction et la compression de l'index. **J** : Jellyfish-HowDeSBT. **S** : Simka-HowDeSBT-disk. **k** : Taille des k-mers. **min** : Nombre d'occurrences pour conserver un k-mer. La mémoire utilisée est indiquée au dessus des barres (en gigaoctets). Le nombre de k-mers indexés est indiqué sous les barres.

	Temps de calcul			Mémoire
	Comptage	BF	Total	
Jellyfish-HowDeSBT	281	61	342	26
Simka-HowDeSBT-pipe	110		110	0.09

TABLE 4 – Comparaison de Jellyfish-HowDeSBT et Simka-HowDeSBT-pipe pour l'indexation du jeu de données *synth_5*. Les temps de calcul sont en seconde et la consommation de mémoire en gigaoctets.

V.2.2 Utilisation de pipe

Dans cette partie, les performances de Jellyfish-HowDeSBT et Simka-HowDeSBT-pipe sont comparées pour le jeu de données synthétique *synth_5*. Pour ce test, on utilise des k-mers de taille $k = 21$ et on indexe l'intégralité des k-mers retrouvés dans les 5 jeux de données (environ 80 millions).

Les résultats présentés dans la table 4 montrent des performances accrues pour Simka-HowDeSBT-pipe par rapport à Jellyfish-HowDeSBT.

Dans le cas de Jellyfish-HowDeSBT, le comptage suivi du calcul des filtres de Bloom s'effectue en 342 secondes contre 110 secondes pour Simka-HowDeSBT-pipe. Le temps de calcul est réduit d'un facteur 3. La consommation de mémoire est aussi bien inférieur pour Simka-HowDeSBT-pipe avec seulement 90 mégaoctets.

V.3 Comparaison avec une méthode de mapping

L'indexation du jeu de données *meta_10* est réalisée uniquement avec Simka-HowDeSBT-disk et avec 3 combinaisons de paramètres. Les résultats pour chaque étape de l'indexation sont présentés sur la figure 12.

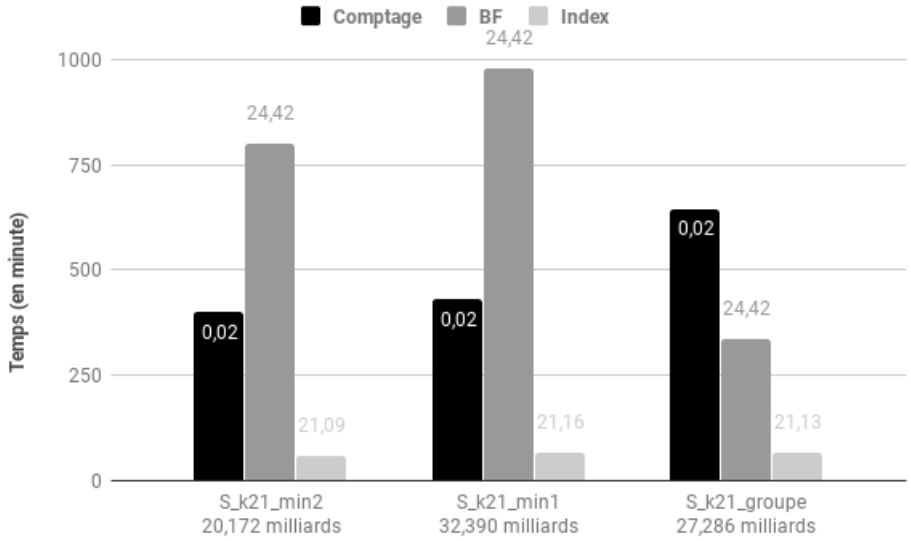

FIGURE 12 – Temps de calcul pour l'indexation de 10 fichiers fasta pour différents paramètres. **Comptage** correspond à la partie de comptage des k-mers. **BF** correspond à la création des filtres de Bloom. **Index** correspond à la construction et la compression de l'index. **S** : Simka-HowDeSBT-disk. **k** : Taille des k-mers. **min** : Nombre d'occurrences pour conserver un k-mer. La mémoire utilisée est indiquée au dessus des barres (en gigaoctets). Le nombre de k-mers indexés est indiqué sous les barres.

Bien que destiné à l'évaluation de la précision de la méthode, l'indexation de ce jeu de données *meta_10* apporte également des informations de performances intéressantes.

On observe ici un effet de la gestion particulière des k-mers rares. En effet, dans cet ensemble de jeux de données le nombre total de k-mers distincts atteint 54 milliards. L'élimination des k-mers rares non partagés permet de réduire le nombre de k-mers à indexer à environ 32,4 milliards pour S_k21_min1. La gestion par groupe a aussi un effet avec un nombre de k-mers indexés d'environ 27,3 milliards.

Aussi, on note l'impact du nombre de k-mers sur le temps de calcul des filtres de Bloom sauf dans le cas de S_k21_groupe. Celui-ci montre un temps de calcul des filtres de Bloom plus faible que S_k21_min2 alors qu'il index un plus grand nombre de k-mers. La gestion par groupe est la raison de ce résultat. Une fois le comptage des k-mers effectué pour chacun des groupes, le calcul des filtres de Bloom, ne demandant que peu de ressources, peut être réalisé parallèlement pour chacun des groupes. C'est aussi cette gestion par groupe qui augmente le temps de calcul pour la partie comptage.

Concernant la taille des index, on obtient des index d'environ 17 Go pour S_k21_min2 et S_k21_groupe. Avec S_k21_min1, l'index obtenu est légèrement plus imposant avec 19 Go. Des résultats raisonnables par rapport à la quantité de données indexées qui représente environ 1,2 To.

L'ensemble des 957 génomes bactériens sont requêtés sur les 3 index. Les résultats concernant les temps de requêtes sont présentés sur la figure 13 pour 5 valeurs différentes de θ. Pour rappel, ce seuil correspond à la fraction de k-mer de la requête que doit contenir une expérience pour que l'on considère la séquence comme contenue dans l'expérience.

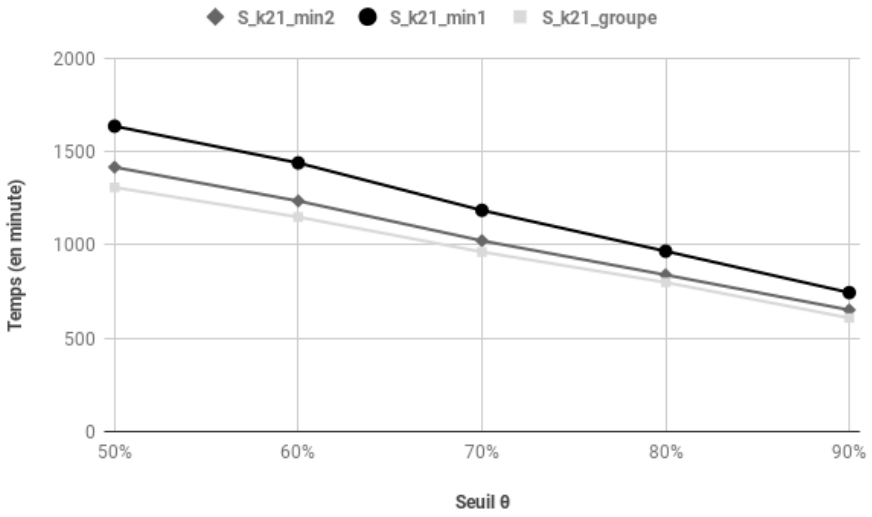

FIGURE 13 – Temps de requête en fonction du seuil θ pour les 3 combinaisons de paramètres. Requête des 957 génomes pour un total d'environ 320 000 séquences.

Comme le montre la figure 13, la valeur de θ a une forte influence sur les temps de requête avec une réduction d'un facteur 2 entre $\theta = 50\%$ et $\theta = 90\%$. Les différents paramètres ont moins d'impact sur le temps de calcul. On peut tout de même noter des valeurs plus importantes pour S_k21_min1. L'ensemble de ces 957 génomes représentent environ 320000 séquences ce qui correspond à environ une séquence requêtée toute les 0,25 seconde pour $\theta = 50\%$.

Les résultats des requêtes sont comparés aux résultats de mapping. On peut alors calculer les taux de vrais/faux positifs et vrais/faux négatifs (figure 14). Attention, bien que nous employons les termes vrais/faux positifs/négatifs, il s'agît d'un abus de langage. HowDeSBT n'est pas comparé à une réalité mais a une autre méthode permettant la recherche de séquence dans un ensemble de données[44].

Les paramètres de construction de l'index ainsi que les valeurs θ ont un impact sur les résultats des requêtes. On observe une diminution des taux de vrais et faux positifs avec l'augmentation de la valeur θ. Inversement, l'augmentation de θ entraîne une augmentation des taux de vrais et faux négatifs.

Pour les taux de vrais et faux positifs, on observe un comportement similaire en fonction des combinaisons de paramètres. Plus l'index contient un grand nombre de k-mers, plus les taux de vrais et faux positifs sont élevés. Bien sûr, c'est l'inverse qui se produit pour les taux de vrais et faux négatifs.

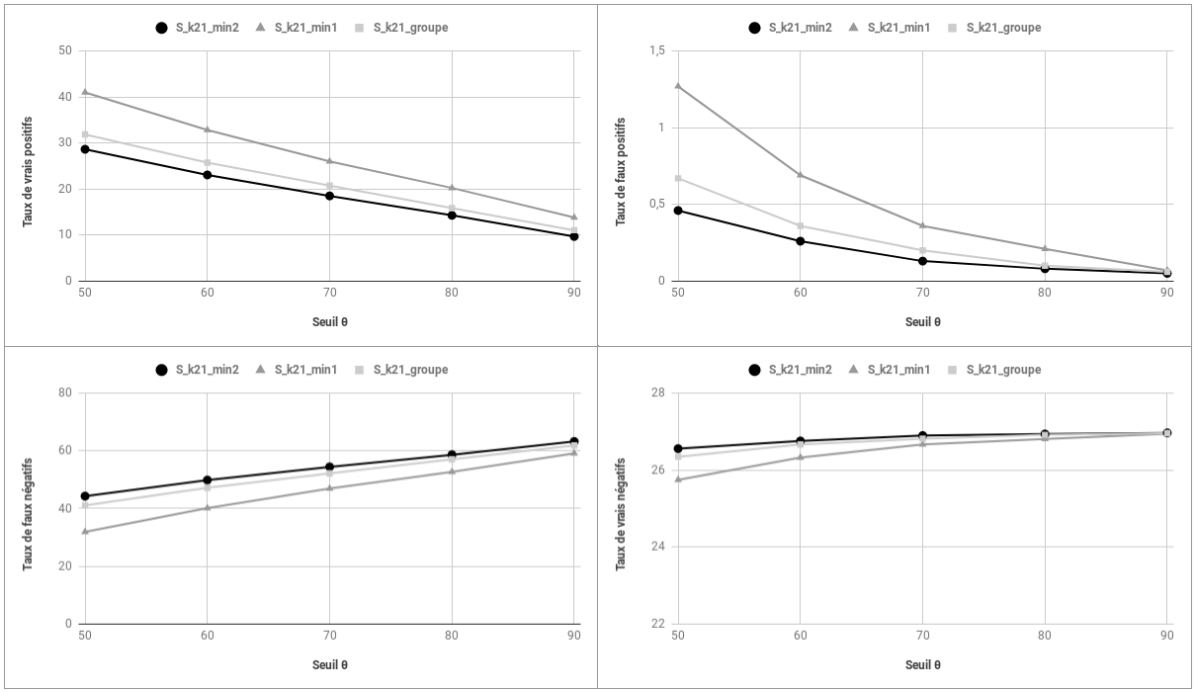

FIGURE 14 – Résultats des requêtes des 957 génomes pour les trois combinaisons de paramètres. De gauche à droite et de haut en bas : vrai positif, faux positif, faux négatif, vrai négatif.

VI Discussion

Les méthodes d'indexation utilisent pour la plupart des structures de données probabilistes assumant la présence de faux positifs. Cette imprécision doit être contrôlable et vérifiable pour assurer une utilisation correcte des résultats. La comparaison des taux de faux positif observées avec HowDeSBT et des taux théoriques valide le fait que la représentation hiérarchique des filtres de Bloom n'a pas d'influence sur les taux de faux positifs. Il s'agît d'une représentation facilitant le stockage et la requête avec une conservation des propriétés des filtres de Bloom. Néanmoins, la comparaison montre une des contraintes associée aux filtres de Bloom qui demandent un espace de stockage important pour maintenir un taux de faux positif faible lors de l'indexation d'un grand nombre d'éléments. Ceci est à nuancer dans le cas de HowDeSBT, et plus généralement, pour les méthodes d'indexation basées sur l'utilisation de k-mers. En effet, comme constaté dans notre cas, les taux de faux positifs restent faibles ($< 1,5\%$) alors que seulement 20 milliards de bits sont utilisés par filtre de Bloom. Pour rappel, les trois tests indexent environ 20, 27 et 32 milliards de k-mers. Bien sûr, si l'on résonne en terme k-mers, les taux de faux positifs sont élevés mais ils sont compensés lors de la requête. Plus la séquence requêtée est longue, plus elle contient de k-mers. Un grand nombre de k-mers couplé à un seuil θ, élevé lui aussi, permet de conserver une bonne qualité de résultat. Cette propriété est décrite précisément par Solomon & Kingsford dans le premier papier traitant de Sequence Bloom Tree[28]. L'utilisation de filtre de Bloom de taille plus ou moins importante dépend donc aussi du type de requête que l'on souhaite effectuer. Ainsi, dans le cas de requête courte, une taille importante de filtre de Bloom est a privilégier. C'est précisément dans ce cas que la fonction permettant le calcul de la taille des filtres à partir du nombre de k-mers à indexer trouve son utilité.

La comparaison de Jellyfish-HowDeSBT et Simka-HowDeSBT montre une nette amélioration des performances avec l'utilisation de *pipe*. Dans le cas de l'écriture de la matrice sur le disque observe une amélioration moins importante de l'ordre de 11 à 18% en fonction des tests. La réduction des temps de calcul est aussi valable lorsque l'on considère les k-mers rares avec une très légère augmentation de la durée de l'étape de comptage pour Simka. Pour Jellyfish, le comptage des k-mers rares augmente fortement le temps de calcul. Les évaluations sur l'ensemble des 10 jeux de données ont permis de révéler cette différence entre Simka et Jellyfish avec les tests J_k21_min1 et S_k21_min1. Le calcul des filtres de Bloom n'est pas comparable pour ces deux tests puisque Jellyfish conserve la totalité des k-mers distincts tandis que Simka conserve un k-mer rare uniquement si il est présent dans un autre jeu de données. Néanmoins, pour l'étape de comptage, l'intégralité des k-mers est bien comptée par les deux outils. Simka prend simplement en compte la gestion particulière des k-mers rares pour adapter sa sortie. On constate alors que le nombre d'occurrences minimales pour qu'un k-mer soit conservé n'a que très peu d'impact sur les temps de calcul avec Simka. En revanche, avec Jellyfish, le temps de calcul est multiplié par environ 2,5 lorsque le nombre d'occurrences minimum pour conserver un k-mer passe de 2 à 1. Ceci semble contredit par les résultats présentés sur la figure 12. En effet, un temps de comptage bien supérieur est observé pour S_k21_groupe. Ce résultat s'explique par la gestion par groupe qui demande une exécution de Simka pour chacun des groupes. Dans ce cas, les k-mers partagés entre ces groupes sont comptés, et donc écrits sur disque, plusieurs fois, ce qui réduit les performances. Une méthode plus efficace, pas encore implémentée, est la gestion directe de ces groupes au sein de Simka et non par Simka-HowDeSBT.py comme décrit à la section IV.2.2.c. Cependant, c'est aussi ce fonctionnement qui réduit le temps nécessaire au calcul des filtres de Bloom pour ce test. Une fois les k-mers comptés, le calcul des filtres ne demandant que peu de ressources, celui-ci est effectué en parallèle pour chacun des groupes entraînant une diminution du temps de calcul. Ce calcul parallèle est amené à disparaître avec l'utilisation de *pipe* qui le rend impossible. Le problème rencontré avec l'utilisation de *pipe* n'a pas permis de réaliser des tests sur les même données. Néanmoins, le test sur des données simulées montre une amélioration des performances en comparaison à l'écriture sur disque. Des recherches précises doivent être effectuées pour déterminer les raisons de ce problème qui semble être en lien avec le noyau Linux utilisé. Bien sûr, l'efficacité de *pipe* doit aussi être confirmée sur des jeux de données plus volumineux.

Une méthode d'indexation efficace ne doit pas seulement permettre une construction rapide de l'index et son stockage sur un espace raisonnable. La finalité de ce type d'approche est l'exploitation conjointe de grands ensembles de données. L'efficacité des requêtes a alors une grande importance. Ici, les résultats semblent compatibles avec l'utilisation de Simka-HowDeSBT comme moteur de recherche pour des jeux de données de séquençage non assemblés avec des temps de requête par séquence inférieurs à la seconde. L'influence du seuil θ est expliqué par le système de requête de HowDeSBT. Lors de la requête d'une séquence, dès que le nombre de k-mers absent est supérieur à $1 - \theta$ la requête est immédiatement arrêtée, avec une diminution des temps de calcul pour des θ élevés. La taille des index finaux pour les différents tests constitue

aussi un résultat intéressant. En effet, les index ne dépassent pas les 20 gigaoctets pour indexer un total de 1,2 teraoctets de données.

Les tests réalisés sont évidemment loin de la réalité des bases de données qui accumulent toujours plus de données. L'utilisation d'un ensemble de données plus conséquent permettrait de mieux rendre compte des performances de la méthode dans un cas réaliste d'indexation. En revanche, ils montrent l'efficacité de la méthode pour des ensembles de données plus raisonnable. Cette approche pourrait être utile pour des projets à moindre échelle en assurant des requêtes très rapides et facilitant ainsi l'exploration des données. Dans un cas d'utilisation concrète, une machine permettant le stockage en mémoire de l'index dans son intégralité doit être privilégiée pour obtenir des temps de requête encore plus confortables.

La comparaison de HowDeSBT avec la méthode issue du papier d'où proviennent les données permet de rendre compte des effets du traitement des k-mers rares sur la qualité de l'indexation. Plus la quantité de k-mers indexés est importante plus la part de "vrais positifs" augmente mais reste faible ($\leq 40\%$). Dans le même temps, on constate un plus grand nombre de faux positifs. Le traitement particulier des k-mers rares a été pensé pour augmenter la précision de l'outil tout en gardant un contrôle sur les faux positifs. C'est aussi dans cette optique que nous avons pensé la gestion par groupe. L'objectif est d'éviter de conserver un k-mer par le simple fait du hasard, avec par exemple un k-mer erroné dans un jeu de données qui correspond à un k-mer bien réel dans un autre. Pour cela, les groupes doivent être constitués de manière cohérente. Bien sûr, de nombreux k-mers peuvent être partagées par des organismes très différents et dans ce cas précis la formation de groupe réduit la précision de la méthode. D'un point de vue biologique, l'objectif est d'accroître la précision avec un plus grand nombre de k-mers considérés tout en limitant les erreurs. Aussi, d'un point de vue technique, nous sommes limités par l'efficacité des méthodes qui ne permettent pas encore l'indexation d'un nombre illimité de k-mers. Il s'agissait donc de trouver un compromis entre précision, taux d'erreurs et performances.

Ces résultats sont discutables à plusieurs niveaux. Tout d'abord, il s'agît d'une comparaison de deux méthodes. Simka-HowDeSBT n'est pas comparé à une réalité, même si les méthodes par alignement sont réputées plus précises. La représentation des séquences par un ensemble de k-mers rend la méthode plus sensible à la variabilité des séquences. Une mutation sur un unique nucléotide, peut fortement affecter la requête dans le cas de méthodes basées sur des k-mers tandis que l'effet sera moins important dans le cas de méthodes par alignement.

Par ailleurs, la méthode présentée dans ce rapport n'est pas spécifiquement destinée à ce type d'étude et n'a donc pas pour objectif d'obtenir une précision équivalente. Ce type de méthode est d'avantage amenée à être utilisée comme filtre rapide permettant de cibler efficacement un sous-ensemble d'expériences avant de les utiliser dans un cadre précis d'analyses faisant appel à d'autres outils plus spécifiques.

VII Conclusions

Les nouvelles implémentations présentées dans ce rapport améliorent les méthodes précédentes avec un gain de performance en terme de mémoire et de temps. De plus, ce gain est conservé même avec l'apport de nouvelles fonctionnalités, comme le traitement des k-mers rares qui ne sont simplement pas traités par les autres méthodes.

Néanmoins, avant d'être pleinement utilisable, plusieurs points importants de la méthode demandent à être éclaircis. D'abord, il sera nécessaire de déterminer les raisons précises du problème avec *pipe* avant de confirmer l'efficacité de cette technique à grande échelle. Ensuite, l'impact du traitement des k-mers rares sur la qualité de l'index devra être clairement quantifiable, tout comme l'impact de la gestion par groupe.

Plus généralement, l'étude de l'état de l'art montre que les méthodes existantes ne permettent pas encore l'indexation de bases de données de manière systématique. La durée de l'indexation ainsi que la taille de l'index final sont encore trop élevées pour l'indexation d'une base de données comme SRA qui dispose de plusieurs pétaoctets de données. Depuis l'apparition des premières méthodes d'indexation pour les données biologiques, chaque modification ou nouvelle approche permet une légère amélioration des méthodes précédentes mais sans répondre au problème de passage à l'échelle. A l'avenir les méthodes proposées devront réduire les temps de calcul et l'espace de stockage de plusieurs centaines de facteurs pour permettre l'exploitation des bases de données dans leur intégralité.

Références

1. Lander, E. S. et al. Initial sequencing and analysis of the human genome. Nature **409,** 860-921. issn : 0028-0836 (15 fév. 2001).
2. Adams, M. D. et al. The genome sequence of Drosophila melanogaster. Science (New York, N.Y.) **287,** 2185-2195. issn : 0036-8075 (24 mar. 2000).
3. Stephens, Z. D. et al. Big Data : Astronomical or Genomical ? PLoS Biology **13.** issn : 1544-9173. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4494865/ (2019) (7 juil. 2015).
4. Moore, G. Cramming More Components Onto Integrated Circuits. Proceedings of the IEEE **86,** 82-85. issn : 0018-9219, 1558-2256. http://ieeexplore.ieee.org/document/658762/ (2019) (jan. 1998).
5. Leinonen, R., Sugawara, H. & Shumway, M. The Sequence Read Archive. Nucleic Acids Research **39,** D19-D21. issn : 0305-1048. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3013647/ (2019) (Database issue jan. 2011).
6. Schildgen, V. & Schildgen, O. How is a molecular polymorphism defined ? Cancer **119,** 1608-1608. issn : 1097-0142. https://www.onlinelibrary.wiley.com/doi/abs/10.1002/cncr.27966 (2019) (2013).
7. Janin, L., Schulz-Trieglaff, O. & Cox, A. J. BEETL-fastq : a searchable compressed archive for DNA reads. Bioinformatics **30,** 2796-2801. issn : 1367-4803, 1460-2059. https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/btu387 (2019) (1er oct. 2014).
8. Dolle, D. D. et al. Using reference-free compressed data structures to analyze sequencing reads from thousands of human genomes. Genome Research **27,** 300-309. issn : 1088-9051, 1549-5469. http://genome.cshlp.org/lookup/doi/10.1101/gr.211748.116 (2019) (fév. 2017).
9. Ferragina, P. & Manzini, G. Opportunistic data structures with applications in Proceedings 41st Annual Symposium on Foundations of Computer Science Proceedings 41st Annual Symposium on Foundations of Computer Science (nov. 2000), 390-398.
10. Burrows, M. & J. Wheeler, D. A Block-Sorting Lossless Data Compression Algorithm. Digital Systems Research Center Research Reports **1** (1995).
11. Bruijn, N. G. d. A combinatorial problem, 7.
12. Grabherr, M. G. et al. Full-length transcriptome assembly from RNA-Seq data without a reference genome. Nature Biotechnology **29,** 644-652. issn : 1546-1696 (15 mai 2011).

13. Haas, B. J. et al. *De novo* transcript sequence reconstruction from RNA-seq using the Trinity platform for reference generation and analysis. Nature Protocols **8,** 1494-1512. issn : 1750-2799. https://www.nature.com/articles/nprot.2013.084 (2019) (août 2013).

14. Li, B. et al. Evaluation of de novo transcriptome assemblies from RNA-Seq data. Genome Biology **15,** 553. issn : 1474-760X. https://doi.org/10.1186/s13059-014-0553-5 (2019) (21 déc. 2014).

15. Wittler, R. Alignment- and reference-free phylogenomics with colored de-Bruijn graphs. arXiv :1905.04165. https://pub.uni-bielefeld.de/record/2935604 (2019) (2019).

16. Uricaru, R. et al. Reference-free detection of isolated SNPs. Nucleic Acids Research **43,** e11-e11. issn : 0305-1048. https://academic.oup.com/nar/article/43/2/e11/2414265 (2019) (30 jan. 2015).

17. Smith, T. F. & Waterman, M. S. Identification of common molecular subsequences. Journal of Molecular Biology **147,** 195-197. issn : 0022-2836. http://www.sciencedirect.com/science/article/pii/0022283681900875 (2019) (25 mar. 1981).

18. Bloom, B. H. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM **13,** 422-426. issn : 00010782. http://portal.acm.org/citation.cfm?doid=362686.362692 (2019) (1er juil. 1970).

19. Bradley, P., Bakker, H. C. d., Rocha, E. P. C., McVean, G. & Iqbal, Z. Ultrafast search of all deposited bacterial and viral genomic data. Nature Biotechnology **37,** 152. issn : 1546-1696. https://www.nature.com/articles/s41587-018-0010-1 (2019) (fév. 2019).

20. Bingmann, T., Bradley, P., Gauger, F. & Iqbal, Z. COBS : a Compact Bit-Sliced Signature Index. arXiv :1905.09624 [cs]. arXiv : 1905.09624. http://arxiv.org/abs/1905.09624 (2019) (23 mai 2019).

21. Pandey, P. et al. Mantis : A Fast, Small, and Exact Large-Scale Sequence-Search Index. Cell Systems **7,** 201-207.e4. issn : 2405-4712. http://www.sciencedirect.com/science/article/pii/S2405471218302394 (2019) (22 août 2018).

22. Pandey, P., Bender, M. A., Johnson, R. & Patro, R. Squeakr : an exact and approximate k-mer counting system. Bioinformatics **34,** 568-575. issn : 1367-4803. https://academic.oup.com/bioinformatics/article/34/4/568/4386917 (2019) (15 fév. 2018).

23. Pandey, P., Bender, M. A., Johnson, R. & Patro, R.
A General-Purpose Counting Filter : Making Every Bit Count in
Proceedings of the 2017 ACM International Conference on Management of Data (ACM, New York, NY, USA, 2017), 775-787. isbn : 978-1-4503-4197-4.
http://doi.acm.org/10.1145/3035918.3035963 (2019).

24. Muggli, M. D. et al. Succinct colored de Bruijn graphs. Bioinformatics **33** (éd. Birol, I.) 3181-3187. issn : 1367-4803, 1460-2059.
https://academic.oup.com/bioinformatics/article/33/20/3181/2995815 (2019) (15 oct. 2017).

25. Liu, X. et al. A novel data structure to support ultra-fast taxonomic classification of metagenomic sequences with k-mer signatures. Bioinformatics **34,** 171-178. issn : 1367-4803.
https://academic.oup.com/bioinformatics/article/34/1/171/3931858 (2019) (1er jan. 2018).

26. Yu, Y., Belazzougui, D., Qian, C. & Zhang, Q. Memory-Efficient and Ultra-Fast Network Lookup and Forwarding Using Othello Hashing. IEEE/ACM Transactions on Networking **26,** 1151-1164. issn : 1063-6692 (juin 2018).

27. Limasset, A., Rizk, G., Chikhi, R. & Peterlongo, P. Fast and scalable minimal perfect hashing for massive key sets. arXiv :1702.03154 [cs]. arXiv : 1702.03154.
http://arxiv.org/abs/1702.03154 (2019) (10 fév. 2017).

28. Solomon, B. & Kingsford, C. Fast search of thousands of short-read sequencing experiments. Nature Biotechnology **34,** 300-302. issn : 1546-1696.
https://www.nature.com/articles/nbt.3442 (2019) (mar. 2016).

29. Hamming, R. W. Error detecting and error correcting codes.
The Bell System Technical Journal **29,** 147-160. issn : 0005-8580 (avr. 1950).

30. Solomon, B. & Kingsford, C. Improved Search of Large Transcriptomic Sequencing Databases Using Split Sequence Bloom Trees. Journal of Computational Biology **25,** 755-765. issn : 1557-8666.
http://www.liebertpub.com/doi/10.1089/cmb.2017.0265 (2019) (juil. 2018).

31. Sun, C., Harris, R. S., Chikhi, R. & Medvedev, P. AllSome Sequence Bloom Trees.
bioRxiv. http://biorxiv.org/lookup/doi/10.1101/090464 (2019) (23 mar. 2017).

32. Harris, R. S. & Medvedev, P. Improved Representation of Sequence Bloom Trees.
bioRxiv. http://biorxiv.org/lookup/doi/10.1101/501452 (2019) (12 fév. 2019).

33. Mäkinen, V. & Navarro, G. Rank and select revisited and extended.
Theoretical Computer Science **387,** 332-347. issn : 03043975.
https://linkinghub.elsevier.com/retrieve/pii/S0304397507005300 (2019) (nov. 2007).

34. Raman, R., Raman, V. & Satti, S. R. Succinct Indexable Dictionaries with Applications to Encoding k-ary Trees, Prefix Sums and Multisets. ACM Transactions on Algorithms **3,** 43-es. ISSN : 15496325. arXiv : 0705.0552. http://arxiv.org/abs/0705.0552 (2019) (1er nov. 2007).

35. Turner, I., Garimella, K. V., Iqbal, Z. & McVean, G. Integrating long-range connectivity information into de Bruijn graphs. Bioinformatics **34,** 2556-2565. ISSN : 1367-4803. https://academic.oup.com/bioinformatics/article/34/15/2556/4938484 (2019) (1er août 2018).

36. soixanteseize. Explore to understand, share to bring about change https://oceans.taraexpeditions.org/en/m/science/news/the-oceanomics-project/ (2019).

37. Tara Oceans Coordinators et al. A global ocean atlas of eukaryotic genes. Nature Communications **9.** ISSN : 2041-1723. http://www.nature.com/articles/s41467-017-02342-1 (2019) (déc. 2018).

38. Villar, E. et al. The Ocean Gene Atlas : exploring the biogeography of plankton genes online. Nucleic Acids Research **46,** W289-W295. ISSN : 0305-1048. https://academic.oup.com/nar/article/46/W1/W289/5000016 (2019) (W1 2 juil. 2018).

39. Delmont, T. O. et al. Nitrogen-fixing populations of Planctomycetes and Proteobacteria are abundant in surface ocean metagenomes. Nature Microbiology **3,** 804. ISSN : 2058-5276. https://www.nature.com/articles/s41564-018-0176-9 (2019) (juil. 2018).

40. Marçais, G. & Kingsford, C. A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. Bioinformatics **27,** 764-770. ISSN : 1367-4803. https://academic.oup.com/bioinformatics/article/27/6/764/234905 (2019) (15 mar. 2011).

41. Benoit, G. et al. Multiple comparative metagenomics using multiset *k* -mer counting. PeerJ Computer Science **2,** e94. ISSN : 2376-5992. https://peerj.com/articles/cs-94 (2019) (14 nov. 2016).

42. Drezen, E. et al. GATB : Genome Assembly & Analysis Tool Box. Bioinformatics **30,** 2959-2961. ISSN : 1367-4803. https://academic.oup.com/bioinformatics/article/30/20/2959/2422199 (2019) (15 oct. 2014).

43. Rizk, G., Lavenier, D. & Chikhi, R. DSK : k-mer counting with very low memory usage. Bioinformatics **29,** 652-653. ISSN : 1367-4803. https://academic.oup.com/bioinformatics/article/29/5/652/253092 (2019) (1er mar. 2013).

44. Eren, A. M. et al. Anvi'o : an advanced analysis and visualization platform for 'omics data. PeerJ **3,** e1319. ISSN : 2167-8359. https://peerj.com/articles/1319 (2019) (8 oct. 2015).