Indexing and analysis of large sequencing collections using k-mer matrices

Téo Lemane

DOCTORAT BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES 1

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par

Téo LEMANE

Indexing and analysis of large sequencing collections using k-mer matrices

Thèse présentée et soutenue à Rennes, le 16 décembre 2022
Unité de recherche : Equipe GenScale, Univ Rennes, Inria, CNRS, IRISA

Rapporteurs avant soutenance :

Daniel GAUTHERET	Professeur des universités, Université Paris-Sud
Eric PELLETIER	Directeur de recherche CEA, Genoscope Evry

Composition du Jury :

Rapporteurs :	Daniel GAUTHERET	Professeur des universités, Université Paris-Sud
	Eric PELLETIER	Directeur de recherche CEA, Genoscope Evry
Examinateurs :	Laurent JACOB	Chargé de recherche CNRS, LBBE Lyon
	Thérèse COMMES	Professeure des universités, Université de Montpellier
	Guillaume RIZK	Principal Software Engineer, Illumina R&D Rennes
Dir. de thèse :	Pierre PETERLONGO	Directeur de recherche Inria, Inria Rennes
Co-dir. de thèse :	Rayan CHIKHI	Chargé de recherche, Institut Pasteur Paris

Remerciements

Tout d'abord, je tiens à remercier Daniel Gautheret et Eric Pelletier d'avoir accepté d'être rapporteurs de mes travaux, ainsi que les membres de mon jury de thèse, Thérèse Commes, Laurent Jacob et Guillaume Rizk. Merci pour l'intérêt que vous portez à mon travail.

Je tiens à exprimer toute ma reconnaissance à Pierre Peterlongo et Rayan Chikhi, mes encadrants de thèse. Merci pour votre confiance et vos précieux conseils. Je suis honoré d'avoir pu faire mes premiers pas dans le monde de la recherche avec vous.

Je souhaite aussi remercier les membres de mon comité de suivi de thèse, Sophie Schbath, Laurent Jacob et Antoine Limasset, qui ont suivi le déroulement de ma thèse et apporté un regard extérieur essentiel.

Merci à l'Université de Rennes 1, l'école doctorale MathSTIC, IRISA et Inria de m'avoir donné la possibilité de réaliser ce travail de thèse. Une pensée particulière pour les personnels administratifs, merci pour votre patience face à ma rigueur administrative légendaire.

Merci à mes collègues de l'équipe Symbiose pour l'environnement de travail exceptionnel, tant sur le plan humain que scientifique. J'espère avoir l'occasion de revenir déambuler dans les couloirs de temps à autre.

Merci à mes collègues de l'institut Pasteur. Je suis très heureux d'avoir pu travailler et échanger avec vous.

Je remercie l'ensemble de mes amis qui m'ont beaucoup soutenu et permis de décompresser pendant ces années.

Enfin, un immense merci à l'ensemble de ma famille, et plus particulièrement mes parents, sans qui rien de tout cela n'aurait été possible. Une pensée spéciale pour mon frère, Clément, avec qui j'ai partagé un toit pendant ces trois longues années.

Indexing and analysis of large sequencing collections using k-mer matrices

The 21st century is bringing a tsunami of data in many fields, especially in bioinformatics. This paradigm shift requires the development of new processing methods capable of scaling up on such data. This work consists mainly in considering massive tera-scaled datasets from genomic sequencing. A common way to process these data is to represent them as a set of words of a fixed size, called k-mers. The k-mers are widely used as building blocks by many sequencing data analysis techniques. The challenge is to be able to represent the k-mers and their abundances in a large number of datasets. One possibility is the k-mer matrix, where each row is a k-mer associated with a vector of abundances and each column corresponds to a sample. Some k-mers are erroneous due to sequencing errors and must be discarded. The usual technique consists in discarding low-abundant k-mers. On complex datasets such as metagenomes, such a filter is not efficient and discards too many k-mers. The holistic view of abundances across samples allowed by the matrix representation also enables a new procedure for error detection on such datasets. In summary, we explore the concept of k-mer matrix and show its scalability in various applications, from indexing to analysis, and propose different tools for this purpose. On the indexing side, our tools have allowed indexing a large metagenomic dataset from the Tara Ocean project while keeping additional k-mers, usually discarded by the classical k-mer filtering technique. The next and important step is to make the index publicly available. On the analysis side, our matrix construction technique enables to speed up a differential k-mer analysis of a state-of-the-art tool by an order of magnitude.

Indexation et analyse de grandes collections de séquençages via des matrices de k-mers

Le 21ème siècle subit un tsunami de données dans de nombreux domaines, notamment en bio-informatique. Ce changement de paradigme nécessite le développement de nouvelles méthodes de traitement capables de passer à l'échelle sur de telles données. Ce travail consiste principalement à considérer des jeux de données massifs provenant du séquençage génomique. Une façon courante de traiter ces données est de les représenter comme un ensemble de mots de taille fixe, appelés k-mers. Les k-mers sont très largement utilisés comme éléments de bases par de nombreuses méthodes d'analyses de données de séquençages. L'enjeu est de pouvoir représenter les k-mers et leurs abondances dans un grand nombre de jeux de données. Une possibilité est la matrice de k-mers, où chaque ligne est un k-mer associé à un vecteur d'abondances. Ces k-mers sont erronées en raison des erreurs de séquençage et doivent être filtrés. La technique habituelle consiste à écarter les k-mers peu abondants. Sur des ensembles de données complexes comme les métagénomes, un tel filtre n'est pas efficace et élimine un trop grand nombre de k-mers. La vision des abondances à travers les échantillons permise par la représentation matricielle permet également une nouvelle procédure de détection des erreurs dans les jeux de données complexes. En résumé, nous explorons le concept de matrice de k-mer et montrons ses capacités en termes de passage à l'échelle au travers de diverses applications, de l'indexation à l'analyse, et proposons différents outils à cette fin. Sur le plan de l'indexation, nos outils ont permis d'indexer un grand ensemble métagénomique du projet Tara Ocean tout en conservant des k-mers rares, habituellement écartés par les techniques de filtrage classiques. En matière d'analyse, notre technique de construction de matrices permet d'accélérer d'un ordre de grandeur l'analyse différentielle de k-mers.

RÉSUMÉ EN FRANÇAIS

Introduction

Le 21ème siècle subit un tsunami de données dans de nombreux domaines, notamment en bio-informatique. Ce changement de paradigme nécessite le développement de nouvelles méthodes de traitement capables de passer à l'échelle sur de telles données. Ce travail consiste principalement à considérer des jeux de données massifs provenant du séquençage génomique. Une façon courante de traiter ces données est de les représenter comme un ensemble de mots de taille fixe, appelés k-mers. Les k-mers sont très largement utilisés comme éléments de bases par de nombreuses méthodes d'analyses de données de séquençages. L'enjeu est de pouvoir représenter les k-mers et leurs abondances dans un grand nombre de jeux de données. La principale difficulté réside dans la taille des échantillons qui peuvent contenir des centaines de milliards de k-mers distincts.

Dans ce travail, nous explorons une représentation particulière des séquences, la matrice de k-mers, et ses capacités à répondre à des problèmes variés dans des contextes d'indexation ou d'analyse.

En résumé, ce travail à conduit à différentes contributions :

1. **Une méthode efficace de construction de matrice de k-mers.** Une matrice de k-mers est construite à partir des vecteurs d'abondances de chacun des k-mers d'un jeu de données. La première étape consiste donc à compter les occurrences des k-mers dans l'ensemble des jeux de données étudiés. Nous avons étendu les techniques de comptage classiques, simple jeu, au comptage multi-jeux permettant ainsi la construction efficace et parallèle de matrice de k-mers à partir de nombreux jeux de séquençages.
2. **Une méthode de filtration des k-mers erronés.** Nous montrons que les techniques classiques de détection des k-mers erronés ne sont pas adaptées à tous types de jeux de données. La représentation matricielle permet un nouveau type de correction basé sur la redondance entre les échantillons.
3. **Une méthode efficace pour la construction parallèle de matrices de filtre de Bloom.** Un filtre de Bloom [1] est une structure de données probabiliste permettant de déterminer si un élément fait partie d'un ensemble. Ce type de structure est très utilisé dans le cadre de l'indexation de données, ici l'indexation de k-mers. La construction de filtres de Bloom à partir de données de séquençage est un goulot d'étranglement majeur des processus d'indexation. Nous proposons une méthode de construction de matrices de filtre de Bloom, basée sur la méthode de construction de matrice d'abondances.

4. **Un ensemble d'outils génériques pour la construction et l'analyse des matrices de k-mers.** Une matrice de k-mers est une représentation générique pouvant être utilisée dans des contextes d'analyses variés. Nous pensons qu'un outil générique permettant la construction et le traitement de ce type d'objet serait bénéfique pour la communauté bio-informatique. En conséquence, nous proposons un outil, `kmtricks`, pour la construction, le streaming et l'indexation des matrices de k-mers. Cet outil est accompagné d'une librairie et d'un système de plugins permettant d'étendre ses fonctionnalités et de construire de nouveaux outils.
5. **L'application des matrices de k-mers dans le cadre de l'analyse différentielle de k-mers.** L'analyse différentielle de k-mers consiste à trouver les k-mers différentiellement représentés entre deux groupes de jeux de données, par exemple des cas et des contrôles. Ces méthodes appliquent des tests statistiques sur chacun des vecteurs d'abondances des k-mers. L'obtention de ces vecteurs d'abondances est actuellement le facteur limitant. Nous avons combiné notre méthode de construction avec des outils statistiques de l'état de l'art pour produire un nouvel outil, `kmdiff`, pour améliorer le passage à l'échelle de ce type de méthodes. Nous montrons que `kmdiff` est plus performant d'un point de vue computationnel tout en produisant des résultats équivalents.
6. **L'application des matrices de k-mers dans le cadre de l'indexation de grands ensembles de données.** Nous avons utilisé notre méthode de construction de filtre de Bloom pour proposer un pipeline d'indexation complet, de l'indexation à la requête. Ce pipeline a permis l'indexation d'un grand ensemble de séquençage métagénomique du projet Tara Ocean. Ce même jeu de données a aussi été utilisé pour évaluer notre méthode de filtration des k-mers erronés.

1 Les matrices de k-mers

Une matrice de k-mers M construite à partir d'un ensemble de jeux de données S est un type de données abstrait représentant les abondances de chacun des k-mers appartenant à S. En d'autres termes, chaque élément $M(i, j)$ représente l'abondance d'un k-mer i dans un jeu de données j. En fonction du contexte, M peut être une matrice binaire où chaque élément $M(i, j)$ est un bit représentant la présence ou l'absence d'un k-mer dans un jeu de données.

Plusieurs outils utilisent déjà ce concept de matrice de k-mers dans leurs analyses. Toutefois, il n'existe pas de méthodes et outils spécialisés dans sa construction et son traitement. Les outils existants utilisent des méthodes plus ou moins naïves basées sur les compteurs de k-mers traditionnels. En conséquence, nous proposons une méthode de construction basée sur les algorithmes de comptage de k-mers sur disque qui reposent sur le paradigme "diviser pour régner". L'idée est de diviser l'espace des k-mers afin de construire parallèlement de nombreuses

sous-matrices. Notre méthode de construction et de streaming a permis de réaliser des analyses différentielles de k-mers à large échelle (voir .3). Une modification de l'algorithme permet de produire une représentation indexable d'une matrice de présence/absence, i.e une matrice de filtre de Bloom. Ceci nous a permis de proposer un pipeline complet d'indexation que nous avons appliqué à un grand ensemble de données métagénomiques (voir .4).

L'un des principaux problèmes liés à l'utilisation des k-mers est celui des erreurs de séquençage qui entraînent un grand nombre de k-mers erronés. Il existe des méthodes simples pour résoudre ce problème et permettre d'éliminer ces k-mers. Cependant, les techniques classiques ne sont pas toujours adaptées aux échantillons de séquençage complexes tels que les métagénomes. Nous décrivons comment la vue holistique des abondances de k-mer dans plusieurs échantillons permet une nouvelle méthode de filtrage des k-mers que nous appelons *k-mer rescue*. Les méthodes classiques consistent à rejeter les k-mers avec une abondance inférieure à un seuil donné. En effet, nous pouvons supposer qu'un k-mer présent une seule fois dans un génome aura une abondance environ équivalente à la couverture de séquençage. Ainsi, les k-mers avec une abondance faible sont probablement des erreurs. Toutefois, il existe des jeux de données pour lesquels la représentation des séquences n'est pas uniforme dans l'échantillon de départ. C'est par exemple le cas des métagénomes qui peuvent contenir un très grand nombre d'organismes, certains étant sous-représentés. En d'autres termes, des k-mers peu abondants peuvent alors correspondre à des séquences sous-représentées. Lorsque l'on considère un ensemble d'échantillons, nous pouvons supposer qu'il existe une certaine redondance entre ces échantillons. Par exemple, le projet Tara Ocean collecte des échantillons marins dans l'océan à des fins de séquençage métagénomique. L'analyse de ces échantillons révèle, comme attendu, des similarités génétiques entre ces échantillons. L'idée est donc d'exploiter la redondance entre les jeux de données pour sauver des éléments rares. Ceci demande d'analyser les vecteurs d'abondances de chacun des k-mers, une information naturellement donnée par les matrices de k-mers. Lorsqu'un k-mer est peu abondant dans un jeu de données, la méthode consiste alors à examiner les abondances dans les autres échantillons pour décider de le conserver ou non, en fonction de paramètres définis par l'utilisateur.

2 `kmtricks`: un outil de construction et d'analyse de matrice de k-mers

Les matrices de k-mers sont des objets génériques pouvant avoir des applications dans de nombreuses analyses bio-informatiques. L'objectif était donc de proposer un outil, `kmtricks`, permettant de construire différents types de matrices de k-mers de manière efficace et simple d'un point de vue utilisateur.

`kmtricks` est composé de différents éléments : **1.** Un pipeline pour une utilisation de bout en

bout. **2.** Un ensemble de modules pour une utilisation étape par étape, les étapes intermédiaires pouvant être intéressantes indépendemment de la construction des matrices. **3.** Une librairie C++ permettant le développement de nouveaux outils basés sur les matrices de k-mers. **4.** Un système de plugin permettant d'étendre ces fonctionnalités.

3 L'analyse différentielle de k-mers à grande échelle

L'analyse différentielle de k-mers permet de détecter les k-mers différentiellement representés entre deux conditions, chacune représentée par plusieurs séquençages. Généralement, le principal goulot d'étranglement correspond au comptage des k-mers dans chacun des jeux de données suivi de la fusion des tables d'abondances pour obtenir les vecteurs d'abondances. En d'autres termes, ces méthodes peuvent être accélérées par une construction efficace d'une matrice de k-mers. Nous avons combiné notre méthode de streaming de matrice avec un modèle statistique de l'état de l'art proposé par `HAWK` [2, 3] et proposé un nouvel outil, `kmdiff`.

Nous avons comparé les performances de `HAWK` et `kmdiff` sur différents jeux de données. D'abord sur un jeu de données bactérien pour s'assurer de l'équivalence des résultats. L'analyse de différentes cohortes de séquençage humains a ensuite permis d'évaluer les performances à plus grande échelle. Les résultats montrent que `kmdiff` est plus efficace que `HAWK`, que ce soit en termes de temps, d'usage mémoire, ou d'usage disque, tout en proposant des résultats équivalents. Par exemple, `kmdiff` permet d'effectuer une analyse différentielle de k-mers sur un ensemble de 80 séquençages humains représentant 2.3 TB de données compressées en 9 heures. Cette même analyse a demandé plus de 6 jours de calculs dans le cas de `HAWK`.

4 L'indexation de k-mers à grande échelle

Les méthodes d'indexation de k-mers reposent généralement sur la construction de structure de données de bases, comme les filtres de Bloom, pour chacun des jeux de données. Ces structures sont ensuite agrégées pour réduire la taille de l'index final tout en accélérant les temps de requêtes. La construction de ces structures de base est toujours le facteur limitant, quelle que soit la méthode. Nous avons utilisé notre méthode de construction de filtres de Bloom pour proposer un pipeline d'indexation complet. Nous avons évalué ce pipeline sur un jeu de données métagénomique du projet Tara Ocean composé de 241 échantillons représentant plus de 6.5 TB de données compressées. Le pipeline s'est montré environ quatre fois plus rapide que des méthodes existantes tout en considérant un nombre de k-mers beaucoup important. En effet, les méthodes de filtration de k-mers classiques rejetaient approximativement neuf fois trop de k-mers pour le jeu de données Tara Ocean, un jeu de données métagénomique complexe. Ainsi, notre méthode considère un plus grand nombre de k-mers et capture donc une plus grande part

de la diversité des données.

Conclusions

Dans ce travail, nous avons exploré la capacité des matrices k-mer à répondre aux problèmes actuels, principalement liés à la croissance des données. Nous avons montré que de telles représentations peuvent être impliquées à différents niveaux du traitement des données de séquençage sans référence tout en permettant de nouvelles opérations telles que le *k-mer rescue*.

Nous proposons des méthodes de construction et de streaming pour différents types de matrices à des fins d'analyse ou d'indexation. Ces capacités de construction et de streaming efficaces nous ont permis de présenter deux applications majeures: **1. Une analyse différentielle de k-mers efficace**. L'analyse différentielle de k-mer ne nécessite pas la représentation complète de la matrice. Par conséquent, le streaming partitionné et parallèle de la matrice a considérablement accéléré une méthode existante. **2. Un pipeline d'indexation de bout en bout appliqué à Tara Ocean**. Notre méthode a permis de construire efficacement l'index de Tara Ocean en considérant plus de k-mers que les autres outils grâce au *k-mer rescue*. De plus, notre implémentation est la première à bénéficier de `findere` [4], un algorithme permettant de réduire les faux positifs lors des requêtes. Il s'agit maintenant de rendre accessible l'index Tara Ocean, un travail en cours en collaboration avec l'équipe de la plateforme Ocean Gene Atlas (OGA). OGA est un service web permettant de requeter les données Tara Ocean assemblées. Notre index permettrait d'étendre ces requêtes à l'ensemble du jeu de données.

Plus généralement, les matrices de k-mers sont des objets génériques permettant différentes analyses. Nous espérons que les outils développés dans le cadre de ces travaux seront bénéfique à la communauté bio-informatique.

Table of Contents

List of acronyms

AMQ	Approximate Membership Query
AMQF	Approximate Membership Query Filter
BIGSI	Bitsliced Genomic Signature Index
BF	Bloom filter
BBF	Blocked Bloom filter
BWT	Burrows-Wheeler Transform
CF	Cuckoo filter
CQF	Counting quotient filter
COBS	Compact Bit-Sliced Signature Index
DBG	De Bruijn graph
FDR	False discovery rate
FGS	First Generation Sequencing
FWER	Family-wise error rate
GWAS	Genome-Wide Association Study
HowDeSBT	HowDe Sequence Bloom Tree
KFF	K-mer File Format
MPHF	Minimal Perfect Hash Function
NGS	Next Generation Sequencing
OGA	Ocean Gene Atlas
ONT	Oxford Nanopore Technology
PacBio	Pacific Biosciences
PCA	Principal Components Analysis
pBF	partitioned Bloom filter
QF	Quotient filter
RSQF	Rank-and-Select based Quotient Filter
SBT	Sequence Bloom Tree
SBT-ALSO	AllSome Sequence Bloom Tree

SRA	Sequence Read Archive
SSBT	Split Sequence Bloom Tree
SIMD	Single Instruction Multiple Data
SPSS	Spectrum-preversing string set
SNP	Single-Nucleotide polymorphism
TGCC	Très Grand Centre de Calcul du CEA
TGS	Third Generation Sequencing

List of Figures

List of Tables

INTRODUCTION

1 Biological data: a sequence point of view

The evolution of research in biology has led to the production of an immense amount of biological data in the last decades. At the same time, bioinformatics has emerged to analyze this mass of data. Nowadays, digital methods form an integral part of the biological research and enable large-scale studies.

An important part of biological data concerns biological sequences, i.e. nucleotide or protein sequences. In this work, we focus only on DNA sequence data, obtained from sequencing.

A Sequencing

Over time, the techniques for reading genome sequences have evolved and there are now many different types of sequencing methods. The different techniques produce data with particular characteristics that must be considered in the downstream analysis methods. The following sections give a brief overview of main types of methods and their specificities.

1 First generation sequencing

The first sequencing technology was developed by Frederick Sanger and published in 1977 [8]. It allowed to produce the first genetic sequence of the human mitochondrial DNA [9]. Later, it is also this technique that led to the first complete sequence of the human genome in 2001 [10, 11].

In a few words, the process consists in the synthesis of a DNA sequence from a template and a primer. The reaction medium contains also a low concentration of dideoxynucleotides (ddNTPs: ddATP, ddCTP, ddGTP, ddTTP) which are in charge of stopping the elongation. These blocks are randomly used by the DNA polymerase in place of the usual deoxyribonucleotide triphosphates. Thus, the reaction produces a lot of sequences of various sizes that start at the primer and stop at a given ddNTP. The ddNTP of a fragment being known, the sequence can then be reconstructed by ordering all fragments by length using electrophoresis.

Although its throughput is very limited, the Sanger method is still used today for its high accuracy [12] to complete regions that are difficult to sequence with modern technologies [13]. The Sanger method is presented here for completeness but is not related to this work which concerns high-throughput sequencing.

2 Next generation sequencing

The Next Generation Sequencing (NGS) corresponds to the methods developed in the 2000s and has resulted in a major change in the use of sequencing due to unprecedented throughputs and costs. From a methodological point of view, the main novelty is the possibility to perform reactions in parallel on a solid surface [14]. Billions of reactions occur and are analyzed in parallel allowing to sequence a lot of small DNA fragments at the same time. The preparation of the sample consists of the fragmentation of the DNA molecules followed by amplification. After these preparation stages, the sequencing method depends on the platform.

Nowadays many technologies with different capacities are able to produce short reads such as Illumina sequencing [15] or Ion Torrent sequencing [16]. It is generally admitted that these methods have a high-throughput while keeping low error rates [17].

The main drawback is the limited size of the fragments which rarely exceed 250bp. The small size causes difficulties in assembling repeated regions or in detecting long structural variants which require long-range information. This information was introduced with paired-end sequencing which allows sequencing of both ends of a DNA fragment. These types of reads have enabled more efficient mapping resulting in better prediction of genetic variations [18]. Recently, a new technology, 10x Genomics [19], has made it possible to preserve long-range information. The DNA fragments from a single longer molecule (dozen of kilobases) are marked with a DNA barcode. During downstream analyses, it is therefore possible to track reads which belong to the same molecule. This information allows to guide certain analyses like assembly [20] or variant detection [21].

3 Third generation sequencing

The Third Generation Sequencing (TGS) were developed at the end of the 2000s and try to address some NGS issues [22]. The main novelty is the direct sequencing of a single DNA molecule while NGS relies on the amplification of many short fragments. Direct sequencing allows to process molecules of several kilobases and produce thus long reads up to hundreds of kilobases. However, it presents much higher error rates than NGS. The TGS technologies are led by two principal companies, Pacific Biosciences (PacBio) and Oxford Nanopore Technology (ONT).

The ONT platform is based on nano-scaled pores. The variations of the electrical signal when DNA molecules pass through such pores are measured to identify the nucleotides that compose the sequence. Although this method allows long molecule sequencing, it is constantly evolving and still has a high error rate, usually greater than 10% [23]. In addition, the ONT techniques seems to present various error patterns and biais [24]. For example, homopolymers (sequences of identical nucleotides) are difficult to sequence using such a technique due to a non-constant flow in the pore. The number of identical nucleotides passing through the pore in a given time is sometimes difficult to determine and leads to sequencing errors.

The PacBio platforms consist in synthesizing and sequencing single molecules in real-time in a nanowell [25]. A DNA template is used to synthesize a new molecule using fluorescent nucleotides. These nucleotides produce light signals which are interpreted in real-time to produce the sequence. Two protocols are available. The Continuous Long Reads (CLR) protocol produces reads up to 100 kb with relatively high error rates of 8-15% [26]. The Circular Consensus Sequence (CCS) protocol allows multiple readings of the sequence to output a consensus. This technology is currently used to produce High-Fidelity (HiFi) reads with an accuracy higher than 99% [27]. These reads are however smaller with sizes ranging from 10 to 30 kb [27].

As described before, TGS techniques address certain problems of short reads. However, they produce noisier reads at a higher cost. Recent technologies such as HiFi are beginning to address this issue but the costs are still high and few datasets are available. TGS methods are usually seen as new possibilities with their advantages and disadvantages rather than as a replacement for short reads. Moreover, short and long reads are frequently used in conjunction in various operations as assembly [28] or error correction [29].

4 From the k-mers point of view

The characteristics of the different sequencing techniques affect the downstream analysis methods. Indeed, the length of fragments and the error rate prevent the use of a certain type of methods. As a result, the majority of tools focus on one sequencing technology.

In this work, we focus on k-mer-based methods. Given a biological sequence S of length L. A k-mer of S denotes a substring of S of length k, usually $20 \leq k \leq 40$. k-mers are generic objects that can be derived from arbitrary sequences. A sequence or a set of sequences, e.g. a set of sequencing reads, is then represented by its k-mer set. Such representation was primarily used for genome assembly to represent reads as a de Bruijn graph [30], before searching a path representing the complete genome in this graph.

The k-mer-based methods are intensively used for the analysis of short reads from the second sequencing generation. On the contrary, they are less universal in the context of the analysis of long reads from the third generation sequencing. Indeed, a high error rate leads to a large number of erroneous k-mers. For example, assuming that errors are uniformly distributed, all k-mers with a size greater than 10 would be wrong with an error rate of 10%. Moreover, the use of k-mers causes the loss of the long-range information provided by the third generation. However, k-mer-based methods will probably appear in the analysis of long reads with the emergence of accurate long reads as HiFi reads.

B Analysis methods

1 Reference-based

Reference-based methods usually share a common first step: aligning the reads from a sequencing sample on a reference genome. Reference genomes are representative sequences of the genome of a species. They are produced from multiple sources of DNA belonging to various individuals and therefore constitute a blueprint of the genome of a species rather than a natural genome. This first particularity leads to a well-known and studied problem, the reference bias. The reference bias refers to the bias induced by the linear reference genome, which does not account for population or individual genetic variations. Methods that use a reference are therefore subject to this bias, regardless of the quality of the reference. In other words, the results of an analysis depend on the reference, for example, in the case of variant detection which uses the alignments of reads on a reference to identify genetic variations [31, 32, 33]. The choice of reference alleles also implies many biases. Indeed, some populations are poorly represented in the reference, making the analysis difficult [34].

Recently, new methods have proposed to use multiple references to limit these biases. In this case, multiple references are often represented as a sequence graph and the reads are then aligned on such graphs [35, 36].

Whether we consider one or several references, the alignment of the reads is always the first step of the analysis, after the possible pre-processing steps, such as error correction. Local sequence alignment is a costly problem often addressed by heuristics to keep computation time acceptable. Many alignment tools exist and implement different heuristics that produce different results [37, 38, 39, 40]. Consequently, the choice of the alignment method significantly impacts the results.

Finally, some experiments simply cannot use a reference. As previously discussed, the reference strongly impacts the quality of the downstream analysis and must therefore be as accurate as possible. Unfortunately, non-model species rarely have reference genomes of sufficient quality or no reference genome at all. Multi-organism datasets such as metagenomes are also poorly suited to the reference methods because they contain many organisms of different species, sometimes hundreds of thousands.

2 Reference-free

Reference-free methods focus on the processing of the reads without a reference genome. In contrast to reference-based methods, the reference-free paradigm is suitable for any study model since it generally requires only reads without any prior knowledge as reference genomes. Especially, non-model species or complex datasets as metagenomes benefit from these approaches. The reference-free analyses are mainly based on k-mer decomposition and have applications in many

fields such as assembly [41], variants detection [42], error correction [43], metagenomics [44], RNA-Seq quantification [45], etc.

Reference-free methods are also particularly interesting for the analysis of multi-sample datasets. Indeed, they enable the direct comparison of samples while reference-based consit in performing single analyses against the reference followed by the comparison of the results. Such techniques have already demonstrated their ability to produce new results. For example, the joint reference-free analysis of sample collections in the context of variant detection has allowed the detection of sequences missed by reference-based analysis because such sequences were out of the scope of the reference [2, 46]. Usually, reference-free methods require fewer computational resources but sometimes the requirements are still high, as for genome assembly.

Note that a reference is sometimes used at the end of an analysis to locate a genetic variant for example, but is not directly used by the primary analysis.

2 Challenges

A The continuous growth of data

The reduction in costs and the ever-increasing throughput of sequencing technologies are leading to the continuous growth of databases. This tsunami of data concerns many disciplines and is not exclusive to bioinformatics. In sequence bioinformatics, the Sequence Read Archive (SRA) database [47] is a good indicator of the situation since it tries to capture the major part of the sequencing data produced. Figure 1 [page 6] shows the growth of this database and gives an idea of what to expect in the next few years.

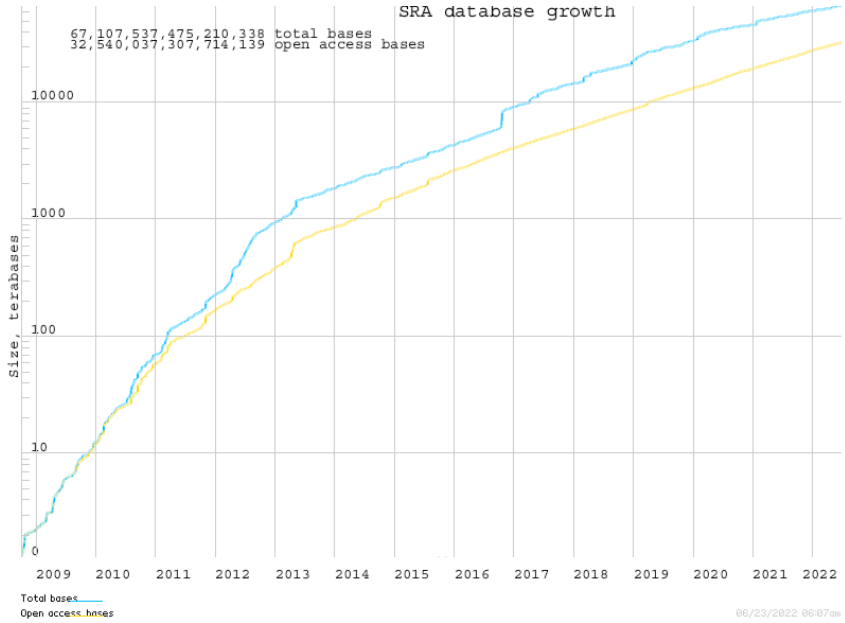

Figure 1 – Available bases in SRA, reprinted from https://www.ncbi.nlm.nih.gov/sra/docs/sragrowth/. The number of bases presently (2022) stored in SRA is 67 petabases, including 32 open access petabases.

This amount of data raises different problems at several levels, storage, accessibility, and analysis, which are all directly related to the work presented in this thesis.

Storage. Because of the mass of data, the representation and storage of the data are critical. All sequences submitted to SRA must contain the quality score per base, regardless of the format. Many downstream analyses does not require quality scores, representing a non-negligible part of the data. SRA addresses this problem with another format that does not contain these scores, SRA Lite, which can be requested to facilitate data transfer.

Note that the data stored on SRA are stored in a raw format by discarding the original format, e.g FASTQ, and dispatching the data in several data tables that contain distilled data. As a result, the SRA and SRA Lite formats are only distribution formats generated on the fly. This makes downloads from SRA tedious with very low bandwidths, usually far from the saturation of the I/O operations. In addition, the SRA format is supported only by a few tools and therefore requires conversion to underlying formats, adding extra overheads.

Efficient read compression methods, with different trade-offs, exist but are not currently used at database scale [48, 49, 50, 51]. It is important to note that some of this data is intended for further analysis. In other words, the representation/compression of these files must be standardized, stable, easily available on every machine, and especially fit the requirements of the database. Some projects directly provide FASTA/Q files in gzip format, which is a generic compression method widely used in many fields, including bioinformatics. The download of these

files is significantly faster and many tools natively support them.

The file representation is a critical aspect from the point of view of both storage and downstream analyses. Due to a large amount of data, many experiments are nowadays limited by IO operations. The current paradigm offers mainly raw data but some projects make available other types of pre-processed data such as alignment files, i.e. the read alignments on the reference genome. Some pre-processing steps are common to many analyses and repeated many times by many end-users for the same dataset. The question of the availability of this kind of data, alignments or k-mer sets for example, leads to many discussions within the community. Indeed, the availability of this data would probably allow saving many hours of computation while requiring consequent additional storage and infrastructures.

Accessibility. From a database point of view, accessibility can be understood as removing barriers that prevent optimal use of the data. In the context of bioinformatics, and more specifically sequencing data, the most important barrier is the sequence search. Nowadays, the samples are associated with various types of metadata, allowing the search of samples according to various parameters such as sequencing platforms, sequencing library, etc. However, the metadata are often very technical or simply missing [52], not bringing much value to the primary data. Other informations like taxonomic informations or experimental conditions would increase the chances that a sample will be reused by someone else outside the original project scope.

In the meantime, metadata queries do not answer a fundamental question: "In which sequencing samples occurs a sequence of interest?". In other words, sequencing samples cannot be selected according to their sequence content. The query of a sequence requires an upstream selection of a sample. Reads indexing techniques have been developed in the last decades and are progressing rapidly, but their capacities are currently quite limited regarding the size of databases. The indexing of collections of sample is nevertheless interesting at smaller scale, as discussed in section .2.B [page 8]. The indexing of sequencing collections constituted an important part of my work and is treated in chapters IV [page 61] and V [page 81].

In summary, the dream would be an alignment-free tool that associates sequences with a collection of sequencing samples. Of course, such a tool is currently unrealistic due the petascale databases. However, there are numbers of intermediate steps between no indexing and a global index. Indexing and querying subsets of data according to a well-designed nomenclature would be a big step forward. Indeed, searching for a subset of sequence samples corresponding to a specific species or a specific sequencing project is obviously attractive.

Analysis. The growth of the data also requires the development of new analysis methods. Indeed the single-sample techniques cannot continuously be transposed to multi-samples, mainly because of the scaling problem. In addition, joint analyses of vast amounts of samples could bring new knowledge, as already allowed by the GWAS studies which identify genes associated with particular traits from the analysis of a large number of samples. Some projects have already

demonstrated the importance of such approaches, both in terms of scaling and results qualities, as presented in the section .2.B [page 8].

The arrival of cloud computing will lead to significant changes in how data is distributed and processed, and bioinformatics is not an exception. Indeed, SRA data is now available on several Amazon S3 providers. This paradigm shift increases the accessibility of the data and has recently allowed a large-scale study [53] on a large part of available sequencing data in SRA. Sequence alignment of RNA polymerase sequences was performed against 5.7 million samples representing 10.2 petabases of raw data. The study, the first at this scale, identified 10^5 novel RNA viruses, expanding the number of known species by an order of magnitude. Such cloud-based experiments will probably become more common in the future simply because downloading data from a central repository is becoming more and more complicated due to the size of the data.

B Making the raw data talk

At the database level, read indexing can be seen as another approach to find interesting samples, alongside the metadata. For example, we can envisage to query a relevant sequence to collect specific samples for further analyses. In addition, read indexing can contribute directly to data analysis, at least on a medium scale as domain-specific projects like 1000 Genome Project [54] or the Human Microbiome Project [55].

Sequencing samples indexing actually index k-mers, i.e. they associate a k-mer with a list of samples that contain it. Queries on such indexes are usually used as an approximation of a sequence alignment. The proportions of k-mers shared between a query and the samples in the index give this approximation. As a result, indexing has already been used to answer, often more efficiently, questions that were previously addressed by alignment-based methods, as illustrated below.

RNA-Seq analysis[56]. The k-mer indexing was used to analyze expressed isoforms on a collection of 2652 human RNA-Seq samples using a new k-mer index layout called Sequence Bloom Tree (SBT). The computation was compared to two mapping algorithm `SRA-BLAST` [57] and `STAR` [58]. The query time of single transcript was 20 min for SBT, and estimated to 2.2 and 921 days for `SRA-BLAST` and `STAR`, respectively. In addition, the pipeline outputs comparable results to the state-of-the-art isoform quantification methods.

Comparative analysis of hundreds of genomes [59]. `BlastFrost` is an index and a query system based on `Bifrost` [60], a compacted representation of De Bruijn graph (DBG). Such an index was used for indexing hundreds of bacterial genomes to perform queries of antimicrobial resistance genes. The results on precision and sensitivity were compared to `Blast` [61] and show a high correlation. Regarding query time, `Blast` remains faster for single query but grows rapidly with the number of queries. Regarding `BlastFrost`, the number of queries has a

small impact on the time since the query time is mostly dependent on the size of the index. In addition, the results show the superiority of `BlastFrost` over time because it is updatable. Indeed, the Blast index must be recomputed at each addition of new sequences while the `BlastFrost` index supports updates. Although `Blast` is faster for a single round of indexing and querying, `BlastFrost` becomes more interesting for maintaining a durable index and supporting multiple rounds of queries. The `BlastFrost` index is also an order of magnitude smaller than the Blast index.

Alignment-free phylogeny [62]. Inferring a phylogeny is an expensive task that is sometimes challenging when reference genomes are unavailable, or the number of input genomes is too large to perform all pairwise alignments. `SANS` is an alignment-free and reference-free method allowing to infer a phylogeny from various inputs, i.e. genomes or reads. It is based on a `Bifrost index` and infers edges of the phylogenetic tree using common subsequences extracted from the DBG and therefore not relies on pairwise comparisons.

These examples demonstrate that read indexing can help solve problems previouslt addressed with expensive alignment-based methods. In addition, they show the importance of project-scale indexing for analysis purposes. Database-scale indexing is still an ideal goal but is not yet achievable with current methods and tools. Indeed, even if the methods are more and more efficient, indexing databases like SRA requires a drop in computation times of several orders of magnitude.

Note that out of the scope of indexing, raw data can also be used for analysis without any assembly or alignment to reference genomes. Reference-free techniques are especially relevant for analyzing large datasets for which reference-based methods may not scale up. For example, the comparisons of sequencing samples based on k-mers has been used in comparative metagenomics [63, 64] or in GWAS-like studies [2, 3, 46]. The last point is related to this work and will be discussed later.

3 Outline

In this work, we explore the k-mer matrix representation with respect to the issues discussed in the previous sections. A k-mer matrix is a simple representation that associates k-mers to abundances across multiple input samples. We are interested in the construction and the use of such objects in various contexts, from indexing to analysis of sequencing samples.

In summary, this work leads to the following contributions:

1. **An efficient method for parallel construction and streaming of a k-mer matrix.** The construction of a k-mer matrix requires the abundances of each$k$-mer across multiple samples. We extend the disk-based k-mer counting techniques to joint multi-sample k-mer counting. An in-depth presentation of the structure and the construction procedures

are presented in chapter II [page 41].

2. **A new method for k-mer filtering.** Sequencing errors lead to erroneous k-mer that must be discarded. We show that the usual filtering method, consisting in discarding k-mers with an abundance less than a threshold, is not usable for all types of datasets. We propose a new method enabled by the matrix representation. The procedure is described in chapter II [page 41], and its application and evaluation on a real dataset are presented in chapter IV [page 61].
3. **An efficient method for parallel construction of Bloom filters matrices for indexing purposes.** A Bloom filter is a data structure allowing to represent succintly collections of items. It is widely used in the context of indexing, especially for k-mer indexing. We extend the k-mer matrix construction algorithm to achieve parallel construction of Bloom filters matrices in low-memory from a collection of samples. Such a matrix is a building block that can be used for large-scale indexing. The extended method is discussed in chapter IV [page 61].
4. **A k-mer matrix framework.** A k-mer matrix is a generic representation that could be useful in various contexts, some of which are presented in this manuscript. We believe that a generic tool that works with such representation would benefit for the bioinformatic community. Consequently, we propose a tool, `kmtricks`, for the construction, streaming, and indexing of k-mer matrices. It is accompanied by an API and a plugin system to extend its features and build new tools. The chapter V [page 81] is dedicated to the presentation of the components of the tool and the implementation details.
5. **An application in large-scale differential k-mer analysis.** The differential k-mer analysis consists in finding the differentially represented k-mers between two groups of samples such as cases and controls in a disease study. We combine our method of k-mer matrix construction with state-of-the-art statistical models to produce a new tool, `kmdiff`, allowing large-scale differential k-mers analysis. We show that this strategy leads to better performances while providing equivalent results as state-of-the-art tools. The pipeline and results on various datasets are presented in chapter III [page 51].
6. **An application in large-scale indexing.** We use our Bloom filters matrix construction to accelerate a state-of-the-art indexing method and to provide an end-to-end indexing pipeline, i.e. from indexing to query. We present our method scaling capabilities and its application on a real and complex collection of samples in chapter IV [page 61].

The next chapter presents an overview of state-of-the-art k-mer-based indexing and analysis methods and argues on the direction of our work.

Chapter I

State of the art

Preamble

This chapter presents the current paradigms of k-mer-based sequencing indexing and analysis. Obviously, such techniques rely on k-mer counting, also introduced here and of significant importance in this work.

I start by presenting generic indexing techniques through the prism of k-mers to reflect their advantages and disadvantages in our specific case, namely k-mer-based sequencing samples indexing. This overview of k-mer indexing also argues the choices made in this work.

On the analysis side, I subsequently present the concept of differential k-mer analysis as well as the current methods capable of achieving it. Our goal was to upscale such techniques by improving the handling of k-mers.

Finally, I present succinctly both k-mer counting paradigms, in-memory counting, and disk-based counting. More details are provided along the manuscript since some of the work derives from these methods.

Contents

1 Sequence indexing

A The need for succinctness

In this section, I discuss the size of the data and the necessity of appropriate data structures to index and analyze them, taking the human genome as an example.

Biological sequences can correspond to different biological entities like nucleic or amino acids with sometimes virtual characters representing, for example, uncertain characters or even combinations of characters. In this document, we consider only nucleic sequences without consideration of "N" or IUPAC characters [65]. In other words, a sequence is over the alphabet $\Sigma = \{A, C, G, T\}$.

DNA strands being inversely complementary, a first possible solution to reduce the space is to represent only one strand. Two paradigms coexist: the first one is commonly used for long sequences such as assemblies and represents the sequence from 5' end to 3' end. The second one is used for short sequences and represents only the smallest string, in the lexicographic sense, between a sequence and its reverse complement. The smallest sequence is called the canonical form. Both representations avoid to multiply by two the required space. Using this technique, a human genome in plain text format requires 3.2 gigabytes (GB).

The small size of the alphabet allows using a more specific encoding than ASCII, which requires 8 bits per character. The number of bits required to encode an alphabet of size $|\Sigma|$ is $\lceil log_2(|\Sigma|) \rceil$. Thus a DNA character can be represented using only 2 bits. The most common encoding is $\{A = 0,\ C = 1,\ G = 2,\ T = 3\}$. However, another encoding scheme $\{A = 0,\ C = 1,\ G = 3,\ T = 2\}$ is sometimes preferred because it corresponds to the 2^{nd} and the 3^{th} bits in the ASCII representation of nucleotides. The encoding operations are then only two fast bitwise operations without any memory access. It is the encoding used in this work. Note that it changes the canonical form because G is greater than T in this case. Using the 2-bit encoding, the storage of a human genome requires 0.8GB.

A commonly used representation in biological sequence analysis and indexing is the k-mer set. Given a sequence S, its k-mer set $K(S)$ is the set of all overlapping sub-words of size k from S. Considering $k = 32$, the number of bytes required to represent a k-mer using the 2-bit encoding is 8 (64 bits). Assuming unique k-mers, the cardinality of a k-mer set from a genome of size N is approximately $N - k + 1$, which gives $3.2e9$ for a human genome. The space required by the whole set is then 95.3GB in plain text and 23.8GB using the 2-bit encoding. In the context of k-mer counting, k-mer abundances are also considered. If k-mer abundances stored using 4 bytes per abundance, the space reaches 35.8GB. In practice, k-mer sets are often computed on sequencing samples that contain sequencing errors resulting in an even larger number of k-mers. For example, the plain text representation of the k-mer set of human short-read sequencing (from https://www.ncbi.nlm.nih.gov/sra/ERX009609) with 5.7 billion of k-mers uses 192GB.

However, k-mers come from a set of sequences; therefore, overlaps are expected. Some methods [66, 67] take advantage of overlaps to build compacted sequences that correspond to maximal unique paths in k-mers graph, where nodes are k-mers and edges correspond to $k-1$ overlaps between k-mers. This representation is called a Spectrum-preversing string set (SPSS). As a side note, recently Schmidt et al. shows that an optimal SPSS can be computed in linear time. These representations are nevertheless costly to compute (See Section VI.2 [page 106]) because they need a pre-existing navigational k-mer index [69].

All techniques presented here can be considered as plain text representation (despite the encoding) without any compression or tricky indexing techniques. For a single dataset, plain text representations seem usable, but the data become intractable on a larger scale with hundreds or thousands of samples. As a result, efficient and succinct data structures are needed to survive the never-ending growth of data.

B Query

Before describing the indexing methods allowing to perform sequence queries in a large set, it is important to understand what we call a query.

Different methods exist to determine the membership of a sequence to another sequence or a set of sequences. The most accurates are the alignment techniques which perform base-level alignments, but they suffer from some limitations. Their computational costs allow them to be used on a genome or a limited set of genomes but make them difficult to apply to extensive collections of sequences, especially on collections of reads. Beyond the computational costs, mapping is also hardly applicable to unassembled sequences. In this case, the database is a collection of short reads, making difficult the searching of sequences larger than reads, such as relevant sequences like genes.

In this section, I present the k-mer query paradigm allowing to perform membership queries between a sequence and a collection of samples.

Given a collection of samples $S = \{S_0, \ldots, S_n\}$, the purpose is to determine the set $S_i \in S$ that contains a query sequence Q. A common method is to compute the intersection between the k-mers from Q, denoted $K(Q)$, and each $K(S_i)$. Note that this technique remains an approximation. Indeed, if Q is present in a sample S_i, then $K(Q) \subset K(S_i)$, but finding all k-mers of an arbitrary query in a sample does not imply that all k-mers originally came from the same sequence. However, this is usually considered a suitable approximation when k is sufficiently large, usually $k \geq 20$. It is therefore reasonable to consider that $Q \in S_i$ if $|K(Q) \cap K(S_i)| = |K(Q)|$, i.e. all k-mers from Q exist in S_i.

Sequencing errors and sequence variations impact the k-mers set since a single nucleotide variation can affect up to k k-mers, drastically reducing the number of common k-mers between a query and a sequence. Using a less strict approach, it is possible to overcome this problem by

considering "partial matches". Thus $Q \in S_i$ if $\frac{|K(Q) \cap K(S_i)|}{|K(Q)|} \geq \theta$ where θ is a user-defined threshold. Partial matches also allow faster queries because the operation can be canceled immediately when the threshold is reached or is no longer reachable.

The most common way to implement this query model is to use a membership data structure to perform k-mer membership queries and compute k-mer intersections between the query and the sample. Indexing of multiple samples is performed in the same way. An index, usually composed of several membership data structures, is used to associate a given k-mer to the list of samples containing it (illustrated in Figure I.1 [page 14]).

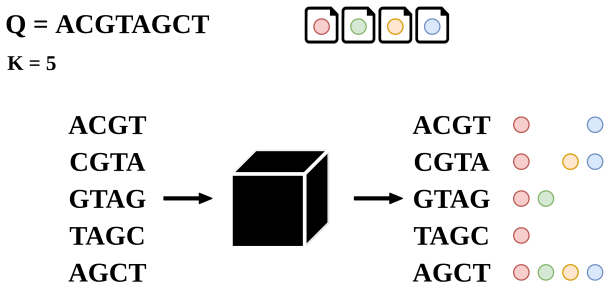

Figure I.1 – Overview of the membership query model. The black-box is an arbitrary index allowing k-mer queries against a collection of 4 samples. The query Q is split into k-mers, using $k = 5$. Each k-mer is queried and the index returns a list of samples for each one. The membership of Q in each sample is then determined accordingly to θ, a user-defined threshold denoting the minimal proportion of shared k-mers between a query and a sample to consider a match.

C Assembled sequences indexing

Genomes and reads data differ on several points. In assembled sequences, all bases are expected to be valid bases, unlike reads which contain various types of errors. These errors involve pre-processing operations like k-mer counting (see Section I.3 [page 36]) to be filtered out. Moreover, splitting the genome into k-mers leads to the loss of long-range information. As a result, genome indexing techniques often use string-based techniques such as FM-index [70, 71] which allows keeping the long-range informations, unlike k-mer-based techniques. Sequence mappers widely use FM-index to locate alignment seeds before performing the alignment [72].

MinHash-based techniques are also intensively used in the context of indexing or comparing genomes [73, 74]. MinHash [75] is a locality sensitive hashing scheme [76] allowing to approximate the Jaccard similarity coefficient [77] in a time and space-efficient way. Unfortunately, these methods are not suitable for fragmented data like reads, except for non-noisy long reads.

D Indexing sequencing collections

1 Core data structures

In this section, I present the two main types of data structures allowing to index a set of k-mers, exact and approximate. I argue that although exact representations are commonly used in many applications like assembly, for example, they are not very suitable for large and multi-samples indexing.

a Exact representations

Beyond classical methods like hash tables, overlapping and complementarity properties of k-mers allow to build domain-specific representation, thus saving space and time. Two paradigms coexist, hash and string based, and propose different space and time trade-offs.

Methods differ significantly in terms of their memory usage as well as their construction time and query complexity. Some allow the reconstitution of the original k-mer set, while others allow only membership queries without any possibility of restoring the original set. In both cases, the abundances of k-mers can be stored.

Many k-mer dictionaries exist [78, 79, 80, 81] and are used in various applications. Although these types of representations are space-efficient and allow fast queries, the construction is often expensive. They are therefore rather used in the context of a single sample, for example, in assembly tools [82, 83, 84, 85] or for indexing highly similar genomes [59] such as genomes from the same species. For the indexing of many sets of reads, approximate methods are often preferred and are presented in the next section.

Note that a recent and unpublished method, Metagraph [86], shows promising results on large-scale and multi-sample indexing in the context of uniform datasets sharing a large part of their k-mers such as a collection of samples from the same species. As for many other techniques, the efficiency is less significant on high-diversity datasets as collections of metagenomic sequencings with rather long construction and larger indexes.

b Approximate Membership Query Filters

Approximate Membership Query Filters (AMQFs) are probabilistic data structures designed to address the membership problem in a space-efficient way. This problem can be defined as follows: Given two sets $\mathcal{U}$ and $\mathcal{V}$, determine for each element $u \in \mathcal{V}$ if $u \in \mathcal{U}$. In other words, V is a set of queries. An AMQF allows to answer this question approximately, i.e. it addresses the membership of an element $u \in \mathcal{V}$ to a set $\mathcal{U}$ with a non-zero false positive rate of ϵ. In other words, the possible answers are “definitively not in set” and “possibly in set”.

These structures have a lot of applications in various fields and are now ubiquitous in bioinformatics in response to the endless growth of data [56, 87, 88, 89]. Indeed, the space required

by AMQF is relatively small. The information theoretical bound requires $log_2(\frac{1}{\epsilon})$ bits per item with ϵ the false positive rate. In this section, I present three AMQFs with a focus on Bloom filters which are central in this work.

i Bloom

Bloom filter (BF) [1] is probabilistic data structure that allows two operations: `insert` and `lookup`. It is a bit array $B[0..n)$, initialized to 0, and k hash functions $h_i : \mathcal{U} \to \{0, ..., n-1\}\ \forall i \in [1..k]$. `Insert` can be defined as follows $B[h_i(x)] \leftarrow 1, \forall i \in [1..k]$ and `lookup` as $\bigwedge_{i=1}^{k} B[h_i(x)]$. As both operations depend only on k, the time complexity is $O(k)$. An example of a BF construction is shown in figure I.2 [page 16].

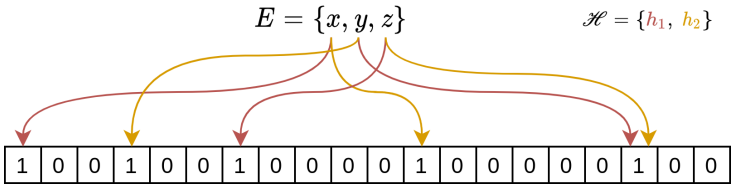

Figure I.2 – Example of a BF using 20 bits and $k = 2$ hash functions. $h_1(y)$ and $h_2(z)$ collides.

Like all AMQFs, BF is exposed to false positives. Assuming bit independence, the probability that the value of a bit is 0 after the insertion of m elements is $(1 - \frac{1}{n})^{km}$. The false positive rate is then

$$\epsilon = (1 - (1 - \frac{1}{n})^{km})^k \approx (1 - e^{km/n})^k \tag{I.1}$$

An optimal BF uses $1.44log_2(\frac{1}{\epsilon})$ bits per item [1]. Compared to the information theoretical bound (see Section I.1.D.1.b [page 15]), the overhead is about 44%. In this context, the optimal number of hash functions is $k = log_2(\frac{1}{\epsilon})$ where ϵ is the false positive rate. There is thus an inverse relation between ϵ and k: when ϵ decreases k increases. In other words, positive queries must probe more bits when the false positive rate is low. A high number of hash functions then leads to a higher number of cache misses for both `insert` and `lookup`. Consequently, the use of an optimal number of hash functions (for a given ϵ) increases the query time.

However, the time complexities of the `insert` and `query` operations are constant regardless of the load factor, i.e. whatever the number of elements already in the filter. As previously stated, both operations depend only on k, the number of hash functions. In contrast, other AMQFs use addressing schemes to resolve collisions during insertions and the efficiency is thus related to the load factor (see Section I.1.D.1.b.ii [page 17]).

BFs are therefore exposed to the problem of a poor locality of reference [90], i.e. inserting and querying require looking at non-contiguous memory areas. As a result, inserting or querying an element leads to approximately k cache misses. The Blocked Bloom filter (BBF) variant [91]

partially addresses this problem by ensuring that the k hash values of an element reside in a 512-bits block of memory and fit in a single CPU cache-line of 64 bytes. This scheme offers at the same time really fast `insert` and `lookup` through Single Instruction Multiple Data (SIMD) allowing to perform each operation in a single CPU operation with the recent instruction sets [92]. The locality problem remains partially solved. The insertion of n elements always leads to a large number of cache misses, approximatively n in total. A similar number of cache misses can be reached in a lower space (for a given ϵ) using a classical BF with a single hash function. However, as described before, a single hash function is synonymous with a high false positive rate. For example, given a BF of size 1000 containing 100 elements, using 7 hash functions instead of one reduces the false positive rate by an order of magnitude.

In the context of sequencing samples indexing, we handle a collection of BFs, one per sample. In this case, the data locality becomes more interesting: Given several BFs of the same size using the same hash functions, the insertion pattern of a given element is the same, whatever the filter. AMQFs that use addressing scheme does not have this property since an insertion depends on the current state of the structure. This leads to some advantages for BFs in terms of space and time which are described in the sections I.1.D.1.b.iv [page 21], I.1.D.2.b.ii [page 26] and I.1.D.2.b.i [page 24].

ii Cuckoo

Cuckoo hashing [93] is an open addressing technique for resolving collisions in hash tables. A cuckoo hash table uses two tables T_1 and T_2 each with n buckets. Two independent hash functions h_1 and h_2 are associated to each table and map a universe U to bucket locations $\{0, ..., n-1\}$. A key k can be stored in only two locations $T_1[h_1(k)]$ and $T_2[h_2(k)]$. The `lookup` operation is then $lookup(k) = T_1[h_1(k)] == x \vee T_2[h_2(k)] == x$. The `delete` operation is performed in the same way by checking the two possible positions. The `insert` operation derives from the behavior of the cuckoo chick bird, which pushes the other eggs out of the nest when it hatches. Similarly, a new key may push an older key to a new location when collisions occur. To insert a key k, the first bucket $T_1[h_1(k)]$ is examined. If the bucket is free, the key is inserted; otherwise the preoccupied key k_{old} is removed before inserting k. An insertion attempt is then performed for k_{old} in T_2 by following the same procedure. The process continues until an empty position is found. Since there is not always a solution, in the case of collision cycles, for example, a maximum number of iterations is defined. An example is given in figure I.3 [page 18].

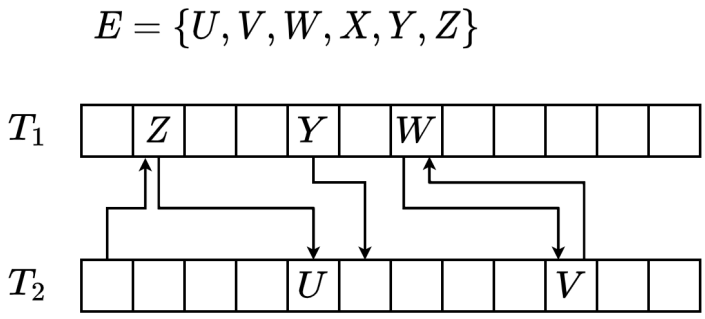

Figure I.3 – An example of a Cuckoo hash table using two tables T_1 and T_2 associated with two hash functions h_1 and h_2, respectively. Arrows represent alternative positions of keys. Keys are inserted following the set order. $h_1(U)$ and $h_1(Z)$ being equals, Z is inserted at its first location $h_1(Z)$ after kicked out U which is inserted at its alternative location $h_2(U)$. V and W are involved in a cycle which prevents the insertion of X that shares the same hash values.

Both operations `lookup` and `delete` are $O(1)$ in worst-case. The time complexity of the `insert` operation is $O(n)$ but an amortized $O(1)$ time is expected with an high probability [93]. In comparison to linear probing, the fastest open addressing scheme [93], the Cuckoo hashing lookup is slower in practice because it often causes two cache misses. However, it is still interesting when the query time is critical thanks to its constant time lookup, which guarantess fast queries independently of the load factor. Recently, the number of cache misses was reduced to 1 on average with multi-way bucketed Cuckoo hashing [94].

A Cuckoo filter (CF) [95] is an AMQF based on a multi-way bucketed Cuckoo hashing (see Figure I.4 [page 19]) allowing fast lookup and high table occupancy. To switch from a hash table to an AMQF, CF uses a scheme called *partial-key Cuckoo hasing*, which consists in storing only constant-sized fingerprints in buckets instead of keys. This prevents the use of the classical Cuckoo hashing insertion algorithm because the relocation is based on rehashing and therefore requires access to the original key. As a result, the possible locations of a key k are computed as follows:

$$\begin{aligned} h_1(k) &= hash(k) \\ h_2(k) &= h_1(k) \oplus hash(f(k)) \end{aligned} \tag{I.2}$$

This scheme shows two important properties: **1.** The fingerprint hashing ensures that the new location is randomly chosen. Without hashing and assuming small fingerprints, the xor operation would only affect the low-order bits and, therefore, a new location close to the original one. **2.** The xor operation allows to compute the alternative location from the original location. In other word, given an original bucket b, the alternative bucket is $b' = b \oplus hash(f(k))$.

Besides this scheme, the `insert` operation follows the same procedure as the cuckoo hashing

by kicking out fingerprints when the buckets are full, and thus the amortized time complexity is still $O(1)$. This insertion algorithm is not appropriate to store a set that contains more than $2m$ (with m the bucket size) copies of an element because the two possible buckets will be overloaded. However, the storage of identical fingerprints allows CF to support the `delete` operation and prevent false deletion on fingerprint collisions.

The `lookup` operation is a bit more expensive because it depends on the size of the buckets. However, for reasonable buckets and fingerprint sizes, all fingerprints of a bucket fit in a single CPU cache-line allowing to preserve efficient queries.

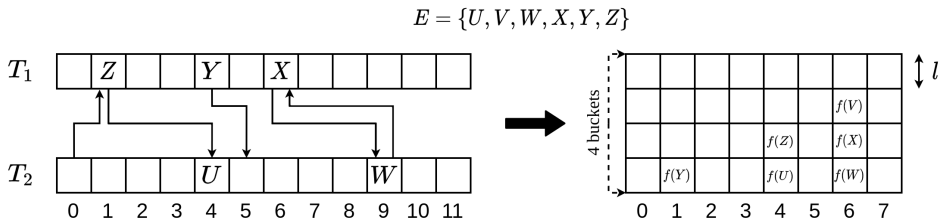

Figure I.4 – From Cuckoo hashing to Cuckoo filter. The CF uses a 4-way bucketed table and a fingerprint size of l. $f(\cdot)$ denotes the key fingerprints. Keys that collided in the Cuckoo hash table are inserted here in the same bucket thanks to the multi-way bucketing.

An CF uses $\frac{log_2(\frac{1}{\epsilon})+log_2(2b)}{\alpha}$ bits per element, where b is the size of bucket and α is the load factor which can reach up 95.5% according to empirical experiments [95]. Consequently, the space required by CF is lower than the BF space when ϵ is small.

iii Quotient

The Quotient filter (QF) [96, 97] is an AMQF that represents a set of elements S by storing p-bits fingerprints in an open hash table T with 2^q buckets. The p-bits fingerprints are obtained using the quotienting technique [98]. A fingerprint $f = h(x)$ where $x \in S$ and $h : \mathcal{U} \to \{0, \ldots, 2^q - 1\}$, is partitioned into f_q, the *quotient*, and f_r, the *remainder* which correspond to the r least significant bits and the $q = p - r$ most significant bits, respectively. The *remainder* f_r is inserted in the bucket $T[f_q]$. Note that the original fingerprint f is given by $f = f_q 2^r + f^r$.

The collision between two quotients, f_q and f'_q, is called a soft colision. Such collisions are resolved using linear probing to store all fingerprints with the same quotient consecutively in a *run* and f_r is always stored before f'_r when $f_q < f'_q$. In addition, each bucket is associated with 3 bits of metadata, is-occupied, is-continuation and is-shifted that provide the state of the buckets.

The possible states are:

— 0|0|0 → An empty slot.

— 0|0|1 → A start of a shifted run

— 0|1|1 → A continuation of a shifted run

— 1|0|0 → A start of a canonical run

— 1|0|1 → A start of a shifted run (canonical slot can exist but is shifted right)

— 1|1|1 → A continuation of a shifted run (canonical slot can exist but is shifted right)

The is-occupied bit of a bucket x is set to one if a quotient $f_q = x$ exists. The is-shifted bit indicates whether the remainder f_r stored in a bucket x is at its canonical location (f_q) or it is shifted. If the remainder f_r stored at x belongs to same run as the remainder stored at $x - 1$, the is-continuation is set to one. In summary, is-occupied tells which buckets correspond to the beginning of a run (a consecutive sequence of remainders with the same quotient), is-continuation allows to identify the start and the end of a run, and is-shifted tells if a remainder is stored at its canonical location. A cluster denotes a set of one or more consecutive runs and is preceded by an empty slot. Note that a empty slot denotes a slot for which the is-occupied is 0. However, such a slot can contain a shifted remainder. An example with 10 slots containing 8 elements is given in figure I.5 [page 20].

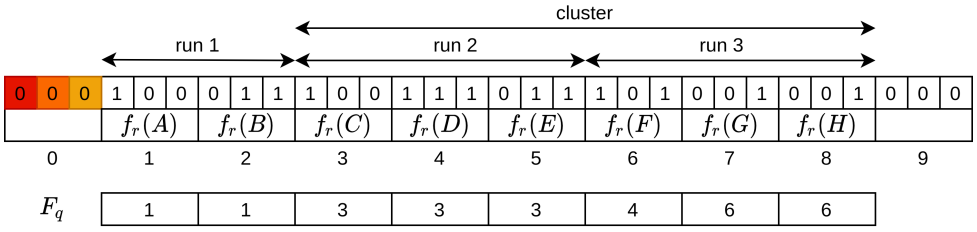

Figure I.5 – Overview of the Quotient filter data structure, adapted from [97]. Each element is inserted following the lexicographic order. The quotients are given by the F_q array.

The lookup algorithm consists in computing the fingerprint, i.e. the quotient and the remainder, for a given element. According to the quotient, a backward scan is performed until is-shifted is 0 to find the beginning of the cluster. The correct run in the cluster is then found using a forward scan and looking at is-continuation. The remainder is finally searched in the run. The insertion and delete operations work in the same way but update the metadata bits and may shift some remainders in the process.

The lookup time depends on the time to scan backward, i.e. find the beginning of a cluster, and to scan forward, i.e. find an eventual remainder. As a result, the lookup time increases with the size of the clusters, i.e. when the occupancy of the QF increases. This also applies to insertion and deletion operations which use the same algorithm to find the location of the remainder. In practice, the authors report that the performance drops off when the load factor exceeds 75%.

Being an AMQF, QF is sensitive to false positives due to collisions. The false positive rate depends on the size of the fingerprints and the number of inserted elements and is given by $1-(1-\frac{1}{2^p})^n$, with p the fingerprint size and n the number of elements in the QF.

The original QF was improved with a new layout, the Rank-and-Select based Quotient Filter (RSQF) [88], which transform QF metadata operations into bit array rank-and-select operations [99]. Such layout allows keeping good performances up to a load factor of 95%, for both insertions and lookups. In addition, it reduces the space by using 2.125 bits of metadata per slot instead of 3. The RSQF was used to build a QF that allows counting. Basically, the Counting quotient filter (CQF) [88] stores "false remainders" that encode actually abundances. The abundance is stored in the bucket that immediately follows a remainder seen more than once. Such representations are used in the context of k-mer indexing in `Squeakr` [89] and `Mantis` [100] (see Section I.1.D.2.a [page 23]).

iv Back to simplicity

As described previously, both CFs and QFs present lower bounds than BFs when the false positive rate is low and propose additional operations such as the deletion of keys. Nevertheless, BFs remains a valuable solution in the context of k-mer indexing for various reasons:

1. **Simplicity.** In the previous descriptions, it is obvious that the BF is significantly simpler than the CF and QF while maintaining very good performances. A BF consists simply of a bit array associated with a set of hash functions. The available operations, `insert` and `lookup`, are simple bitwise operations. In other words, they are not error-prone and very easy to handle from an algorithmic point of view. In addition, being a simple bit-array, BFs natively support the usual bit-array compression techniques such as RRR [101] or roaring compression [102].
2. **Constant time insertions and lookups regardless of the load factor.** BFs do not use any probing scheme to resolve collisions unlike CFs and QFs. As a result, the insertions and lookups are constant and allow to work with high load factors. For example, the first implementation of QF shows bad performances above 75% occupancy. The recent implementation of QF, RSQFs, allows efficient queries up to 95% occupancy at the expense of a little extra space (see Figure I.6 [page 22]). One could argue that working at high load factors leads to high false positive rates. However, false positive rates are not necessarily a problem in our case as mentioned in the next point.
3. **False positive rates do not really matter.** In the context of k-mer indexing, queries are long sequences composed of many k-mers. The false positive rate from the point of view of a sequence query is, therefore, more interesting than the false positive rate of single k-mer. As shown in [56], a false positive rate of 0.5 still allows keeping good results. This effect is discussed in detail in the section I.1.D.2.b.ii [page 26]. In addition,

a technique exists to drastically reduce the false positive rate at query time when a query corresponds actually to a set of sub-queries, e.g. the query of a sequence corresponds to the query of its constitutive k-mers. Such technique is presented extensively in the section I.1.D.3.b [page 34]. Considering these factors, BFs seems perfectly suitable since they use less space than other AMQFs for any false positive rates greater than 1/64 (see Figure I.6 [page 22]). Superior performances in terms of space can therefore be achieved when the applications do not require a low false positive rate, as in our case.

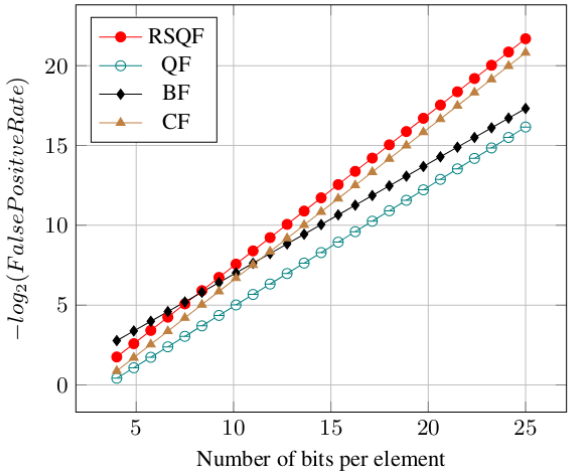

Figure I.6 – Space comparisons of different AMQFs according to the expected false positive rates, reprinted from [88]. BFs uses less space than RSQFs, QFs and CFs for any false positive rates greater than 1/64. The load factors are 100% for BF, 95% for RSQF and CF, and 75% for QF.

4. **Dealing with a filter is not dealing with a collection of filters.** When considering a collection of filters, BFs have a significant advantage. The insertion pattern does not depend on the input set nor on the order of the insertions of the elements. In the case of filters based on probing, the same set of keys can produce a different filter depending on the order of insertions because of collisions. Thus, the same element is not always inserted in the same position. In the case of BFs, for a given size and hash functions, the insertion of a key is always the same regardless of the other keys. This property is exploited by BF-based indexing tool to speed up the lookups (see Section I.1.D.2.b.i [page 24]), and also for saving space in the case of SBT for example (see Section I.1.D.2.b.ii [page 26]).

As a side note, the field of AMQFs continues to progress with the emergence of new filters such as the XorFilter [103]. To our knowledge, the XorFilter is not yet used in bioinformatics. It is a filter that focuses on space at the expense of a slightly longer construction time but still keeps fast queries. In terms of space, it is theoretically smaller than BFs and CFs, whatever the false

positive rates. However, it is not a perfect drop-in replacement as the set of keys to be indexed must be known in advance, in the same spirit as a Minimal Perfect Hash Function (MPHF).

2 Indexing methods

There exist two main and intuitive paradigms to index a collection of sets. Given a collection of samples $D = \{S_0, ..., S_n\}$, the first possibility is to represent the content of each S_i by an arbitrary data structure, e.g. an AMQF. A query consists then in a search in each index. The second one is an inverted index [104] where each element $e_i \in \{S_0 \cup ... \cup S_n\}$ is indexed and associated with a list of samples that contain it. In the context of k-mer indexing, this was formalized by Marchet et al. in a review on k-mers indexing. In this section, I present the two paradigms, k-mer-aggregative (Section I.1.D.2.b [page 24]) and color-aggregative (Section I.1.D.2.a [page 23]), with a focus on a particular representation, namely the Sequence Bloom Tree (Section I.1.D.2.b.ii [page 26]).

a Color-aggregative

As described in the last section, the color-aggregative way consists in indexing the union of each sample and associating each element with a list of colors representing the samples that contain it. Although there are several color-aggregative methods, I present here only one of them that has shown applications on reads indexing and has been compared to this work (see Section IV.2 [page 68]).

Mantis [100] is an k-mer indexing method based on the idea of colored DBG. A colored DBG is a k-mer graph where k-mers are associated with an identifier, namely a color, corresponding to a set of experiments. In the same way, Mantis associates k-mers to colors using a CQF in combination with a color table which maps color IDs to bit-vectors representing collections of samples. The construction of the index consists of building one CQF per input samples using `Squeakr` [89], a k-mer counter based on CQF. All CQFs are then merged into a single CQF which stores color IDs instead of the k-mer abundances. Squeakr has the advantage to present an exact mode and therefore allow to build an exact representation of the Mantis index. However, the exact CQF of Squeakr is significantly more expensive in terms of space and is therefore not suitable for all types of datasets.

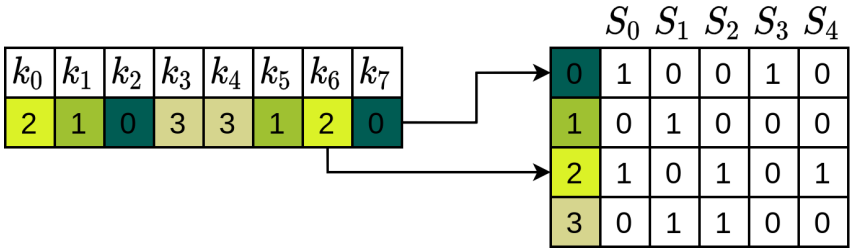

Figure I.7 – Overview of Mantis [100]. The k-mers are stored in a CQF, where remainders correspond to color classes. Each color class corresponds to a membership pattern and indicates the samples that contain a k-mer.

b k-mer-aggregative

i Bloom filter matrix family

As described in section I.1.D.1.b.iv [page 21], BFs have the particularity of not requiring any addressing scheme. In other words, for a given array size, an element will always be inserted at the same position, whatever the sample. A Bloom filters collection can then be seen as an inverted index through a bit-sliced layout [106].

Given a collection of BFs B of the same size m, these are stored as a $|B| \times m$ matrix in a column-major way. A row represents the membership of one index (i.e. a k-mer) in all samples.

In this section, I present three methods that use different forms of this representation to index collections of biological sequences. Such a type of layout is used as a building block in this work, and it is described in section IV.1.B [page 63].

Bitsliced Genomic Signature Index (BIGSI) [107] BIGSI is the first method to use filter arrays to index collections of samples. The matrix is built by concatenating the filters representing the samples.

The queries consist in retrieving the set of row vectors corresponding to the k-mer hash values of the queried sequence. In the first version of the tool, a bitwise AND between these vectors allowed to obtain a bit-vector representing the presence or absence of the query in each data set. The current tool also proposes to add these vectors to allow partial matches (as presented in the Figure I.8 [page 25]).

The bit-matrix layout offers several advantages in terms of construction and query. First, the construction does not require complex operations and does not require any temporary space. Also, this layout facilitates the addition of new data. Indeed, the addition consists simply of the concatenation of one or more new columns to the bit matrix without the need to recalculate the index in its globality.

From a query point of view, the bit matrix allows time and cache-efficient queries since it simply consists in retrieving a vector with given indexes. However, to keep this efficiency, the bit-vectors are not compressed. Consequently, such types of indexes allow very fast queries with relatively high index sizes.

BIGSI is the first tool to implement a very large-scale genome indexing with the indexing of all bacterial and viral genomes present in ENA in 2016. The final index occupied a space of 1.5 TB as opposed to 170 TB for the input sequences. The authors also showed that this type of index could greatly accelerate genotyping. They show 10,000x faster results and 99.997% agreement with a samtools-based genotyping pipeline.

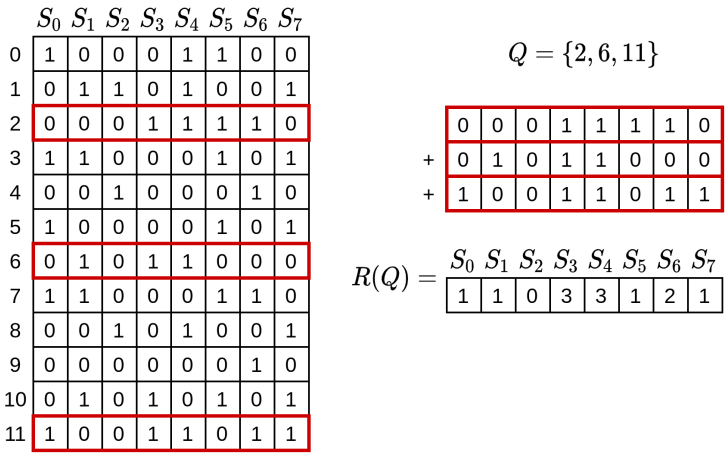

Figure I.8 – An example of BIGSI [107] on 8 samples using one-hash BF. The query Q is directly represented by the hash values. $R(Q)$ denotes the response, an array representing the number of positive matches in each sample obtained by adding all vectors corresponding to the hash values.

Compact Bit-Sliced Signature Index (COBS) [108] is the successor of BIGSI and is based on the same principle but more adapted to samples with a variable number of k-mers. It addresses a common problem when indexing samples of very different sizes. Usually, the size of the core data strucuture, here a BF, is adapted to the size of the largest sample. Thus small samples are represented by over-scaled filters resulting in a loss of space. COBS proposes to build several matrices corresponding to different filter sizes (as described in figure I.9 [page 26]). For that, a staircase function corresponding to the size of the filters is determined from a group size and the list of samples, sorted by size.

This scheme has an impact on query efficiency since it is now necessary to query multiple arrays. However, its efficient implementation (SIMD, memory-mapped I/O) allows for more efficient construction and queries than BIGSI. The final index is also smaller when the sample size is variable.

COBS has some limitations. Namely, a kmer size limited to 31 and a preprocessing of samples with McCortex, which is a DBG constructor. It allows here the decomposition in k-mers and the filtration. Although very efficient for genomes, McCortex is very slow for read sets compared to classical k-mer counters.

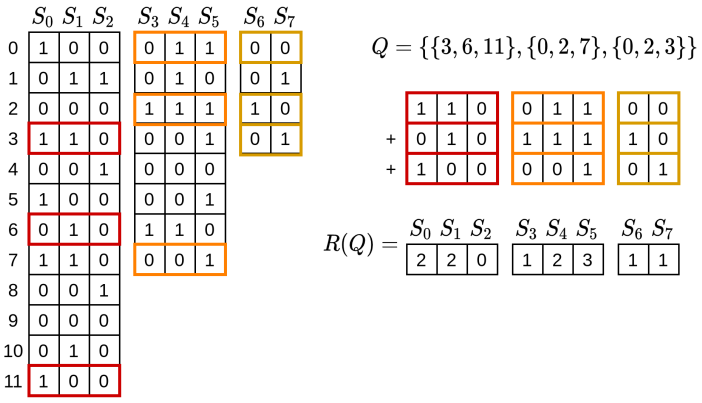

Figure I.9 – An example of COBS [108] on 8 samples using one-hash BF. BFs are grouped by sizes which are determined according to sample sizes. The query Q is represented by 3 sets of hash values, one per BF group. $R(Q)$ denotes the response for each group. The lookup procedure is the same as in BIGSI and is simply repeated for each BF group.

MetaProFi [109] is the most recent unpublished method that uses a filter matrix. Unlike the previous ones, the bit vectors are compressed via Zstandard compression algorithm. In order to keep queries efficient, the vectors are compressed by chunk, allowing to decompress only the chunk involved in a query. This compression is done with Zarr [110], a python library using the Zstandard algorithm [111] and allowing the creation and storage of chunked and compressed N-dimensional arrays.

It has the advantage of allowing indexing and retrieval of nucleic and amino acid sequences. However, it does not apply any filtration step (no k-mer counting), making it difficult to use on read sets.

ii Sequence Bloom Tree family

SBT are binary tree [112]-based data structures designed to reduce the cost of searching in a collection of BFs and, more recently, to reduce the space by exploiting redundancy between sets. In this section, I introduce the abstract type before focusing on the four variants and their respective improvements and compromises. Special attention will be given to the most recent one, which has been used intensively in this work.

Given a sequencing dataset D composed of a collection of samples S_i represented by their Bloom filters B_i, a naive query would consist in searching in each filter. However, it is reasonable to expect some redundancy between the samples; this means that some k-mers are shared

between the samples. SBTs exploit this redundancy to group filters that are similar to limit the number of operations during a query. As development progressed, this aspect also allowed to move towards an efficient compression of the data structure.

Terminology

Let D a dataset composed of a collection of samples S_i. The database representation of S_i is denoted as $B(S_i)$ and represents all entries from S_i. It denotes an abstract type allowing membership queries, whose internal representation depends on the SBT implementation. The set of k-mers of a sequence S is denoted $K(S)$.

Let $\mathcal{T}$ a rooted binary tree. Given a sub-tree rooted at an internal node u, the set of its leaves is denoted $l(u)$. Let $c_l(u)$ and $c_r(u)$ denote the left and the right children of a node u. The parent of a non-root node v is $p(v)$.

The intersection between two bit-vectors x and y of the same length is written as $x \wedge y$, and the union is $x \vee y$.

A SBT is a rooted binary tree where $B(u)$ denotes the database representation of a leaf u. For an internal node v, $B(v)$ is $\bigvee_{i \in l(v)} B(i)$ and therefore is a representation of all entries contained in $l(v)$ which is equivalent to $B(C_l(v) \vee C_r(v))$. The matches of a query sequence Q are the leaves i for which $\frac{|\{k \in K(Q) \,:\, k \in B(i)|\}}{|K(Q)|} \geq \theta$, where θ is a user-defined threshold in $[0, 1]$. This tree representation allows reducing the cost of negative queries because a negative query at a node u means that the element does not exist in $l(u)$. SBTs are dynamic structures, allowing insertion and deletion of new experiments, and thus existing SBT can be updated when more samples are needed. In practice, the insertion and deletion operations are not always implemented.

SBT [56] by Solomon and Kingsford is the first implementation (an overview of the structure is presented in the Figure I.10 [page 29]) and gives several contributions:

1. **A greedy construction algorithm.** The construction algorithm is designed to group together similar Bloom filters. As the internal nodes are the union of their children, grouping dissimilar filters leads to the saturation of top-level filters. At the same time, this allows to reduce the query time by favoring an earlier pruning of the tree. Given a collection of Bloom filters D and a possibly empty tree $\mathcal{T}$, the insertion of a BF B in $\mathcal{T}$ is performed in the following way:
   — **The current node u has a single child.** B is inserted as the second child of u.
   — **The current node u has two children.** B is compared to the two children of u, $c_l(u)$ and $c_r(u)$. The most similar node, according to the Hamming distance between

the BFs, becomes the current node, and the process is repeated.

— **The current has no children.** The current node represents a sample. A new union node v is created as a child of $p(u)$. The new node v is then the parent of u and B.

Once the tree is built, all the nodes are compressed using the RRR compression [101].

2. **A reduction of query time.** The tree representation allows to reduce the query time. At a given node, the query process can be aborted if the threshold *theta* is already reached. However, it is not possible to predict if the threshold can no longer be reached at a given node. Negative queries require therefore examining all nodes of the tree.

3. **An analysis of false positive effects on sequence searching.** The false positive rates of Approximate Membership Querys (AMQs) correspond to the probability of observing a false match in a single query. Yet the use of indexes is more important for sequence queries than for single k-mer queries. The link between false positives on k-mers and false positives on queries is explored in theorem 2 of [56]. This theorem shows that for a given threshold, it is unlikely to observe a fraction of shared k-mer greater than θ when the false positive rate on k-mers is also fixed at θ. Thus, working with high false positive rates leads to few errors when considering the query of a collection of k-mers representing a sequence.

4. **An application on a RNA-seq dataset.** This implementation was benchmarked on a collection of 2,652 human RNA-Seq samples corresponding to the entire publicly set of blood, brain, and breast samples stored at the SRA at the time of the experiment. This dataset is now the reference benchmark for other k-mer indexing tools. On a single core, the construction of the index requires about 2.5 minutes per file with a final size of 200GB (2.3% of the input size). The query of a single transcript is achieved in 20 minutes on average. In comparison, the estimated running time for SRA-BLAST and STAR was 2.2 and 921 days, respectively. However, it is important to note that both tools return sequence alignments while an SBT query provides only a probability of presence/absence result.

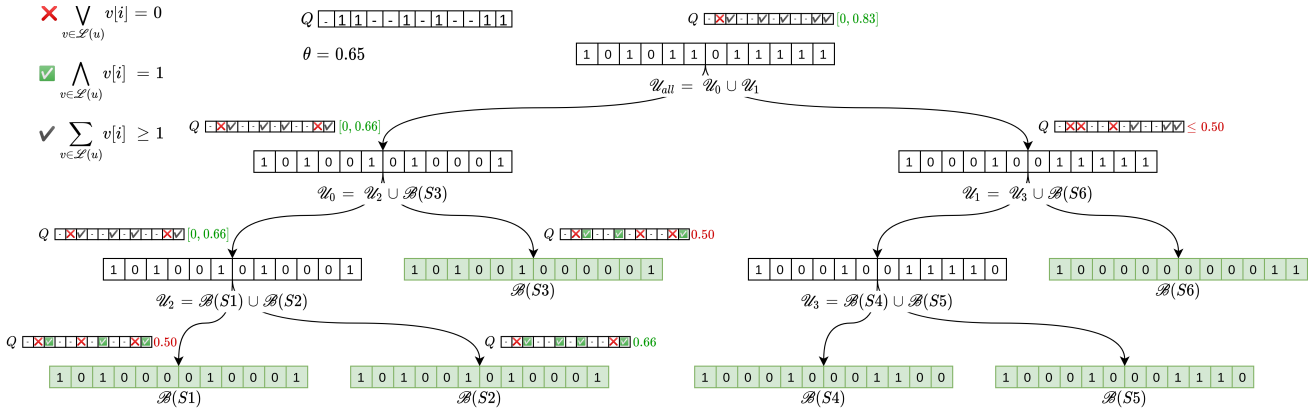

Figure I.10 – Overview of the Sequence Bloom Tree, first implementation from [56]. The query Q is searched in the tree accordingly to a threshold of $\theta = 0.65$. The query state is given at each node.

AllSome Sequence Bloom Tree (SBT-ALSO) [113] by Sun et al. proposes a new representation and construction algorithm. An overview of the structure is presented in the figure I.11 [page 30]. The main novelties are listed below.

1. **The $\mathcal{B}_{All}$ and $\mathcal{B}_{Some}$ representation.** In this implementation, two bit-vectors are used to represent each node. For given node u, $\mathcal{B}_{All}(u) = \mathcal{B}_{\cap}(u) \setminus \mathcal{B}_{\cap}(p(u))$ where $B_{\cap}(u) = \bigcap_{i \in l(u)} \mathcal{B}_{\cup}(i)$ with $\mathcal{B}_{\cup}$ the union of all positive bits in the sub-tree from u. $\mathcal{B}_{All}(u)$ represents therefore the bits that are set in all the leaves of the sub-tree starting at u. $\mathcal{B}_{Some} = \mathcal{B}_{\cup}(u) \setminus \mathcal{B}_{\cup}(u)$ and therefore represents the bits that are positive in some $l(u)$ but not in all. This representation allows to speed up the query for several reasons. First, $\mathcal{B}_{All}(u)$ allows to know if a bit is positive in $l(u)$. Thus a positive k-mer in $\mathcal{B}_{All}$ at a given node u does not need to be queried in the sub-tree starting at u. In the previous implementation, a positive match at a node could correspond to a bit present in only one of the leaves and therefore do not allow aborting the query of a k-mer. In contrast, $\mathcal{B}_{Some}$ allows to know if some leaves contain a given bit and therefore enables an early pruning of the tree during the queries. Note that both $\mathcal{B}_{All}(u)$ and $\mathcal{B}_{Some}(u)$ exclude the positive bits in $\mathcal{B}_{\cap}(l(u))$. Thus, the bits resolved above in the tree are here set to 0 for the sub-tree starting at u and are not useful anymore. The deletion of these inactive bits is explored by the next implementations.
2. **The greedy construction is sub-optimal.** The greedy construction algorithm introduced by [56] is sensitive to the order of the inputs. As a result, the tree may have a poor localization if these inputs are not in the right order, i.e. the closest sets in terms of k-mers are not added sequentially. A sub-optimal order can significantly increase the query time. Indeed, a smaller number of nodes are explored if all the query matches occur in the same sub-tree.

3. **A new construction algorithm based on clustering.** To improve the localization of the samples in the tree, Sun et al. propose a new construction algorithm based on agglomerative hierarchical clustering. Each sample is represented by a SBT. The final SBT is computed by merging two temporary SBT. At each iteration, the two SBT which present the smallest Hamming distance between their root nodes are merged.

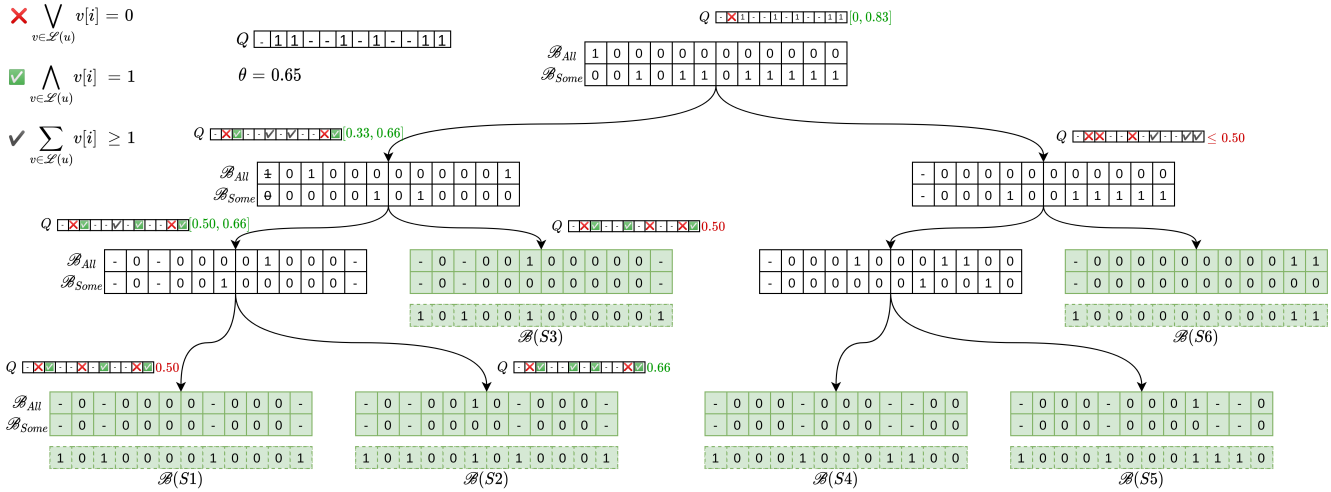

Figure I.11 – Overview of the AllSome Sequence Bloom Tree [113]. The query Q is searched in the tree accordingly to a threshold of $\theta = 0.65$. The query state is given at each node. The bits represented by "-" are inactive.

Split Sequence Bloom Tree (SSBT) [87] by Solomon and Kingsford was proposed independently of SBT-ALSO and presents an equivalent representation with new handling of the inactive bits. The figure I.12 [page 31] is an overview of the data structure.

1. **The $\mathcal{B}_{sim}$ and $\mathcal{B}_{rem}$ representation.** The $\mathcal{B}_{sim}$ and $\mathcal{B}_{rem}$ representation was proposed simultaneously to SBT-ALSO and presents the same layout with two bit-vectors per node. $\mathcal{B}_{All}$ and $\mathcal{B}_{Some}$ are respectively equivalent to $\mathcal{B}_{sim}$ and $\mathcal{B}_{rem}$. However, the construction process differs since it follows the greedy algorithm introduced by the first implementation.

2. **The deletion of inactive bits.** As discussed before, such representation implies inactive bits in internal nodes. An inactive bit at a node u is a bit for which its value can be determined by one of its parents. SSBT proposes to remove these bits in order to reduce the size of nodes. The authors specify that the suppression of these bits is more effective when the samples are uniform in terms of k-mers. Thus, one could argue that the SBT-ALSO construction algorithm would be useful because it allows to group similar samples whatever the order of the inputs. The most recent implementation, presented below, uses this observation while proposing a new representation allowing additional

pruning of the tree during the query.

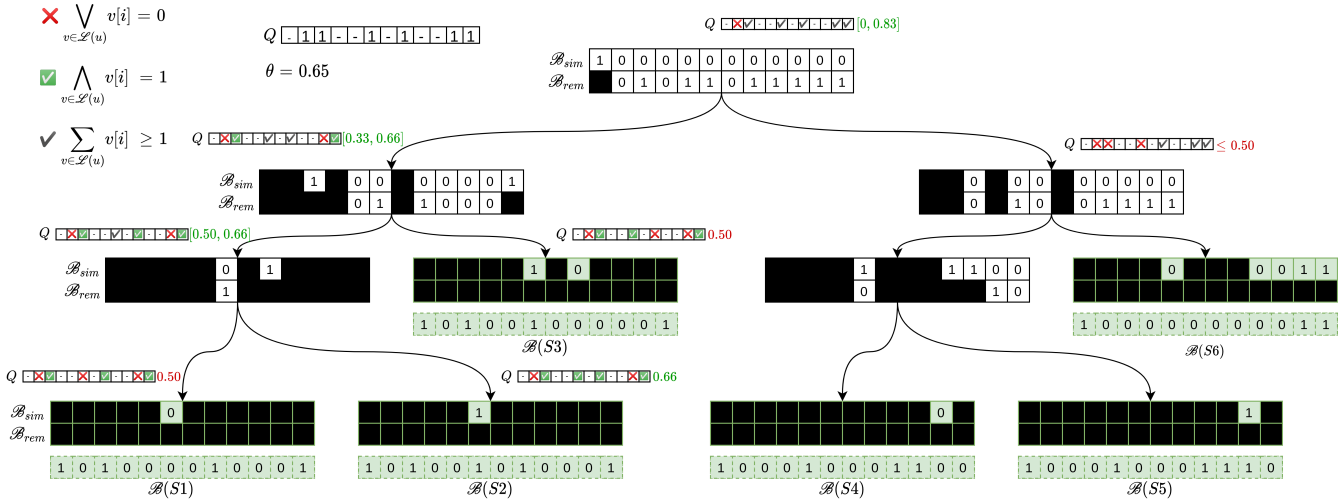

Figure I.12 – Overview of the Split Sequence Bloom Tree [87]. The query Q is searched in the tree accordingly to a threshold of $\theta = 0.65$. The query state is given at each node. The bits represented by a black square are inactive and removed.

HowDeSBT [114] by Harris and Medvedev is the most recent implementation and proposes a new representation, also based on two bit-vectors per node, allowing additional pruning of tree as well as better compression ratios. The structure is presented in figure I.13 [page 33]. The main contributions of HowDeSBT are presented below.

1. **The $\mathcal{B}_{det}$ and $\mathcal{B}_{how}$ representation.** The proposed representation also relies on two bit-vectors and allows to save bits in terms of space and lookups. For a given node u, the bit-vectors are defined as follows:

$$\mathcal{B}_{det}(u) = \mathcal{B}_{\cap}(u) \cup \overline{\mathcal{B}_{\cup}(u)} \tag{I.3}$$

$$\mathcal{B}_{how}(u) = \mathcal{B}_{\cap}(u) \tag{I.4}$$

$\mathcal{B}_{det}(u)$ represents the determined bits, i.e. the bits that have the same value in $l(u)$. $\mathcal{B}_{how}(u)$ informs about the value of determined bits is informative only for bits set to 1 in $\mathcal{B}_{det}(u)$. In the same spirit as SBT-ALSO and SSBT, such representation allows efficient queries by resolving some positions before reaching the leaves. When a position is determined in $\mathcal{B}_{det}(u)$, the corresponding k-mer is considered as resolved in $l(u)$, whether it is present or absent. A lookup in $\mathcal{B}_{how}$ then allows obtaining its value. When a position is not determined, the lookup of $\mathcal{B}_{how}(u)$ is not required. Although the representation from SSBT and HowDeSBT seems highly similar (the different states of bits are described

in the table I.1 [page 32]), the HowDeSBT query requires fewer lookups to be resolved. The numbers of lookups required by both representations are given in the Theorem 2 from [114] and are $(3n+1)/2$ and $(n(4-s)-s)/2$ for HowDeSBT and SSBT, respectively, with n is the number of nodes in the tree and s the ratio of active bits in the leaves. The difference is explained by query algorithms. In the case of HowDeSBT, only one lookup to $\mathcal{B}_{det}$ is required for every internal node. For a given position, a positive lookup in $\mathcal{B}_{det}$ prunes the tree and the current node becomes a leaf. At the leaves, each $\mathcal{B}_{how}$ vectors must be queried. For SSBT, the pruning of the tree (transform an internal node to a leaf for a given position) occurs only when the lookup to $\mathcal{B}_{sim}$ is positive (table I.1 [page 32]-C). In other cases, two lookups are required for every node.

In summary, the improvement in lookups depends on s (the percentage of active bits in the leaves of the sub-tree) and is between 0 and 25% for $s = 1$ and $s = 0$, respectively.

	SSBT		HowDeSBT	
	$\mathcal{B}_{sim}$	$\mathcal{B}_{rem}$	$\mathcal{B}_{det}$	$\mathcal{B}_{how}$
A	0	1	0	-
B	0	0	1	0
C	1	-	1	1
D	-	-	-	-

Table I.1 – State correspondences between $\mathcal{B}_{sim}$ and $\mathcal{B}_{rem}$ from SSBT and $\mathcal{B}_{det}$ and $\mathcal{B}_{how}$ from HowDeSBT.

2. **An analysis of the Shannon's information compression bounds.** The Shannon's information [115] bounds for both SSBT and HowDeSBT are given by the Theorem 1 from [114]. The improvement of the space bounds in HowDeSBT is between 9% and 14% for $s \leq 0.75$. The HowDeSBT representation is theoretically more compressible.
3. **The smallest and the fastest implementation of the SBTs structure.** HowDeSBT was compared to SBT-ALSO and SSBT on the benchmark RNA-Seq collection introduced by [56]. Regarding the construction, HowDeSBT is 2.8x and 6.3x faster than SBT-ALSO and SSBT, respectively. The temporary space used by all tools is equivalent since it mainly corresponds to the BFs, one per input sample. The size of the HowDeSBT final index is 10.1x and 1.6x smaller than SBT-ALSO and SSBT indexes, respectively. HowDeSBT is also most efficient at query time, whether for single or batch queries. Note that the computation times and disk usages concern only the construction of the tree from the collection of BFs.

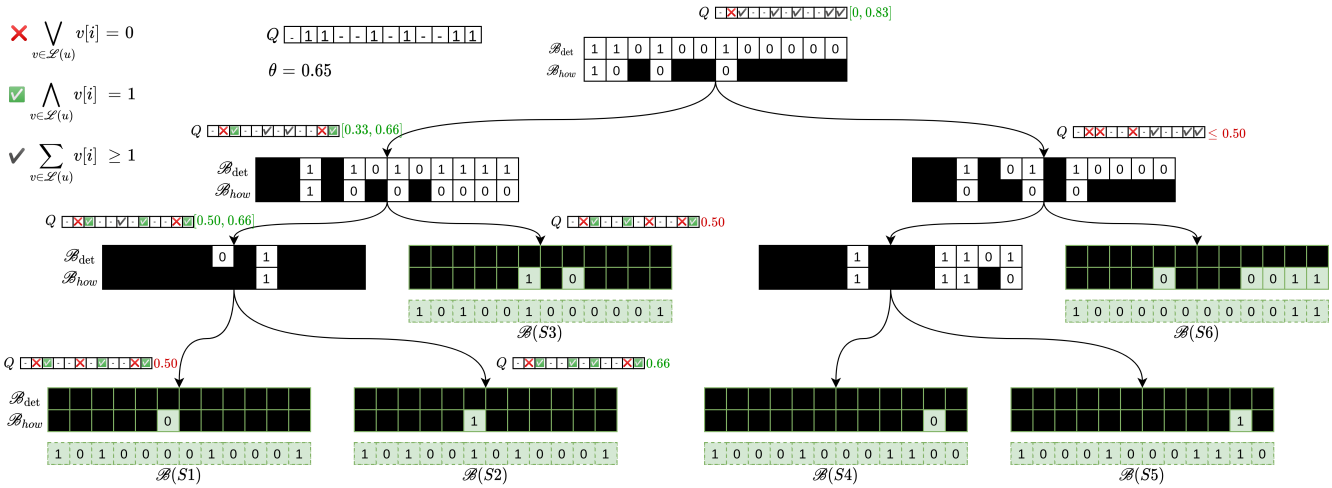

Figure I.13 – Overview of the HowDe Sequence Bloom Tree [114]. The query Q is searched in the tree accordingly to a threshold of $\theta = 0.65$. The query state is given at each node. The bits represented by a black square are inactive and removed.

c A story of trade-offs

The different k-mers indexing tools have been compared several times in each individual publication, but also in several reviews [105, 116]. In summary, these tools offer different implementations and features that lead to different trade-offs. However, one step in the construction of the indexes is often not mentioned. As discussed, many structures build core structures for each sample before creating the final compact index. Being k-mer indexing tools, they require by definition sets of k-mers.

Depending on the type of the inputs, these k-mers can be obtained in different ways. 1) The case of assembled sequences. The assembled sequences are assumed to be error-free and can be directly broken into k-mers, at least for the presence/absence tools which do not use the abundance of k-mers. 2) The case of sequencing reads. Reads are erroneous sequences, at different degrees depending on the sequencing platforms. Consequently, the k-mers obtained from these sequences also contain errors. The usual way to filter these errors consists in discarding rare k-mers, usually those seen only once, and therefore relies on k-mer counting.

The construction of the core data structure, including the counting of k-mers, is sometimes discussed and often not evaluated in k-mer indexing papers. Unfortunately, this step is the most expensive compared to the total resources used to build the index. For example, in the experiments presented in section IV.3.C [page 73], it represents 85% of the construction time.

No tool proposes a specific method combining counting and construction. The counting and the construction are always separated and often achieved by different tools without particular strategies. In other words, the construction of the core data structures uses arbitrary sets of k-mers as inputs, so no assumptions can be made to speed up their constructions.

We argue that the "pre-processing" step should be considered and that the problems of counting and core data structure construction should not always be addressed as two distinct problems. Many k-mer indexing tools are based on BFs, which are highly suitable for k-mer indexing problems despite their apparent theoretical weaknesses as previously discussed in sections I.1.D.1.b.iv [page 21] and I.1.D.2.b.ii [page 26]. In this work, we show how a specific k-mer counting technique can help to build BFs efficiently, which can be used as inputs by indexing tools. In particular for HowDeSBT that is the most modern and efficient SBT representation. The methods and results related to this work are presented in the chapters II [page 41], IV [page 61], and V [page 81].

3 Querying method

a Naive k-mer queries

As stated in section I.1.B [page 13], the naive way to query a sequence against a k-mer index consists of querying all k-mers composing the sequence. Thus, the probability of a false perfect match is ϵ^n, where ϵ is the false positive rate and n the number of k-mers. A recent method, presented in the next section, defines a new query model, allowing to reduce this probability while reducing the query time.

Note that some information can be derived from naive queries and bring additional insight to help the interpretation of results. The position of k-mers in the query allows to infer the coverae of the query, i.e. the number of bases covered by one or more k-mers. The benefits of this information are discussed in section V.2.B.3 [page 86].

b Findere

`findere` [4] is a query-time technique allowing to reduce the false positive rate of an arbitrary AMQF while reducing the number of lookups to resolve a query. It only concerns the query part and can be applied to an existing index. Its scope concerns the indexing of objects that are decomposable into sub-objects, i.e. a sequence of characters. It is based on the following straightforward statement: if a sequence $S \in E$, then its sub-words $w_i \in E$.

Let S a sequence of characters and $W = \{w_1, ..., w_n\}$ the set of its sub-words of fixed lengths. Given an arbitrary AMQF with a false positive rate of ϵ, the probability of observing a false perfect match (all sub-words are wrongly found) is ϵ^n. Using `findere`, each w_i is divided in m sub-words, called s-mer, the probability of observing a false positive for a given word is then ϵ^m. Thus for S, the probability becomes $(\epsilon^m)^n$. In addition to drastically reducing the false positive rate (Figure I.14 [page 35]), `findere` also allows reducing the number of bit lookups. Consecutive words share some s-mers, which can therefore be queried only once. If a s-mer is absent, all consecutive words containing it are simply not queried.

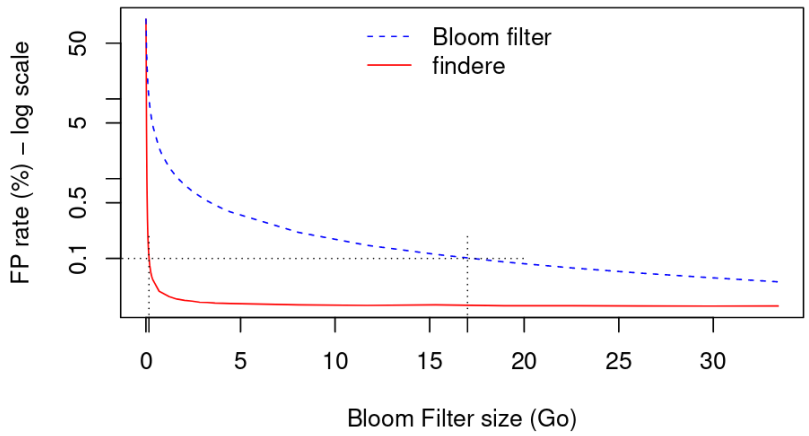

Figure I.14 – Impact of `findere` query algorithm on the BF false positive rate, reprinted from [4]. The experiment was performed on a metagenomic sample from HMP [55]. The false positive rates induced by `findere` are empirically determined at query time.

Although this technique applies to any AMQF, its synergy with BFs seems particularly interesting. As described in section I.1.D.1.b.i [page 16] and I.1.D.1.b.iv [page 21], a high load factor results in poor performances for many AMQFs. Regarding BFs, insertion and query operations do not rely on the load factor. Consequently, `Findere` is one more argument in favor of using BFs with relatively high false positive rates, which are space-efficient and easier to build.

2 Sequence analysis: differential k-mer analysis

In the context of population studies, the most common are Genome-Wide Association Study (GWAS), allowing to identify sequences and genes responsible for a particular trait, such as a disease. Large groups of individuals are studied for small variations, the Single-Nucleotide polymorphisms (SNPs), corresponding to single-base variations. Thanks to a large number of individuals, it is possible to identify statistically more frequent variants in a group with a particular trait than in another control group. In spite of their great usefulness and contributions, GWAS have some drawbacks. First of all, it usually only deals with SNPs, without considering the largest variations, the structural ones. This is mainly due to the technique used, based on the hybridization of DNA sequences from individuals on chips containing a pre-established DNA sequence of known variations. In addition, implementing these analyses is much more complex and costly than a bioinformatics analysis. Especially since we already have many ready to use sequencing cohorts, thanks to projects like 1000 Genome [54], for example.

Recently, studies with similar objectives using sequencing data have emerged. The idea is to find sequences, and more precisely k-mers, that are differentially represented between several

sequencing cohorts, typically two. These k-mers can then be used to reconstruct a set of variants. In the context of this work, we are only interested in the detection of these k-mers, in a one-to-one context.

Several tools allow doing differential k-mer analysis [117, 2, 46, 3]. In a few words, it consists of comparing k-mer abundances across samples. For each k-mer, we need its abundance vector, representing its abundances in each sample. In the context of a one-to-one experiment, the abundance vectors are used to test if k-mers are statistically differentially represented between two conditions. In the context of continuous phenotype values, it consists of testing if the distribution of phenotype values explains the k-mer abundance vectors. In both cases, the bottleneck of the tools was counting the k-mers and aggregating the abundances to obtain counting information across samples, i.e. the abundance vectors. The idea was to speed up existing methods by allowing an efficient recovery of all k-mer abundance vectors, i.e. via an efficient construction and streaming of a k-mer matrix. We thus focused on `HAWK` [2, 3], a tool allowing one-to-one experiment, particularly adapted to the case of disease study, where the groups are controls and cases.

We showed that the implementation of state-of-the-art statistical methods in a more efficient computational framework leads to significantly superior performance of differential k-mer analysis. The work related to differential k-mer analysis is presented in chapters II [page 41] (about the construction of k-mer matrices) and III [page 51] (about the usage of k-mer matrix in the context of differential k-mer analysis).

3 k-mer counting

The methods presented in this chapter all rely on k-mer decomposition, i.e. the decomposition of one or more sequences into its k-mer set. The methods used to perform such an operation depend on the type of inputs. Indeed, assembled sequences are assumed to be exact, while sequencing reads may contain some errors. A k-mer set built from a collection of reads can therefore contain many erroneous and undesirable k-mers. Fortunately, it is possible to discard these errors by counting the number of occurrences of each k-mer. Given a sequencing coverage, the abundance of a unique k-mer (present only once in the input genome) should be around this sequencing coverage. The erroneous k-mers can then be identified and discarded.

Being a fundamental operation of any k-mer-based method, the k-mer counting is a well-studied problem. Several methods coexist and can be classified into two main paradigms: in-memory counting and disk-based counting. Although both paradigms lead to the same results, each method has its strengths, weaknesses, and specificities. The characteristics of each technique are important and should guide the choice of the counter according to the downstream analyses.

In the following sections, I present the two main paradigms used for k-mer counting and

several examples of tools that implement them.

A In-memory counting

In-memory methods are usually based on a hash table that associates k-mers with their abundances. Although the general idea is similar to all methods, the implementations are diverse, with various hashing, filtering, and multi-threading techniques.

The main drawback of these methods is their high memory usage. To address this problem, some methods like `Jellyfish` [118] or BFCounter [119], use a BF to identify and reject singletons. A k-mer is then inserted in the hash table only if it is already present in the BF. However, this method increases the computation time because false positives require a second pass. Indeed, when two unique kmers are inserted and collide in the BF, the second one is inserted with an abundance of 2 in the table.

An exact k-mer counting is not always required. Some methods propose to do approximate counting to reduce the computational cost further. For example, Squeakr [89] offers an approximate mode based on a CQF. In this case, footprint sizes determine the counting accuracy and the performance in terms of time and memory.

Another issue is the synchronization of the hash tables. This problem has been partly solved by lock-free approaches [120] or by using multiple hash tables, as in the unpublished `HackGap` (`https://gitlab.com/rahmannlab/hackgap/`) or `Jellyfish`.

In addition, such techniques require estimating the expected number of distinct k-mers to set the table sizes. A too low user-specified value leads to longer running time and a higher memory usage due to multiple resizing and merging of hash tables.

For all these reasons, in-memory methods are often relatively expensive in terms of time and memory. However, they have a significant advantage. The final representation of the counted k-mers is a hash table. Consequently, it is possible to use the output as a key-value database in other tools thanks to the associated command line tools or library. When this type of operation is unnecessary, the disk-based tools presented in the next section are probably more suitable.

B Disk-based counting

The improvement of disk performances has allowed the development of disk-based counting methods with a minimal memory footprint. Anecdotally, such methods have, for example, allowed performing an assembly of a genome on a Raspberry Pi in 2013 [121].

Disk-based methods rely on the divide-and-conquer paradigm. The idea is to form groups of k-mers and process these groups successively or in parallel according to a given amount of memory. Thus, the minimum memory required by this type of technique corresponds to the memory needed to count one of these groups of k-mers.

To form these groups, it is necessary to identify a common characteristic between k-mers to guarantee that identical k-mers are always put in the same group. For that purpose, the concept of minimizer has been borrowed from the assembly [122]. A minimizer M of a sequence S is the sub-word of size m that minimizes a given function. Usually, a hash function or simply the lexicographic order. By definition, identical k-mers always have the same minimizer, whatever their origin.

The general functioning of k-mer counting on disk is as follows. The sequences, mainly reads, are split into k-mers, and these are written on disk in a file corresponding to their minimizer, often called bin or partition. Then the partitions are counted successively or in parallel according to the available memory. The counting method is variable and relies on hash tables or sorting. In the second case, k-mers from a given partition are sorted. The abundance of a k-mer is then given by its number of consecutive occurrences. Today, sorting methods are favored.

The idea of dispatching k-mers for counting appeared with `Meryl`, the k-mer counter of the `Celera` assembler [123]. Improvements were then proposed and resulted in various tools like `DSK` [124] or `KMC` [125, 126, 127]. Recently, a new unpublished counter, FastK (`https://github.com/thegenemyers/FASTK`) optimized for long reads was released.

As for the in-memory counting, the outputs also have some specificities. Indeed, as the counting is based on sorting, the output is partially sorted. This order allows to make log-time queries and facilitates set operations between multiple k-mer databases.

Thanks to the progress of hardware, these methods are much faster nowadays than in-memory methods. `KMC` is the most used, both for its performance and its robustness.

4 Conlusion

About indexing, BFs are an interesting option when put in perspective with the specific problem of sequencing samples indexing, despite the theoretical and functional advantages of more modern AMQF. However, they suffer from the problem of data locality, complicating their construction. Indeed, the first step, consisting of the creation of one AMQF per input sample, is the main bottleneck of indexing methods. Note that this is not specific to BFs; the construction of AMQF from sequencing samples is always the bottleneck.

About the differential k-mer analysis, the bottleneck is also related to k-mers. It is about obtaining abundance tables for each sample, before aggregating them to get k-mer abundances across samples.

In both cases, the problem is related to k-mer counting, i.e. the counting is always considered as a pre-processing step. Thus, the intrinsic properties of k-mer sets are not exploited, although interesting in downstream processing, as we will see in the following chapters. In summary, the objective is to bring k-mer counting and further processing together by exploiting the k-mer

counting properties.

We argue that the k-mer matrix concept can help in addressing these problems. As a reminder, a k-mer matrix associates k-mers with their abundances across several samples. The next chapter is dedicated to this object and presents our construction method, as well as the new analysis opportunities it offers, notably in terms of k-mer filtering.

The other chapters are dedicated to its use in various contexts, sometimes requiring modifications of the initial representation and construction methods.

Chapter II

The k-mer matrix representation

Preamble

In this chapter, I introduce the concept of k-mer matrix, i.e. a matrix that associates k-mers with their abundances across multiple samples, and describe how such representation can be helpful in various contexts from analysis to indexing. Although the concept is already known and used, no methods and tools were dedicated to its efficient construction and processing. Here we described how k-mer counting algorithms could be extended for fast and parallel construction of such matrices.

One of the major concerns with k-mer-based technique is the sequencing errors, resulting in many erroneous k-mers. Simple methods exist to address this issue and allow discarding erroneous k-mers. However, classical techniques are not always suitable for complex sequencing samples such as metagenomes. We describe how the holistic view of k-mer abundances across multiple samples enables a new k-mer filtering method that we call k-mer rescue.

This chapter describes a method for constructing k-mer matrices and some opportunities provided by the matrix representation. In-depth presentations of their use in analysis and indexing are discussed in chapter III [page 51] and IV [page 61], respectively. Regarding indexing, the methods presented here are modified and extended (see Chapter IV [page 61]) to allow indexing of k-mers and benchmarked on a large-scale metagenomic dataset. The reliability of the k-mer rescue is also evaluated on this dataset (see Chapter IV [page 61]).

Contents

1 A screening of raw sequencing collections

When one wants to represent and/or compare the sequence content across multiple sequencing experiments, one solution is to represent the abundance or presence of each sub-sequence of a fixed size in all samples. One possible type of structure is the k-mers matrix. Given a collection S of N samples and size of k-mers of k, a k-mer matrix M is an abstract data type representing the abundance of each k-mers belonging to S. In other words, each element $M(i, j)$ is the abundance of the k-mer i in the sample j. Depending on the context, M can be a binary matrix where $M(i, j)$ is a bit denoting the presence or the absence of a k-mer. This type of representation is the building block of the work presented in chapter III [page 51] and IV [page 61].

Although this concept is already used in various analyses and tools, no methods and tools are specifically dedicated to its construction and processing. Each tool uses more or less naive methods based on traditional k-mer counters.

In our case, we are interested in a non-indexed form of such matrices, i.e. a matrix allowing sequential access. The problems studied in this work do not require a query-able representation but simply successive views of each row. As a result, we are looking for a construction method allowing to generate the matrix on-the-fly, i.e. streaming of the matrix. We also proposed a modification of the algorithm to produce an indexable representation of k-mer presence/absence which is presented in chapter IV [page 61].

In this chapter, I present a method for constructing a k-mer matrix relying on disk-based k-mer counting techniques, as well as possible applications of such a representation. The methods presented here are implemented in `kmtricks`, a generic tool designed to work with k-mer matrices. The chapter V [page 81] is dedicated to its description and includes the implementation details that are not described in this section.

2 Construction

A k-mer matrix represents the abundance of k-mers across several samples and therefore relies on k-mer counting. The state-of-the-art k-mer counting algorithms have interesting properties for constructing the k-mer matrices and should not be seen as a trivial pre-processing step. Usually, the tools requiring matrix-like structure produce it by computing abundance tables individually for each sample before aggregating them in various ways, for example, using a hash table. In other words, they do not benefit from the intrinsic properties of the k-mer sets produced by k-mer counters. The following sections describe how a disk-based k-mer counting algorithm is extended to an efficient, parallel, and partitioned construction or streaming of k-mer matrices.

A Partitioning

As described in section I.3.B [page 37], disk-based k-mer counting relies on the divide-and-conquer paradigm. It consists in partitioning the samples to count each partition in parallel using a limited amount of memory. The idea is to use an intrinsic characteristic of the k-mers such that identical k-mers are always in the same partition, whatever their origin. For that, we use the concept of minimizer [122]. A minimizer K_m of a k-mer K is the smallest sub-sequence of size m from K that minimizes a given function. In our case, the minimizer is the smallest sequence in a lexicographical sense. Note that the minimizer is determined by examining both strands of a k-mer, i.e. we use the canonical minimizer.

In order to obtain optimal performances, the minimizer partitioning must be as uniform as possible. This means that when k-mers are dispatched in the partitions, the number of k-mers in each one must be equivalent. The partitioning enables a coarse-grained parallelization, i.e. one thread is in charge of one partition. An unbalanced partitioning leads then to non-optimal exploitation of the computing resources and higher memory usage.

The number of partitions is a user-defined parameter or is computed accordingly to the system and input data characteristics such as available memory, available threads, input size, etc. For example, a large number of partitions can be used when the available memory is limited to ensure that at least one partition can be counted at a time. Note that too few or too many partitions only affect performances, not the results. In the following, we denote the number of partitions by p.

The size of the minimizers is chosen accordingly to the k-mer size. However, the number of possible minimizers overgrows as a function of m (4^m). Thus the storage of the distribution function becomes expensive when m grows. Consequently, minimizer-based techniques usually limit the minimizer size. In our case $m \leq 15$.

To obtain a near-optimal partitioning, the frequency of each possible minimizer of size m is estimated by sub-sampling all input samples. The frequencies are then extrapolated with respect to the size of the inputs. Finally, the minimizers are distributed into partitions accordingly to their frequencies. In the end, we obtain a distribution function that maps a given minimizer, and therefore a given k-mer, to a partition.

As a side note, partitioning also enables the sub-sampling of input samples. In other words, it allows processing only subsets of k-mers (corresponding to a subset of minimizers) from each sample. This feature was helpful for a project I contributed to (see Section VI.3 [page 108]).

B Counting

Once the minimizer repartition is determined, the idea is to dispatch sequences in disk partitions (binary files) according to their minimizers. The first disk-based k-mer counting algorithms split the reads into k-mers before writing them into partitions [124, 125]. However, the storage

cost of k-mers composing a sequence is significantly larger than the storage cost of the sequence itself because k-mers are overlapping. Consequently, splitting reads into k-mers in partitions leads to significant and expensive disk usage. The problem was addressed by `KMC2` [126] which introduced the concept of super-k-mers. A super-k-mer is a sequence composed of a set of overlapping k-mers that share the same minimizer, i.e. all k-mers composing the super-k-mer belong to the same partition. Thus k-mers are represented as super-k-mers into partitions, saving disk space and I/O usage. In our case, the super-k-mers are composed of k-mers that share the same canonical minimizer and are stored in disk partitions in a bit-packed and compressed way (see Section V.4.A [page 100]). After dispatching, each sample is represented as p super-k-mer disk partitions.

The partitions are then counted in parallel in the following way. The super-k-mers are read sequentially and split into k-mers. The canonical forms of k-mers are put into an array in memory. The array is sorted using a generic IntroSort [128] algorithm following the lexicographic order. The abundances of k-mers are obtained by looking at the number of consecutive k-mer occurrences in the array. As super-k-mers, k-mers and abundances are finally written to disk in a compressed form (see Section V.4.A [page 100]). Note that each disk partition is sorted for free: an essential property regarding the final goal, i.e. the construction of a k-mer matrix. After counting, we obtain p sorted abundance tables for each sample.

Note that reading the reads and writing the super-k-mers are I/O-bound operations. In other words, the CPU resources are idle during these steps. In our multi-sample context, the resource can be used more carefully, i.e. a partition from a sample x can be counted when dispatching super-k-mers for a sample y. The idea is to avoid idle resources and always have super-k-mer partitions ready to be counted. The computing resources are thus shared between super-k-mer dispatching and partition counting, something that is impossible when counting sequentially with traditional k-mer counters. We call this procedure the joint k-mer counting. The distribution of resources impacts the performances and is controllable by a user-defined parameter for focusing on saving resources, i.e. using less memory and disk, or focusing on speed.

C Merging

At this point, we have partitioned, counted, and sorted partitions for all samples. The construction of the matrix can now start. The idea is to merge abundance tables from equivalent k-mer partitions, i.e. the partitions from each sample that correspond to the same subset of minimizers. Such merging can be achieved in linear time as inputs are sorted. We use a n-way merge algorithm to produce a single sorted set from n sorted sets. Given n sorted k-mer abundance tables, the merge is performed as follows. The first k-mer of each sample is inserted in a key-value min-heap where keys correspond to k-mers and values correspond to their abundances (see Figure II.1 [page 45]). As the input partitions are sorted, the min-heap property ensures

that the smallest k-mer is the first element of the heap. An abundance vector is constructed at each pass for all consecutive and identical k-mers. The next k-mers are then reinserted into the heap, and the process is repeated (see Figure II.1 [page 45]). If the abundance of k-mers is not required, a bit-vector is computed instead to represent the presence/absence of k-mers in samples. Finally, k-mers and vectors are written to disk. Note that each set of equivalent partitions, corresponding to one sub-matrix, is independent. In other words, they can be merged in parallel while using low memory because of the streaming process: the abundance tables are read and merged progressively.

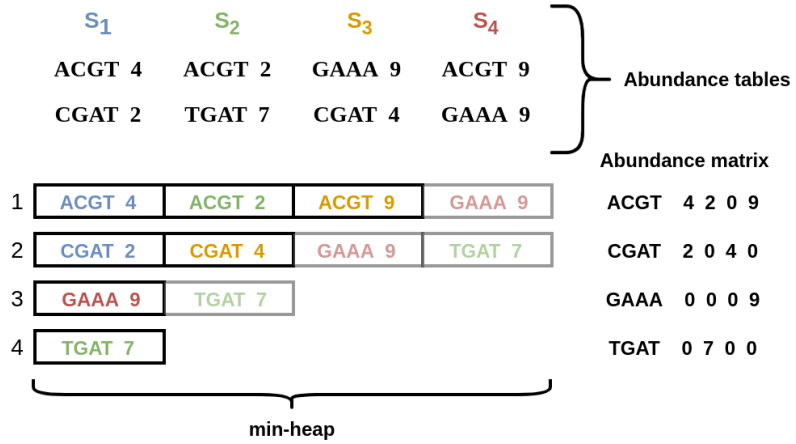

Figure II.1 – Example of merging abundance tables from 4 samples using a min-heap. The smallest k-mers from each table are inserted in the heap. At each pass, the abundance vectors are constructed from the identical k-mers in the heap. Note that the arrays do not represent the real layout of a min-heap and are purely illustrative. The min-heap properties just guarantee that querying the heap returns the current smallest element.

After merging, we obtain partitioned sub-matrices representing k-mers and their abundances in all samples. Partitioned storage is preferred since each sub-matrix is independent. However, the complete matrix can be generated by simply concatenating all sub-matrices. The so-concatenated matrix is a succession of sorted blocks of k-mers and abundances vectors. Thus, an additional merge can be performed to obtain a truly sorted matrix from the sorted sub-matrices, analogously to the merge of abundance tables. Note that the applications do not require the whole matrix but rather a "single view" on each row. In this case, the downstream computations can be applied on-the-fly and in parallel during the streaming of the matrix to save space and time. The tool presented in chapter III [page 51] exploits this streaming capability extensively. As a side note, our matrices allow log-time queries thanks to the sorted layout, although the primary goal is the construction/streaming, not creating a random access data structure.

3 Perspectives

A k-mer matrix is a generic representation with potential usages in various contexts. This section presents a new error k-mer filtering method enabled by the matrix representation and a non-exhaustive list of possible applications. The chapters III [page 51] and IV [page 61] focus on practical applications: differential k-mer analysis and sequencing samples indexing, respectively.

A Sequencing errors filtering

Sequencing errors lead to erroneous k-mers, which are relatively simple to identify by comparing their abundances to the sequencing coverage. Theoretically, a k-mer present in a single copy in the biological sample should have an abundance around the coverage, assuming uniform sequencing. A k-mer histogram representing the abundances distribution of k-mers from a human sequencing sample is presented in the figure II.2 [page 46], which highlights the segmentation between erroneous and true k-mers. The low-abundant k-mers are considered errors, and the filtering consists of simply discarding those with an abundance less than a given threshold.

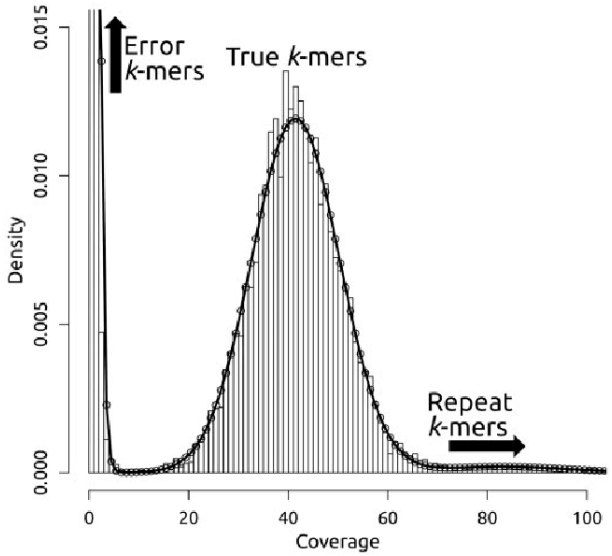

Figure II.2 – k-mer histogram of a human sequencing sample, reprinted from [129]. The abundances are distributed around the sequencing coverage. Rare k-mers (the first peak) correspond to sequencing errors. The separation between true k-mers and errors is obvious in such an ideal case.

In many cases, such filtration is adequate. However, threshold-based filtering is too stringent in some contexts, such as metagenomic sequencing. As a reminder, metagenomic sequencing corresponds to the sequencing of genetic material collected in environmental samples. A unique sample can contain a large amount of species, each represented by different and unknown num-

bers of individuals. From a sequence point of view, some sequences are weakly represented in the sample. Consequently, k-mer spectrums of metagenomic samples differ significantly from those obtained from a single-organism sequencing. Figure II.3 [page 47] is a k-mer histogram computed on a Tara Ocean metagenomic sample, highlighting the discontinuity of the abundances distribution. In such cases, threshold-based filtering is inefficient because the difference in abundance between erroneous and true k-mers cannot be stated.

In the following, we denote a k-mer that pass the filtering process, whatever it is, by the term *solid k-mer*. The others are the *non-solid k-mers*. Note that solid and non-solid k-mers do not mean true and erroneous k-mers. The k-mers are solid or non-solid accordingly to a given filtering technique and its parameters.

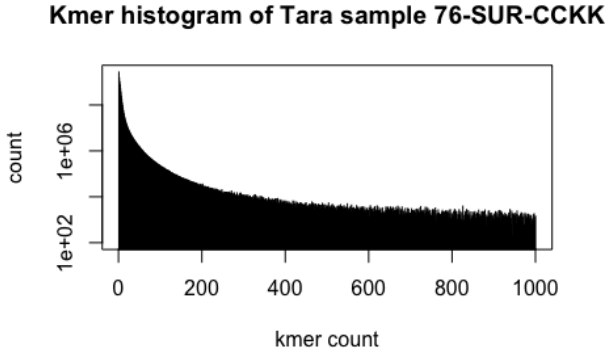

Figure II.3 – k-mer histogram of a Tara Ocean sequencing sample. No informative peaks are distinguishable in contrast to figure II.2 [page 46]. True and false k-mers are mixed.

When considering a collection of samples, we can expect some redundancy between them, especially in the context of sequencing projects which sequence related samples. For example, the Tara Ocean project collects marine samples at different geographic locations around the ocean, at various depths, and uses variable filter sizes to separate organisms. The analysis of these samples revealed, as expected, similarities between samples and allowed the identification of genomic provinces [63, 130]. Regarding the k-mer filtering, the idea is to exploit such redundancy to highlight the rare elements, i.e. the rare k-mers corresponding to the weakly represented sequences. In contrast to the one-sample-at-time counting, our matrix representation naturally gives the required information: the abundances of a k-mer in each sample of the collection.

Following this idea, we designed a straightforward procedure, the k-mer rescue, based on the following statement: if a k-mer is rare in a sample A but abundant in at least n other samples, perhaps it corresponds to a rare but true sequence in A, and we can consider it as a true k-mer.

The procedure is guided by 4 user-defined parameters:

— `hard-min` corresponds to the solidity threshold applied during the counting step. All k-

	Counted k-mers					Post filtration result				
	S1	**S2**	**S3**	**S4**	**S5**	**S1**	**S2**	**S3**	**S4**	**S5**
`soft-min`	**3**	**2**	**2**	**3**	**2**					
k1	2	~~0~~	2	5	2	1	0	1	1	1
k2	4	1	6	2	~~0~~	1	0	1	0	0

Figure II.4 – k-mer rescue procedure example, adapted from [131]. The example relates to 5 samples (S1 to S5) and uses the following parameters: `hard-min=1` and `share-min=3`. The `soft-min` parameter depends on the sample and is given in the table. Underlined values are higher or equal to `soft-min` (solid k-mers) and the strikeout values are lower than `hard-min`. Other values, between `hard-min` and `soft-min` are rescueable. The green value is rescued. **k1** has an abundance lower than 3 in S1 but it solid in at least 3 (`share-min`) samples (S3, S4, S5). Red values are not rescued. **k2** is rescuable in S2 an S4 but solid only in 2 samples. It is therefore discarded in S2 and S4 (but kept in S1 and S3).

mers with an abundance less than `hard-min` are discarded during counting stages and are not involved in the rescue. In other words, it corresponds to the threshold used in threshold-based filtering. Our experiments indicate `hard-min=1` is preferable to perform the rescue on the whole k-mer spectrum (see Section IV.3.C.2 [page 75]).

— `soft-min` corresponds to the solidity threshold applied during the merging. In other words, all k-mers with an abundance between `hard-min` and `soft-min` are considered rescue-able.

— `share-min` corresponds to the minimum number of samples containing a solid version of a k-mer. If a k-mer is non-solid in one sample, it must be solid in at least `share-min` sample to be rescued.

— `recurrence-min` is not directly relied on k-mer rescue. It allows discarding the rows for which the number of samples that contain a solid version of the k-mer is less than `recurrence-min`.

The k-mer rescue, illustrated in figure II.4 [page 48], occurs within the merging step, along with constructing the abundance vectors. It is applied on each k-mer of the matrix in the following way. During the construction of an abundance vector, the number of solid versions of the k-mer (with an abundance $\geq$ `soft-min`) is recorded, and the non-solid versions of the k-mer (with an abundance $<$ `soft-min`) are marked for a subsequent check at the end. If the number of solid k-mers is greater than `share-min`, the marked k-mers are considered solid. After the rescue, a row is kept when the new number of solid k-mers is greater than `recurrence-min`. This really straightforward technique has shown convincing results on a real metagenomic dataset from the Tara Ocean project. The results are presented in section IV.3.C.2 [page 75].

B Indexing

As described in section I.1.D [page 15], reads indexing usually relies on the pre-processing of each sample separately before a pooling stage to produce the final index, e.g construct a SBT from a collection of BFs. The matrix representation allows to consider all samples jointly and can therefore be seen as an input for k-mer indexing tools. This is particularly interesting for tools that follow the color-aggregative paradigm (see Section I.1.D.2.a [page 23]), which index a list of samples for each k-mer.

However, the matrix representation can be central to the indexing process and not just a pre-processing step to obtain the k-mer abundances. For that, the construction algorithm is extended to directly and efficiently produce an index, namely a partitioned BF matrix. Our method is significantly faster than existing constructions of such objects and leads to a fast end-to-end construction of a k-mer index while benefiting from the k-mer rescue. Chapter IV [page 61] presents the algorithm and the results obtained on real datasets.

C Analysis

Two main paradigms coexist in the context of comparing sequencing data: alignment-based and alignment-free. Frequently, the analysis of multiple sequencing samples is performed through the prism of the reference genome. In other words, each sample is processed with respect to the reference, and the individual results from each sample are linked to obtaining a global view. Beyond the reference biases and the cost of mapping, such analyses are not always applicable. High-quality references are available only for a very small subset of known species. As a result, experiments on non-model species require reference-free techniques. In addition, the development of sequencing leads to the growth of datasets with new collections containing more and more individuals or replicates, opening the door to extensive joint analyses. It is essential to have representations allowing the analysis of large cohorts composed of model or non-model species, such as k-mer matrices.

In a metagenomic context, the k-mer-based comparison of sequencing samples has already produced interesting results on large datasets [63, 64, 132]. The usual techniques used to compare a collection of metagenomic reads rely on read alignments. They consist of pairwise comparisons between reads or assigning reads to taxons by mapping on a reference database. These methods are extremely computationally expensive and not applicable to large datasets. The `Simka` [63] tool demonstrated that the k-mer-based analysis of metagenomes is strongly correlated with alignment-based strategy while requiring much less computational resources. The methods and tools developed in this work could be used to implement `Simka`-like strategies. An example is presented in section V.3 [page 91].

In the last decades, studies at the population level proliferated with the idea of characterizing the genetic profile of populations [133, 134]. The most popular studies are probably the

GWASs. A GWAS is a comparison of a set of known genetic variations with a large number of individuals in order to detect associations between variations and phenotypes. These studies rely on DNA microarrays to detect allelic differences between populations, controls and cases for example. Microarrays are chips containing many DNA spots, representing known variations, where the molecules from the studied samples can be fixed. Usually, known variations are SNPs, which are the most studied variations. GWASs rarely consider structural variations, which are actually involved in many diseases [135, 136]. In summary, GWASs rely heavily on known variations, reducing the scopes of the studies and making them unsuitable for non-model species. Furthermore, any significant variations absent from the microarrays will be missed. Recently, k-mer-based methods allowing GWAS-like studies have emerged [2, 46, 117]. Unlike microarray-based studies, these tools require only sequencing data, although some can consider a collection of SNPs for comparison purposes at the end of the analysis. In other words, k-mer-based techniques can be applied on any collection of sequencing samples which are now widely available with more and more large-scale sequencing projects. Such techniques also open the door to non-model population studies, for which the known genetic variations are limited or simply unknown. The starting and common point of these methods is the detection of differentially represented k-mers between two or more phenotypes, where a phenotype correspond to a pool of sequencing samples. We argue that an efficient k-mer matrix construction and representation could be helpful by significantly accelerating the computations. To illustrate that, we applied our matrix techniques in the context of differential k-mer analysis and combined them with state-of-the-art statistical models. In this context, we develop a new tool, `kmdiff`, allowing fast differential k-mer analysis between two cohorts: controls and cases, for example. The chapter III [page 51] is dedicated to this point.

Chapter III

Large-scale differential k-mer analysis

Preamble

This chapter is dedicated to applying k-mer matrices in the context of differential k-mer analysis, i.e. finding differentially represented k-mers between two cohorts of sequencing samples. We show that efficient and partitioned streaming of a k-mer matrix can significantly speed up a state-of-the-art method, `HAWK` (see Section I.2 [page 35]). We describe the method and present various benchmarks at different scales that confirm better performance while maintaining equivalent results. A benchmark also includes `kmerGWAS` (see Section I.2 [page 35]) to compare computational performances with a tool that implements a completely different statistical model.

Contents

1 Matrix-based differential k-mer analysis

A The `kmdiff` pipeline

In this section, I present the `kmdiff` pipeline and place it in perspective with `HAWK` to highlight the improvements. Indeed, The `kmdiff` implementation follows the same main steps and uses the same statistical foundations as `HAWK`. However, the implementation is entirely different and summarized in figure III.1 [page 52].

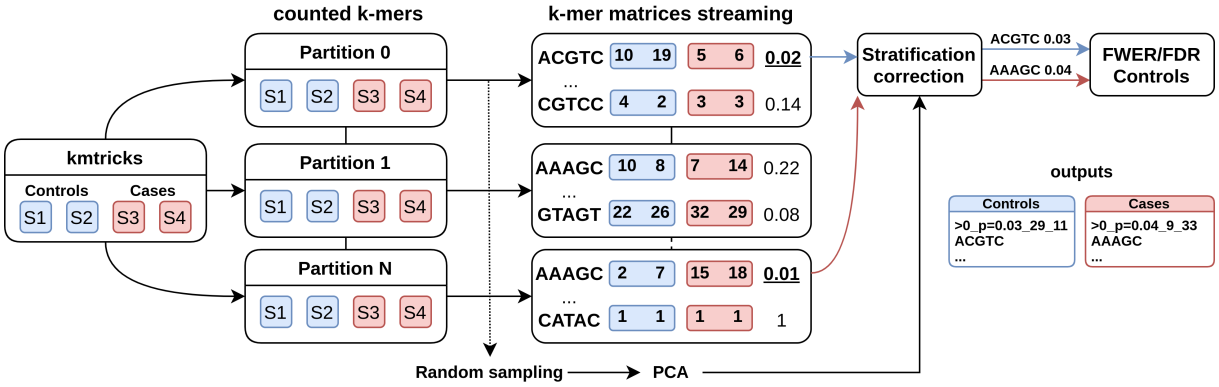

Figure III.1 – `kmdiff` pipeline overview on two cohorts composed of two samples (S1 and S2 for controls and S3 and S4 for cases). **A.** First stage corresponds to partitioned k-mer counting with `kmtricks`. **B.** Matrix streaming process during which k-mers are tested for significance and sampled to contribute to the PCA. **C.** Significant p-values are corrected to account for the population stratification and are then screened by common controlling procedures. Blue and red arrows represent examples of p-values over-represented in controls and in cases, respectively

The first step is common to all k-mer-based tools and consists in counting the k-mers in each sample. In `HAWK`, this operation is performed by `Jellyfish`, which is an in-memory k-mer counter. A slightly modified version allows to output k-mer abundance tables in a plain text format where the 2-bits representations of k-mers representation are written as ASCII integers. This choice is questionable for several reasons: **1.** `Jellyfish` is designed to produce a k-mer table allowing random access and not a simple list of counted k-mers, implying an extra overhead not encountered in some other k-mer counters. **2.** The plain text format significantly increases the size of the counted k-mers leading to more I/O operations. **3.** The last step of the counting consists of sorting the k-mers, a mandatory step to subsequently aggregate abundances of identical k-mers across samples. The data representation negatively impacts sorting since sorting binary integers is obviously more efficient than sorting ASCII integers. `HAWK` relies on the sort(1) Unix command to perform the sorting.

In `kmdiff`, the counting step is managed by the `kmdiff count` command that is basically a

wrapper around the `kmtricks` pipeline (see Section V.2.A [page 82]), ensuring the right parameters are used. `kmtricks` allows a partitioned counting as described in the section II.2 [page 42]. This gives many advantages for further computations: **1.** The abundance tables are partitioned, and their processing can therefore be easily parallelized. Indeed, the multi-sample partitioned counting ensures that identical k-mers across the samples belong to the same partition. **2.** As a part of the counting algorithm, the sorting is free compared to `HAWK`. **3.** The k-mers are stored in a binary and compressed format, reducing the expensive I/O operations. This results on a more efficient counting step as highlighted in section III.2 [page 55].

The second step, which we call the *diff step*, consists of aggregating abundances of identical k-mers from each sample to apply a statistical test on each abundance vector. The test proposed by `HAWK` is a likelihood ratio test assuming k-mer abundances are Poisson-distributed with rates θ_1 and θ_2 for controls and cases, respectively. The null hypothesis is then $H_0 : \theta_1 = \theta_2 = \theta$ and the alternate hypothesis is $H_1 : \theta_1 \neq \theta_2$. Rejecting H_0 means that the representation of a k-mer in both cohorts is significantly different. Let K a k-mer and A_1 and A_2 be the sums of its abundances in controls and cases. N_1 and N_2 denotes the total number of k-mers in controls and cases. The likelihoods are given by (details are available in Appendix 1 from [2]):

$$L(\theta_1, \theta_2) = \frac{e^{-\theta_1 N_1}(\theta_1 N_1)^{A_1}}{A_1!} \frac{e^{-\theta_2 N_2}(\theta_2 N_2)^{A_2}}{A_2!} \tag{III.1}$$

$$L(\theta) = \frac{e^{-\theta N_1}(\theta N_1)^{A_1}}{A_1!} \frac{e^{-\theta N_2}(\theta N <_2)^{A_2}}{A_2!} \tag{III.2}$$

The p-value corresponding to each k-mer is computed using the approximate χ^2 distribution of the likelihood ratio.

In `HAWK`, the diff step relies on a basic hash table. The number of k-mers is usually large, the k-mers are therefore loaded by small batches into the hash tables. As the k-mers are sorted and represented by integers, each batch corresponds to a range, ensuring that no k-mer is missed. The abundance vectors are then computed from individual abundances before computing the likelihood ratio. The main issue with this implementation is that processing each batch implies multiple expensive traversals of the whole hash table to populate it, process it, and dump it. In addition, an expensive thread synchronization mechanism is required while filling the table.

`kmdiff` uses the `kmtricks` library (see Section V.2.C [page 87]) to compute a k-mer matrix on-the-fly. The processing can be performed in parallel and low-memory thanks to the streaming capabilities of `kmtricks` and the independence of partitions. The likelihood ratio and the p-value are computed during the streaming for each k-mer. Significant k-mers with a p-value $< \alpha$, where α is the significance level, are directly written on disk in binary and compressed format. In `HAWK`, p-values are written in plain text, reducing both efficiency and arithmetic precision due to the multiple floating-point to ASCII conversion.

The next step corresponds to the correction of population stratification. In the study of phenotypes, a perfect composition of cohorts would be individuals that differ only in the phenotype of interest, which is naturally impossible. In association studies, there are systematic allelic differences within populations that are not related to the observed phenotypes but simply to the genetic history of individuals. A common variation in a population presenting a particular phenotype could be wrongly identified in such cases. Therefore, consideration of population stratification is essential in association studies to make appropriate corrections. As in `HAWK`, the detection of the population stratification is delegated to `Eigenstrat` [137, 138]. `Eigenstrat` uses a Principal Components Analysis (PCA) on a binary matrix representing the presence/absence patterns of randomly sub-sampled k-mers across samples. In `kmdiff`, the sampling is performed during the streaming of the matrix and the proportion of k-mers used for the PCA is defined by a user parameter. The results of the PCA is subsequently used to re-estimate the p-values of significant k-mers in the following way. As previously, the computation relies on a likelihood ratio test. Under H_0, the k-mer is not associated wtih the phenotype, i.e. abundance differences between cohorts are explained by the population stratification. The likelihoods are computed by fitting a logistic regression model against the population stratification and k-mer abundances for the null and alternate hypotheses, respectively. The corrected p-value is then computed thanks to a likelihood ratio test.

The corrected p-values must finally pass a final control to account for multiple testing since the number of tested k-mers is large, several billion. Consequently, the likelihood of rejecting a true hypothesis increases. Two main types of controls are usually used. Those which control the Family-wise error rate (FWER) rate and those which control the false discovery rate False discovery rate (FDR). The FWER is the probability of rejecting at least one true hypothesis. The objective of the controlling procedure is to ensure that the probability remains lower than a given threshold. The FDR corresponds to the expected proportion of false hypotheses that are rejected. As for FWER, the goal is to keep the FDR below a given threshold. The FDR procedures are less stringent than FWER-procedures at the cost of a greater number of type I errors, i.e. a greater number of false discoveries. `kmdiff` provides both types of controlling procedures. For controlling the FWER, two procedures are implemented: **1.** The Bonferroni correction [139] which defines a new significance threshold as $\alpha' = \alpha/N$, where N is the number of hypotheses and α is the significance threshold. **2.** The Sidak correction [140] is less stringent than the previous one. The new threshold is given by $\alpha' = 1 - (1 - \alpha)^{\frac{1}{N}}$. Regarding the FDR, only the Benjamini–Hochberg [141] procedure is implemented. It assigns a rank to each p-value sorted in ascending order. For a given p-value p and its rank r, the significance threshold is $\alpha\frac{r}{N}$. Note that its cost is slightly higher than others because it requires sorted p-values. Initially, `HAWK` only supported the Bonferroni correction. The Benjamini-Hochberg was added more recently in the last update [2].

Finally, the significant k-mers are written in two FASTA files corresponding to over-represented k-mers in controls and cases. The p-values are given in the FASTA headers.

B About the usage

Despite pretty results, using `HAWK` is particularly tedious. Indeed, in-house scripts are usually needed because it does not provide an end-to-end pipeline. Moreover, parameters are hard-coded, and each change requires a recompilation. Finally, several minor bugs make the experience unpleasant. For example, a crash can occur after hours of computation without any error messages due to a too long filename.

Consequently, our goal with `kmdiff` was to propose a more efficient implementation while remaining user-friendly. The whole `kmdiff` pipeline can be executed using only two commands without any in-house scripts.

In addition, `kmdiff` supports C++ plugins enabling fast and easy prototyping of new stream-friendly models while keeping the pipeline efficiency. This plugin system works the same way as `kmtricks` one (see Section V.2.D [page 88]).

2 Experiments

We performed experiments at various scales to evaluate the scalability and the qualitative results of `HAWK` and `kmdiff`. The goal was to propose a more scalable tool producing equivalent results. We included another tool, `kmerGWAS`, to compare the computational performances of the approach with a different statistical model.

A Benchmark environment

The experiments presented in this section were performed on the GenOuest platform on a node with 2x24 cores Xeon Gold 5220R 2.20 GHz and 128 GB of memory. The filesystem allowed 900 MB/s and 290 MB/s sequential read and write (average on 10 tests). Both tools support multi-threading and were executed using 20 threads. We used `kmdiff` v1.0.0, `HAWK` v1.7.0, and `kmerGWAS` v0.2 compiled directly on the node using GCC 11. `HAWK` uses `Jellyfish` to count k-mers in each sample. We used `Jellyfish` v2.3.0 with the patch provided by `HAWK` to obtain compatible outputs. `kmerGWAS` relies on GEMMA [142] v0.96.0, which is a toolkit for the application of linear mixed models. The running times and memory footprints are tracked with '`/usr/bin/time -f %E,%M`' except for `kmdiff` which tracks them internally. The peak disk usage is tracked using an in-house process that monitors disk usage every second.

All the tools were parameterized with a significance threshold of 0.05 and the Bonferroni correction was applied on each p-value for `HAWK` and `kmdiff`. In `kmerGWAS`, the correction is

performed by GEMMA. Regarding the k-mer counting, we considered only the k-mers with an abundance greater or equal to 2.

The resources to download the data and reproduce all experiments are available at https://github.com/tlemane/kmdiff-experiments.

B Ampicillin resistance

The ampicillin resistance dataset is composed of 241 paired-end RNA-Seq sequencing of *Escherichia coli* strains produced by Earle et al. representing 56 GB of gzipped FASTQ files. Among these strains, 189 are resistant to ampicillin, and 52 are sensitive. We consider that resistant and sensitive strains constitute the *control* and the *case* group, respectively. Several tools performing differential k-mer analysis used this collection as a benchmark dataset [2, 46]. As a side note, this study presents other groups of species and strains associated with different phenotypes that can be very useful for evaluating such tools.

On this dataset, `kmdiff` outperforms other tools in terms of computing resources except for the disk usage, which is slightly ahead of `kmerGWAS`. The detailed results are presented in table III.1 [page 57].

Regarding the `HAWK` comparison, `kmdiff` is 6x faster and uses 8x and 4.5x less memory and disk. The difference in memory usage is the consequence of `kmtricks`, a disk-based k-mer counter. Moreover, its streaming capability reduces the memory footprint compared to `HAWK`, which uses hash tables to merge k-mer across samples. The disk usage remains lower than `HAWK` thanks to the compressed representation of counted k-mers (see Section V.4.A [page 100]). The k-mers found by both tools are the same before the population stratification correction with 13,196,814 over-represented k-mers in cases and 16,804,587 in controls. The imprecise floating-point arithmetics and the non-deterministic sub-sampling of k-mers performed by the population stratification correction result in a slightly different number of k-mers at the end of the pipeline. Indeed, 4542 and 4591 pass the significance threshold in the control group for `HAWK` and `kmdiff` respectively. Both tools do not report any significant over-represented k-mers in the case group. This is expected because the individuals with the ampicillin resistance genes are in the control group. In this experiment, `kmdiff` was able to find 98% of k-mers reported by `HAWK`. The differences concern the k-mers with p-values very close to the significant threshold. They are sometimes slightly above or below, depending on the tool, due to imprecise floating-point arithmetics and non-deterministic sub-sampling during population stratification correction. To support the equivalence of the tools, the p-value cumulative distribution of significant p-values are presented in figure III.2 [page 57].

Regarding the `kmerGWAS` comparison, `kmdiff` is 2.2x faster while using less memory. The disk usage is slightly higher, although anecdotal on such a dataset.

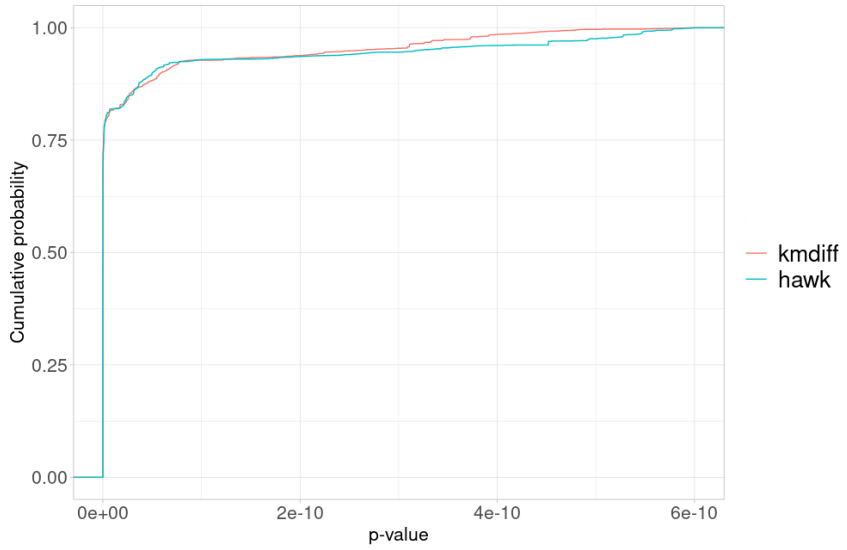

Figure III.2 – Cumulative distribution function of the p-values of the significant k-mers reported by both `kmdiff` and `HAWK` on the ampicillin resistance, reprinted from [144].

Ampicillin 189v52	Time (min)	Memory (GB)	Disk (GB)
`HAWK`	103 + 171	6.1 | 8.1	28.7 | 67.1
`kmdiff`	**11 + 36**	**2.7 | 2.7**	14.8 | 5.7
`kmerGWAS`	89 + 15	3.5 | 1.9	**10.3 | 3.4**

Table III.1 – Benchmarks of `HAWK`, `kmerGWAS` and `kmdiff` on ampicillin resistance dataset. For time, memory, and disk usage, we present two data points. The first one corresponds to the k-mer counting step. For `HAWK`, this step includes the sorting of k-mers, which is not required for `kmdiff`. The second one corresponds to the k-mer associations, including k-mers aggregation, statistical tests, and population stratification correction.

As stated in [143], the ampicillin resistance of the 189 strains is supplied by the Tn3 transposon, a mobile element from prokaryotes [145]. Consequently, we expect to find sequences from Tn3 transposon, which encodes 3 proteins: β-lactamase, Tn3 transposase, and Tn3 resolvase. The significant k-mers from each tool were assembled using `ABYSS` [146]. We queried the NCBI nr database [147] with the assembled contigs using BLASTX [148]. As expected, the majority of contigs correspond to the Tn3 transposon, whatever the tool. `HAWK` and `kmdiff` provide similar results. The assembly from `kmerGWAS` k-mers is more fragmented with shorter contigs. Details are given in table III.2 [page 58].

	N	L/T/R/U	Max	Sum	Mean
`kmdiff`	19	1/10/1/7	2270	5098	269
`HAWK`	17	1/10/1/5	2276	4891	293
`kmerGWAS`	26	1/17/1/6	1377	4315	166

Table III.2 – Summary of results from `kmdiff`, `HAWK`, and `kmerGWAS` on the ampicillin resistance dataset. **N** denotes the number of contigs after `ABYSS` assembly. **L/T/R/U** are the numbers of contigs associated with each category after BLASTX queries: β-lactamase, Tn3 Transposase, Tn3 resolvase, and Unknown, respectively. The unknown contigs always correspond to small fragments ($\approx$ 50pb). **Max** is the size of the longest contig. **Sum** is the sum of the contig lengths. **Mean** is the mean of the contig lengths.

Our benchmarks confirm that a paradigm shift in the implementation of the statistical methods proposed by `HAWK` resulted in better performances while maintaining the results. The scalability of `kmdiff` on larger datasets is evaluated in the next section.

C Scaling capabilities on human cohorts

To illustrate the scaling capabilities of `kmdiff`, we compared it to `HAWK` on human whole-genome sequencing datasets of different sizes. `kmerGWAS` was not included in this experiment because the early stages of the analysis require an amount of memory greater than the capacity of our computing node (128 GB). The cohorts are part of the 1000 Genome project [54] and correspond to two populations TSI (Toscani in Italia) and YRI (Yoruba in Ibadan, Nigeria). This dataset was previously used as a benchmark in `HAWK` publications [2, 3].

We built three benchmark datasets composed of 20, 40, and 80 individuals. The same number of individuals is used in both control and case groups. The larger experiments with 80 individuals was composed of 2.3 TB of gzipped FASTQ files. The parameters used are the same as for the ampicillin resistance, i.e. a minimum abundance of two, a significant threshold of 0.05, and a Bonferroni correction.

Whatever the monitored parameter, i.e. time, memory, or disk, `kmdiff` scales better than `HAWK` as shown in figure III.3 [page 59]. It is an order of magnitude faster, enabling the differential k-mer analysis on 80 human whole genome sequencing in a few hours. In contrast, `HAWK` requires more than six days. The detailed results are presented in the table III.3 [page 59].

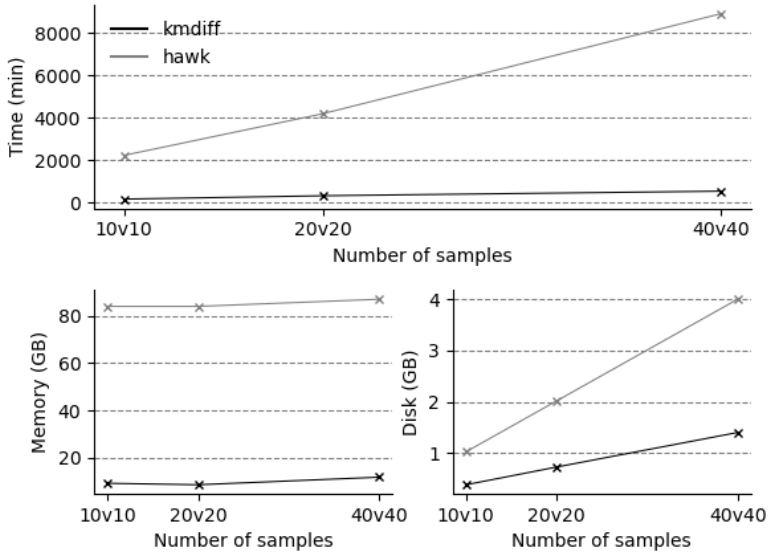

Figure III.3 – Scaling ability of `HAWK` and `kmdiff` on human cohorts. Both tools supports multi-threading and were executed using 20 threads. Compared to `HAWK`, `kmdiff` reduces computation times by 13-16x, memory usage by 8x, and disk usage by 2.6x

Human 10v10	Time (min)	Memory (GB)	Disk (GB)
`HAWK`	2040 + 186	84 | 21.2	1024 | 20.2
`kmdiff`	**129 + 37**	**9.3 | 2.4**	**380 | 4.7**

Human 20v20	Time (min)	Memory (GB)	Disk (GB)
`HAWK`	3916 + 277	84 | 28.5	2016 | 32.7
`kmdiff`	**241 + 83**	**8.7 | 3.26**	**726 | 9.34**

Human 40v40	Time (min)	Memory (GB)	Disk (GB)
`HAWK`	8319 + 592	87 | 48.6	3914 | 93.5
`kmdiff`	**418 + 122**	**11.9 | 6.2**	**1455 | 49.4**

Table III.3 – Benchmarks of `HAWK` and `kmdiff` on different scale datasets. For time, memory and disk usage, we present two data points. The first one corresponds to the k-mer counting step. For `HAWK`, this step includes the sorting of k-mers, which is not required for `kmdiff`. The second one corresponds to the k-mer associations, including k-mers aggregation, statistical tests and population stratification correction.

Chapter IV

Large-scale indexing

Preamble

In this chapter, we describe how k-mer matrix construction methods presented previously (see Chapter II [page 41]) can be extended to build matrix of BFs. We show that our method enables fast BFs construction, leading to efficient indexing. We compare our technique with state-of-the-art indexing tools, based or not on BFs, and evaluate the scalability on RNA-Seq sequencing indexing datasets of various sizes. In addition, we present end-to-end indexing of a real and large metagenomics dataset and show the contribution of k-mers rescue (see Chapter II [page 41]) on this type of data.

Our BF construction method relies on partitioning, which is not perfectly uniform. Consequently, the false positive rates slightly differ according to partitions. We show that the false positive rate of each partition remains close to the expected false positive rate of a classical BF, except for some outliers. Additionally, comparisons between our index and an exact index, i.e. without false positives, reveal highly correlated query results.

Contents

1 From k-mer matrix to Bloom filters matrix

A Fast Bloom filter construction

As discussed in section I.1.D.1.b.i [page 16], the classical construction of BFs from sequencing collections suffers from several issues. As samples usually contain a large number of k-mers (several billion), the required space to represent a BF is relatively high, i.e. several gigabytes per filter. Since each k-mer can be inserted at any position in the filter, the entire filter must be loaded into memory during the construction. Parallel construction of several filters is complicated and sometimes impossible because it requires a relatively large amount of memory. Parallel construction is nevertheless possible by inserting several k-mers simultaneously, at the cost of synchronization mechanisms required to avoid race conditions, resulting in extra overheads. These problems are related to the poor data locality of the BFs, i.e. we cannot predict the insertion location of a k-mer.

Finally, regarding k-mer BF, k-mer counting is always considered as a pre-processing stage allowing the subsequent construction of the BF. Both operations are always performed by different tools, and BFs are built from arbitrary k-mer sets, on which we cannot make any assumptions or predictions. In addition, this usually implies non-specific and non-optimal data representations, increasing the cost of I/O operations.

In summary, the classical construction of a k-mer BF consists in two steps: 1) An efficient k-mer counting performed by a generic k-mer counter, e.g. `Jellyfish` or `KMC`, 2) The insertion of k-mers in the BF. BFs are built sequentially for each sample following these two steps.

A simple observation simplifies this process: constructing a BF does not require any k-mer but only the insertion locations, i.e. the hash values. In other words, only hash values can be counted in the context of BFs construction. Considering the sorting-based counting algorithm (see Chapter II [page 41]), this small change has consequences on the BFs construction:

1. **The counted hash values are sorted.** Thus, the "random" insertions of k-mers become sequential insertions of sorted hash values, i.e. insertion from 0 to N, where N is the size of the BF, reducing the number of cache misses. To illustrate this, consider an example. The construction of an optimal BF to represent $N = 3e9$ elements with a false positive rate $\epsilon = 0.01$ requires a bit-array of $M = 28e9$ bits and $k = 7$ hash functions. The number of bits to insert is therefore $N*k = 21e9$. Considering 512-bits memory blocks (data cache line size), the bit-array can be split into $B \approx 54e6$ blocks. Thus, about 388 bits should be inserted in each block. Assuming hash values are independent, the probability of two consecutive insertions in the same block is $\frac{1}{B}^2$, leading to an important number of cache misses because each block is visited many times. In our case, the insertion of sorted hash values guarantees that each block is visited and loaded in the cache only once.
2. **The counted hash values are partitioned.** Each partition is independent allowing

parallel processing of sub-BFs in low-memory. In addition, the independence exempts from synchronization mechanisms. We move from a fine-grained parallelization, i.e. insert k-mers concurrently, to a coarse-grained parallelization, i.e. process BF partitions concurrently.

The following section presents our construction method, which allows building a collection of BFs as a BF matrix. It consists of a modification of the construction of the abundance matrix presented in the section II.2 [page 42]. Indeed, a BF matrix is a binarized and transposed abundance matrix, where the elements are hash values instead of k-mers.

B Partitioned Bloom filter matrix construction

The construction of the BF matrix relies on the same major steps as the abundance matrix. The complete pipeline is illustrated in the figure IV.1 [page 64] which represents both the construction of an abundance matrix and the construction of a BF matrix from a collection of sequencing samples. The specificities and differences are presented in the next sections. The presented methods are implemented in the `kmtricks` software; its implementation is described in the chapter V [page 81].

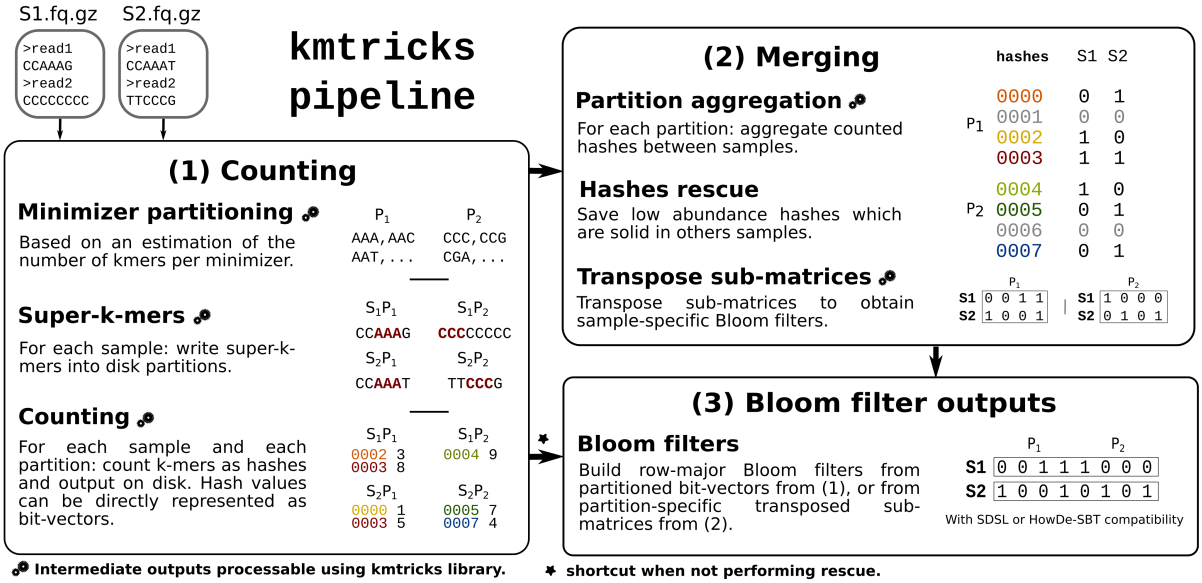

Figure IV.1 – `kmtricks` pipeline overview taking as input two samples, S_1 and S_2. Reprinted from [131].
(1) Counting: Partitions (here, P_1 and P_2) over minimizers (here of length 3) are determined by sub-sampling S_1 and S_2 and super-k-mers ($k = 5$) are then written on disk according to this partitioning. Bold red sequences are minimizers (AAA and CCC). Each partition is then counted and each k-mer is represented by its hash value. When performing the k-mer rescue, counted hashes are written on disk. Otherwise, each partition is directly represented as a bit-vector (the $\star$ symbol indicates that step (2) is skipped).
(2) Merging: Counted hashes from equivalent partitions are aggregated, and counts are binarized to produce a vector of BFs (i.e. a matrix of presence/absence bit-vectors, where row indices represent hashes). This matrix is filtered using the k-mer rescue procedure described in section II.3.A [page 46]. In order to build BFs, i.e. having samples as matrix rows, each partition-specific sub-matrix is transposed.
(3) Bloom filter outputs: a Bloom filter is built for each sample through concatenation of transposed sub-matrices (in those, each row corresponds to a sample). BFs can also be obtained from the first counting step if aggregation is not required. In this case, this corresponds to a concatenation of bit-vectors from (1).

1 Partitioning

In order to benefit from the partitioning and perform parallel construction, we use a variant of the BF that we call partitioned Bloom filters (pBFs). Such filter is a BF partitioned into P partitions with exclusive and consecutive hash spaces $h_p : \mathcal{U}_p \rightarrow \{p \times s, \ldots, p \times s + s - 1\}$ with $p \in [0, P)$ and $s = \left\lceil \frac{B}{P} \right\rceil$ (rounded up to a multiple of 8) where B is the user-defined BF size in bits. The partitioned hash space enables a coarse-grained parallelization at construction stages since only one pBF partition must be populated when counting a k-mer partition. The same

applies to the query since different partitions can be queried simultaneously without thread synchronization. In addition, memory usage is also reduced because only partitions under processing have to be loaded in memory instead of the whole filters.

2 Counting

The early parts of the counting are the same as in section II.2 [page 42] for the construction of the k-mer matrix. The samples are split into super-k-mers, which are written on disk according to their minimizers. The rest of the counting differs and consists in directly counting hash values instead of k-mers. In other words, the super-k-mers are cut into k-mers, which are this time directly hashed using xxHash [149], a robust and ultra-fast hash function. The hash values are then counted in the same way as the k-mers, i.e. using a sorting-based counting algorithm (see Section II.2 [page 42]).

The output of the counting stage depends on whether the k-mer rescue (see Section II.3.A [page 46]) is applied or not. Indeed, the rescue assumes that the counting information is available for all the samples and therefore requires hash value abundance tables for each sample. Using the k-mer rescue, the hash values and their abundances are compressed and written to disk (see Section V.4.A [page 100]) to be merged in the next step. If the k-mer rescue is not required, hash values are directly represented by a bit vector that already corresponds to a partition of a pBF (see Figure IV.2 [page 67]-(2).ⓑ).

Note that the sorting count algorithm involves counted and sorted hash values at the end of the counting stage. Thus, cache misses will be drastically reduced when these hash values are used to populate bit-vectors. Indeed, for a given partition p, the bits are set in a linear way from $p \times s$ to $p \times s + s - 1$, where s is the size of the partition in bits (see Section IV.1.A [page 62]).

At the end of this stage, we have P partitions of counted hash values or bit-vectors on disk for each sample.

3 Merging

As described before, the construction of pBFs depends on the k-mer rescue. Both ways, with rescue (see Figure IV.2 [page 67]-(1)) and without rescue (see Figure IV.2 [page 67]-(2)), rely on a file reorganization that takes advantage of an efficient copy method explained in section V.4.C [page 101]. At the end of this step, sample-specific pBFs compatible with SDSL [150], a widely-used succinct data structure library, or HowDeSBT are obtained.

a Without rescue

When the rescue is not required, the counting outputs are bit vectors representing hash values, one per partition for each sample. Thanks to the consecutive hash spaces, the creation

of a pBF consists of concatenating its P partitions in the correct order (see Figure IV.2 [page 67]-(1)).

b With rescue

When the rescue is performed, the counting outputs correspond to counted and sorted hash values. The partitions sharing the same minimizers across samples are aggregated using the same algorithm as the merging of counted k-mers (see Section II.2 [page 42] and Figure II.1 [page 45]). The abundance vectors are binarized in compliance with the rescue parameters. The main difference with merging counted k-mers of the abundance matrix construction is that all possible values are considered here. In other words, empty bit-vectors are added for missing hash values that correspond to the k-mers not seen in the partition. In this way, the hash values correspond to the index of the rows and are not stored, only the bit vectors are (see Figure IV.1 [page 64]-(2)). For each partition, we obtain a sub-matrix with $\lceil \frac{B}{P} \rceil$ presence/absence bit vectors.

At this point, a row represents the presence or the absence of a hash value in all samples. The structure is color-aggregative (see Section I.1.D.2.a [page 23]), and the BFs correspond to the columns of the matrix. The final index follows the k-mer-aggregative (see Section I.1.D.2.b [page 24]) paradigm and requires sample-specific BFs. For such bit-matrix, switching from the color-aggregative to the k-mer-aggregative representation corresponds to a bit-matrix transposition. The transposition allows to transform a matrix with hash values in rows into a matrix where each row corresponds to a sample and is a part of a one-hash pBFs (see Figure IV.1 [page 64]-(2)). Details about the bit-matrix transposition are given in section V.4.B [page 101]. After the transposition, we obtain P transposed sub-matrices. The sample-specific pBFs are then built by the horizontal concatenation of corresponding rows in the correct order from the first to the last partition (see Figure IV.2 [page 67]-(2).ⓔ).

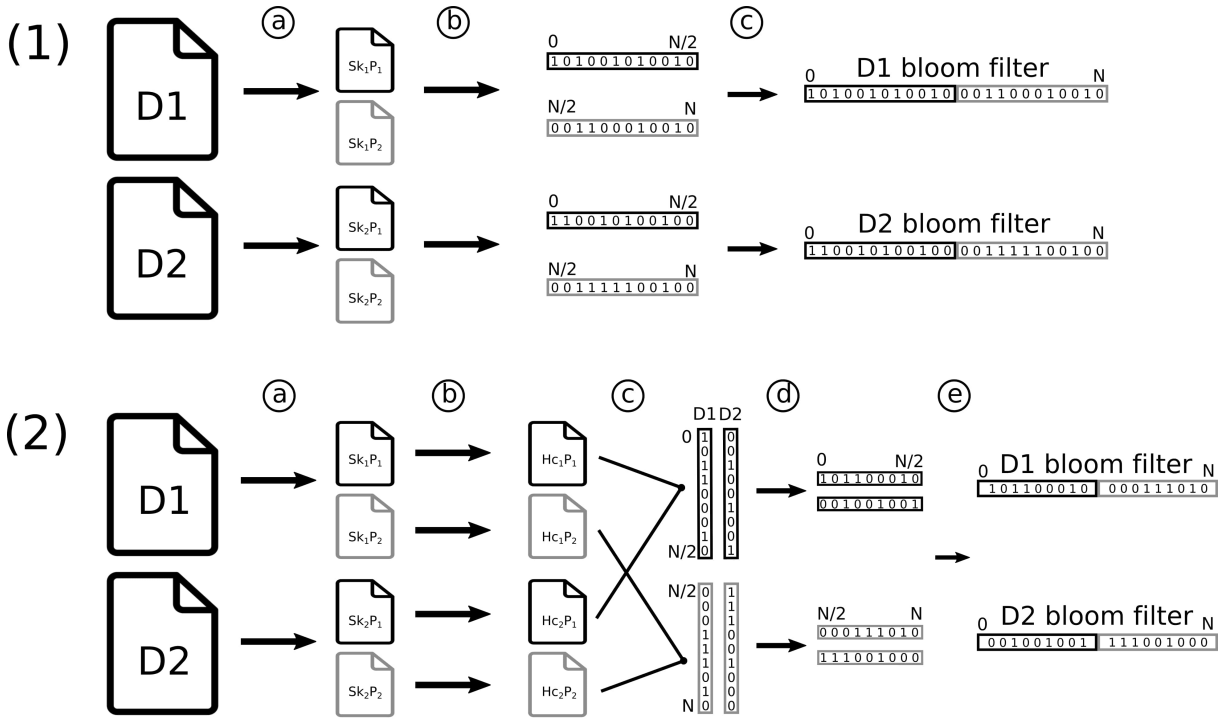

Figure IV.2 – Bloom filters construction pipeline with two samples D1 and D2, using two partitions: black (1) and gray (2). Reprinted from [131]. **Sk** and **Hc** denote super-k-mers and hash counted, respectively. **(1) Bloom filters pipeline without k-mer rescue**: ⓐ Divide sample into partitioned super-k-mers. ⓑ Split super-k-mers into k-mers before hashing them and counting hashes in partitions. For each partition, output presence/absence bit-vectors, i.e. partitioned Bloom filters. ⓒ Concatenate equivalent partitions between samples to obtain one Bloom filter per sample. **(2) Bloom filters pipeline with rare k-mer rescue**: ⓐ same as (1). ⓑ same as (1), but output hashes and their counts. ⓒ Merge and binarize (according to the rescue procedure, see II.3.A [page 46]) equivalent partitions to build one sub-matrix per partition with pBFs in columns. ⓓ Transpose sub-matrices to obtain pBFs in rows. ⓔ same as (1)-ⓒ. Reprinted from [131].

4 Indexing

Our collection of pBFs is already an index. However, the space and query time can be improved by aggregating them into a final structure. The indexing of pBFs is ensured by a modified version of HowDeSBT, the most space-efficient BF-based index (see Section I.1.D.2.b.ii [page 26]). We adjusted the HowDeSBT query algorithm to support our pBFs. Indeed, `kmtricks` outputs pBFs which correspond to BFs that are partitioned according to a minimizer repartition. At query time, minimizers must be computed for each k-mer to select the correct hash function, i.e. the correct partition.

In addition, we added the `findere` (see Section I.1.D.3.b [page 34]) query algorithm, reducing false positives and allowing smaller BFs, reducing at the same time the size of the final index.

2 Indexing a human RNA-seq collection

To evaluate the performance of our k-mer BF construction, we used a reference dataset used in indexing benchmarks, a collection of human RNA-seq sequencing. The original set is composed of 2585 samples. For storage space reasons, we used a subset of this collection for creating two small and medium-scale experiments with 100 and 674 samples, respectively. The RNA-Seq experiments have shown the scalability of `kmtricks` compared to other tools, which use various indexing techniques, based or not on BFs. The impact of partitioning on the false positive rates was also evaluated empirically on these samples. The detailed results are presented in the next sections.

In addition, a large-scale experiment was performed on another dataset and presented in the section IV.3 [page 71].

A Benchmark environment

The benchmarks presented in this section were performed on the GenOuest platform on a node with 2x10 cores Xeon E5-2660 v3 2,20 GHz and 200 GB of memory. The filesystem allowed 900 MB/s and 290 MB/s sequential read and write (average on 10 tests). All tools were compiled using GCC 10. The versions are given in the table IV.1 [page 69]. They support multi-threading and were executed using 20 threads. The running times and memory footprints are tracked with '`/usr/bin/time -f %E,%M`' except for `kmtricks`, which tracks them internally. The peak disk usage is tracked using an in-house process that monitors disk usage every second.

We benchmarked `kmtricks` against the classical construction of HowDeSBT using two k-mer counter, `Jellyfish` [118] and `KMC3` [127]. Here, the BFs construction consists in k-mer counting followed by the construction of BFs with the bundled tool `howdesbt makebf`. We also compared other approaches that use different indexing techniques, `McCortex + COBS` and `Squeakr + Mantis`. Like most read indexing tools, all the tools tested here are approximate methods, i.e. sensitive to false positives. Even if `Squeakr + Mantis` offers an exact index, it is more expensive and targets smaller datasets.

The resources to download the data and reproduce all experiments are available at https://github.com/pierrepeterlongo/kmtricks_benchmarks.

Tool	Version or git sha1
`kmtricks`	1.1.1
`HowDe-SBT`	2.00.02
`Jellyfish`	2.3.0
`KMC`	3.1.1
`McCortex`	1.0.1
`COBS`	1915fc0
`Squeakr`	0.6
`Mantis`	0.2.0

Table IV.1 – Versions or git hash of the tools used in the benchmarks.

B Performance comparisons

Table IV.2 [page 70] presents the construction times, memory, and disk usage of different tools on the human RNA-Seq samples. Data points are given for both constructions of the core data structures (BFs in our case) and the construction of the final index. The construction of the final index is not directly related to this work and is presented for the sake of completeness since it is required to perform queries. As a reminder, the creation of the final index from our pBFs relies on HowDeSBT.

Our method outperforms the classical construction of BFs in HowDeSBT in terms of time and memory, whatever the k-mer counter, while using more space due to the joint counting. The gap is more significant as the number of samples increases.

The other methods which use different indexing techniques are less efficient in this experiment. The relatively high construction time of `McCortex + COBS` is explained by the use of `McCortex`, which is originally an assembler used here to count and filter k-mers. `Squeakr + Mantis` are more efficient than `McCortex + COBS` but the construction of the core data structures remains 2x slower than `kmtricks`. `McCortex + COBS` and `Squeakr + Mantis` use also a larger amount of memory, approximately an order of magnitude.

In addition, the k-mer rescue, which requires an additional merge step, has a low impact on computation times. On the collection of 674 samples, the rescue results in a time overhead of 3%.

The scaling capability on terabyte-sized datasets is evaluated in the section IV.3 [page 71] and reveals a more significant performance gap between other methods and `kmtricks` on large-scale collections.

kmer counter (& bf creation)	Index	Time (min)	Memory (GB)	Disk (GB)
A : 100 RNA-seq (44 GB fasta.gz)				
`Jellyfish` (& `makebf`)	`HowDe-SBT`	147 + 21	13.2 | 2.6	55.1
`KMC 3` (& `makebf`)	`HowDe-SBT`	33 + 21	2.9 | 2.6	28.4
`McCortex` $k = 31$	`COBS`	256 + 67	27 | 1.5	327
`Squeakr`	`Mantis`	64 + 24	3.6 | 27.8	25.8
`kmtricks`	`HowDe-SBT`	24 + 21	3.6 | 2.6	45
`kmtricks` ^{R}	`HowDe-SBT`	26 + 21	3.4 | 2.6	46
`kmtricks` ^{R} $k = 31$	`HowDe-SBT`	20 + 21	3.6 | 2.6	50
B : 674 RNA-seq (961 GB fasta.gz)				
`Jellyfish` (& `makebf`)	$\emptyset$	3543	13.2	206
`KMC3` (& `makebf`)	$\emptyset$	1958	18.7	165
`kmtricks`	$\emptyset$	1033	24	247
`kmtricks` ^{R}	`HowDe-SBT`	1060 + 120	23 | 2.4	320

`kmtricks` ^{R}: `kmtricks` using rescue mode

Table IV.2 – Benchmarks on two human RNA-seq datasets composed of 100 and 674 samples, reprinted from [131]. Computations were done using 20 threads with $k = 20$. However as `COBS` supports only `McCortex`-file for $k = 31$, we also propose results for `kmtricks` + `HowDe-SBT` using $k = 31$. For Time and Memory, when two values are provided in a cell, the first corresponds to the pre-processing time (k-mer counting and possibly BF creation) and the second to the index construction. Memory and Disk correspond to the peak usage. Disk usage corresponds to the total required space to build the index, including temporary files, BFs and the final index. For `McCortex-COBS`, the disk usage corresponds mainly to the ctx files from `McCortex`.
Reprinted from [131]

C Empirical false positive rates analysis

The false positive rate of an usual BF is constant and depends only on the size, the number of hash functions, and the number of inserted elements. In the context of pBFs, the space is partitioned and the false positive rate also depends on the partitioning. One of the partitioning objectives is to distribute k-mers into partitions uniformly. In practice, perfect partitioning does not exist because the distribution depends on the k-mer contents of the samples. Consequently, the number of k-mers in each partition is not identical, resulting in variations of the false positive rates depending on the partition. It is important to verify the partitioning impact on the false positive rates distribution.

We studied the false positive rate per partition on the 100 RNA-Seq dataset. The false positive rate of each partition is compared to the expected false positive rate of a classical BF. Figure IV.3 [page 71] shows the false positive rate per partition for 15 samples. Despite outliers, most partitions have a false positive rate close to the expected rate (see Figure IV.3 [page 71]). Moreover, these outliers correspond to perfectly usable false positive rates from a query point

of view, as discussed in sections I.1.D.2.b.ii [page 26] and I.1.D.3 [page 34].

During the construction of the BFs, a matrix representing the false positive rate per partition is computed using the statistics collected in the merging step. It could be used later to correct the queries. For example, the false positive rates are used in HowDeSBT to correct the number of shared k-mers between a query and a sample. However, comparing the query results between an exact index and our approximate index (without query correction) shows good correlations (see IV.3.C.3 [page 77]). The comparisons were performed on a large-scale datasets, and the results are available in section IV.3.C.3 [page 77].

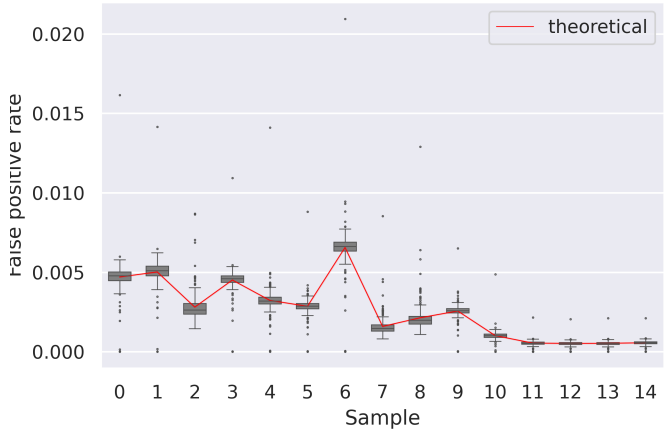

Figure IV.3 – Partition-dependant pBF false positive rate. Given 15 human RNA-seq samples, the distribution of false positive rates across partitions is shown as well as the theoretical false positive rate obtained without partitioning.

3 Scaling up to a large sea water metagenome collection

A The Tara Ocean Project

The Tara Ocean Foundation is an organization dedicated to ocean studies with research and education activities. The consortium organizes and implements large-scale metagenomic sequencing campaigns since 2003. Several sequencing expeditions have been conducted: Tara Arctic, Tara Mediterranean, Tara Microplastics, Tara Oceans and Tara Pacific. The data collected by these campaigns are difficult to analyze. They are complex metagenomic sequencings where each sample may contain millions of organisms and a wide variety of species. The characteristics of the sampling still give some information helping the downstream analyses. The sampling is performed at different depths using various filter sizes to separate the organisms, i.e. bacterial fraction, viral fraction, etc.

Here we focus on the Tara Ocean expedition [151] that took place from 2009 to 2012. The

objective was to explore the plankton ecosystems at a world-scale. The route and sampling stations are presented in figure IV.4 [page 72]. The collected data allowed a better understanding of the organization of oceanic ecosystems at a large scale. Currently, part of the Tara Ocean expedition data is public, and one can easily query some sequences using the Ocean Gene Atlas (OGA) platform [152, 153]. OGA allows to query the catalog of genes and assembled sequences from the metagenomic and metatranscriptomic data using raw sequences or Hidden Markov Models (HMM) profiles [154]. The query results correspond to different maps and plots which allow exploring various features such as the abundances of the input sequences in each sampling station or the impact of marine environmental features on these abundances. An example of possible results is presented in the figure IV.5 [page 73].

Unfortunately, the considerable amount of data and the mapping techniques used by OGA (BLAST [61], DIAMOND [155] and HMMER [156]) do not allow direct querying of the raw sequencing, and thus deprive of a large part of the data. The indexing of the raw reads would enable the query and the analysis of the whole dataset.

We decided to index the bacterial fraction of the Tara Ocean data. This represents 241 sampling stations composed of 712 read files, representing more than 6 TB of compressed data. The great diversity of organisms makes it a very complex dataset from the k-mer point of view with 266 billion of distinct k-mers ($k = 20$).

Because the data correspond to complex metagenomes, the Tara Ocean dataset is also a good candidate to benchmark the k-mer rescue procedure presented in section II.3.A [page 46]. The results are presented in section IV.3.C.2 [page 75].

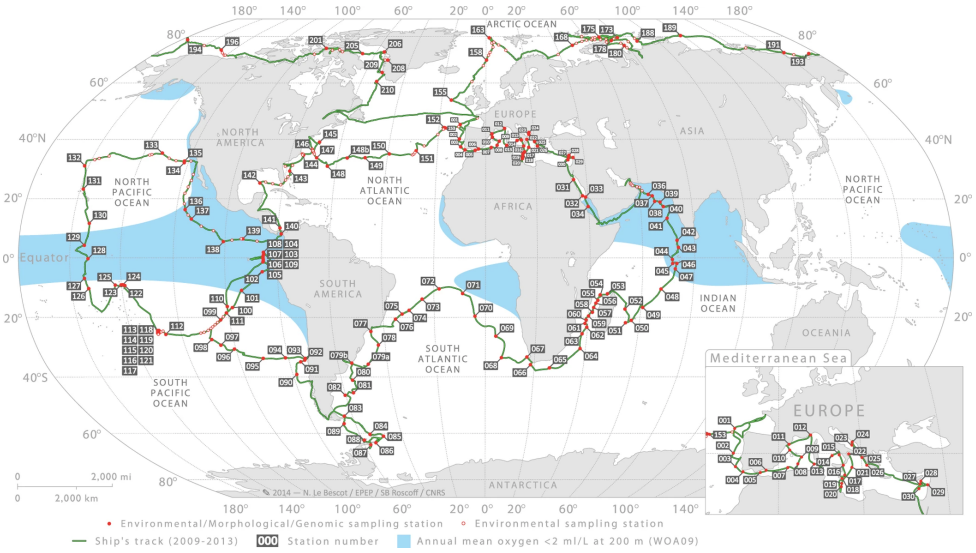

Figure IV.4 – Sampling route and stations of the Tara Oceans Expedition, reprinted from [157]

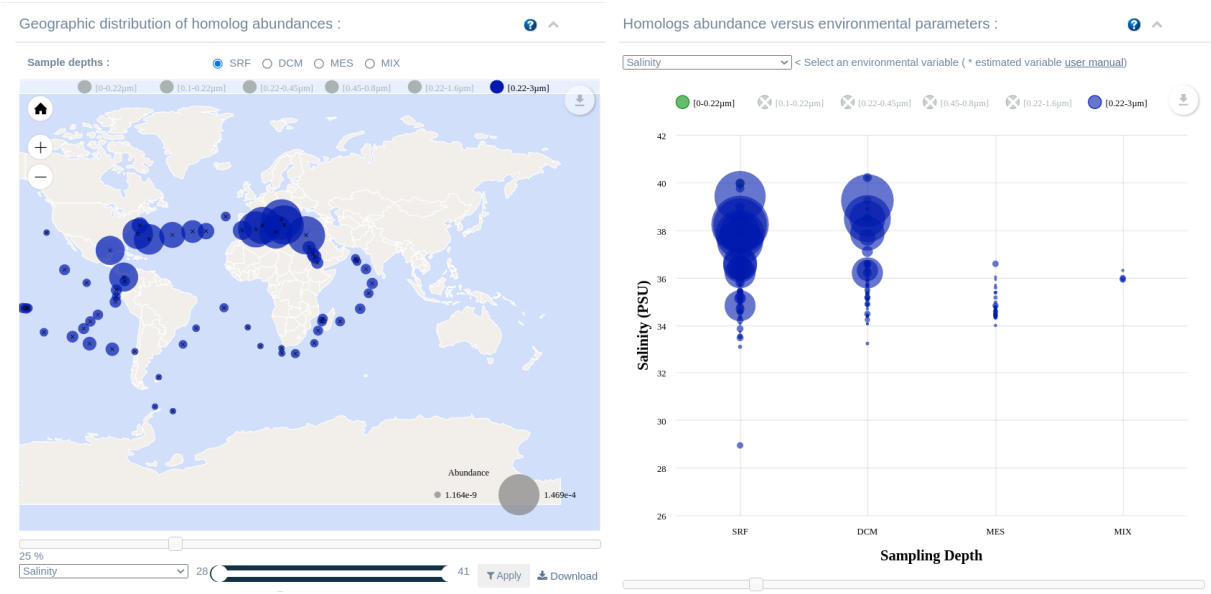

Figure IV.5 – Screenshot of an OGA query, obtained using the OM-RGC builtin example. https://tara-oceans.mio.osupytheas.fr/ocean-gene-atlas/.

B Benchmark environment

The computations were done on the Très Grand Centre de Calcul du CEA (TGCC) on a node with 2x64-cores AMD Milan@2.45GHz with 512 GB of memory. The average sequential read/write were 970MB/s and 216MB/s, respectively. Jobs are limited to 72h. The higher times are therefore extrapolated. We compared the `kmtricks` BFs construction to the construction of HowDeSBT which is the tool used to build the final index from the `kmtricks` BFs. Originally, HowDeSBT uses `Jellyfish` to count k-mers in each sample. Here we added a comparison with another k-mer counter, `KMC3`, which is usually more time and memory efficient than `Jellyfish`. The BFs were built with the `howdesbt makebf` command. Other tools used in the previous experiments were not included. `McCortex+ COBS` showed significantly longer construction time at smaller scale and `Squeakr + Mantis` ended in a segmentation fault on the large Tara Ocean samples. We therefore compared `kmtricks` to `Jellyfish + makebf` and `kmc3 + makebf`.

All benchmarks were run using $k = 20$, each BF size of 25,000,000,000 bits and 128 threads.

C Indexing the bacterial fraction of Tara Ocean data

1 Benchmarks

Using a similar amount of memory, `kmtricks` was able to construct the BFs in less than 24h for the 241 sampling stations. This corresponds to an improvement of the running time of 3.5x to 5.5x in comparison to `kmc3 + makebf` and `Jellyfish + makebf`, respectively. Although

equivalent for all tools, memory usage remains quite high. The main reason is the use of 128 threads, especially for `kmtricks` and `kmc3 + makebf`, leading to highly concurrent processing of partitions. In modern computational environments, such memory usage is still not really limiting and can be reduced by adjusting the parameters without changes in the final results. The original construction, `Jellyfish + makebf`, uses in any case a significant amount of memory to store the `Jellyfish` hash table.

Note that `kmtricks` achieves superior results by performing joint k-mer counting to enable the k-mer rescue, considering more k-mers. As described in section II.2 [page 42], k-mer rescue requires storing all k-mers or hash values on disk. As a result, `kmtricks` shows a higher disk usage (2x) than the other tools. In addition, the joint k-mer counting allows processing multiple samples simultaneously, requiring more temporary space. Using `howdesbt makebf` and whatever the k-mer counter, the BFs are created sequentially, and temporary files are deleted progressively. The peak disk usage, including the final BFs, is however 4x smaller than the compressed input data in the context of k-mer rescue. We argue that is probably not a bottleneck for users. The detailed results are available in table IV.3 [page 74].

	Time (min)	Memory (GB)	Disk (TB)
`kmtricks`	1433	83.4	1.5
`Jellyfish` [a] + `makebf`	$\approx 8071^b$	80.6^b	$\approx 0.8^b$
`KMC3` [a] + `makebf`	$\approx 5310^b$	100^b	$\approx 0.8^b$

[a]Stopped after 72h computation. [b]Extrapolated estimation.

Table IV.3 – Comparison of construction times between `kmtricks` and other methods on the Tara Ocean dataset using 128 threads. The `makebf` step corresponds to `howdesbt makebf` for Bloom filter creation from counted k-mers. The Memory and Disk columns indicate peak usage. `KMC3` and `Jellyfish` counted each sample independently and removed k-mers with abundance one; whereas by default, `kmtricks` performed joint k-mer counting and low-abundance *rescuing* (see Section II.3.A [page 46]) which kept some rare k-mers. Results for `Jellyfish` and `KMC3` are extrapolated as our cluster jobs are limited to 72 hours. For the disk usage, since `Jellyfish` and `KMC3` do single-sample counting, the peak disk usage corresponds to the BFs size plus the space required to count one station.
Reprinted from [131].

Using the so-created pBFs, HowDeSBT was used to build a query-able index. The computation was done in 1250 min with 165 GB of memory. The final index size was 533 GB which is less than 10% of the input size. On this index, the query of 10,000 metagenomic reads took 12 min and 11 GB of memory. Note that at this scale, the query time is bounded by the time required to load the index from disk to memory. As a result, querying one or 10,000 reads takes approximately the same amount of time.

2 Collection-aware k-mer filtering

As stated in section II.3.A [page 46], k-mer filtering usually consists in discarding a k-mer in a sample if its abundance is below a given threshold typically set to 2 or 3. Such a technique is effective when the sequencing coverage is high and uniform for all input sequences. In the context of metagenomic or RNA-Seq sequencings, the sequencing coverage is uneven due to weakly represented species and variable gene expressions, respectively. The sequencing errors are difficult to distinguish from rare but true k-mers. The k-mer rescue was developed to address this problem (see Section II.3.A [page 46]).

We compared the classic k-mer filtering technique, i.e. removing the k-mers seen once (`hard-min = 2`) with our rescue strategy. The rescue is applied using `hard-min = 1` in order to perform it by considering all k-mers. The `share-min` parameter was set to 1, i.e. a low abundant k-mer in sample is kept if it is solid ($\geq$`soft-min`) in at least one other sample. The `soft-min` parameter depends on the sample and is automatically computed such that the number of k-mers occurring `soft-min` times is less than 10% of the total number of k-mers. In other words, 10% of the least abundant k-mers are examined for a potential rescue.

The validation of the results relies on 3 metrics:

— err_{th} is the theoretical expected number of erroneous k-mers.

— err_{one} is the number of k-mers ocurring only once.

— $err_{unrescued}$ is the number of k-mers considered as erroneous after the rescue procedure.

The Tara Ocean samples were sequenced at the Genoscope, a French sequencing center. Benchmarks are performed on these sequencers using a benchmark species, *Acinetobacter baylyi*, with a well-characterized genome. Thanks to these experiments, we know the actual error rate of the machines used for the sequencing of Tara Ocean data. Being machine-specific, these measurements have higher accuracy than the values provided by the manufacturers. The k-mer error rate corresponds to the fraction of erroneous k-mers. It is computed using raw sequencings of *Acinetobacter baylyi* from each sequencer by counting the k-mers ($k = 20$) absent from the reference genome.

Three sequencing platforms produce the samples: HiSeq 2000 (222 stations out of 241), HiSeq2500 (8 stations) and GAIIx (4 stations). A station is composed of several samples and may sometimes be sequenced by different technologies. A total of 7 stations were removed from the analysis because 2 or 3 different technologies sequenced them. The k-mer error rates are 3.38%, 1.27%, and 9.27%, for GAIIx, HiSeq2000, and HiSeq2500, respectively. The estimate respective base error rates being 0.06%, 0.48% and 0.171%. The statistics used to compute these values are given in table IV.4 [page 76].

Technology	Pair	Reads	k-mers	Erroneous k-mers	base error rates
GAIIx	1	12037529	1378377359	25923501	0.0948
	2	12037529	1383555832	67308135	0.2490
	1&2	(2x) 12037529	2761933191	93231636	0.1715
HiSeq 2000	1	12037529	979185638	11029864	0.0566
	2	12037529	976152716	13879961	0.0716
	1&2	(2x) 12037529	1955338354	24909825	0.0641
HiSeq 2500	1	2202754	484673542	17295477	0.1815
	2	2202754	484727354	55029084	0.6007
	1&2	(2x) 2202754	969400896	89620038	0.4838

Table IV.4 – *Acinetobacter baylyi* raw sequencing statistics used to compute base and k-mer error rates.

Thanks to the k-mer error rates, we computed err_{th} for each sample. Other metrics, err_{one} and $err_{unrescued}$ are reported by `kmtricks` logs. We then compared the number of really discarded k-mer under the expected number of erroneous k-mers for both methods. These ratios are given by $err_{unrescued}/err_{th}$ for the rescue strategy and err_{one}/err_{th} for the classical filtering, i.e. discarding k-mers occurring once. The closer the ratio is to one, the better. As highlighted by the figure IV.6 [page 77], the number of discarded k-mers is close to the expected value with ratios close to one when using the rescue strategy. The average ratios are 1.01 and 9.12 for the rescue strategy and the classical filtering, respectively. This indicates that classical filtering, which discard k-mers seen once, is already too strict for this dataset by rejecting too many k-mers. From a matrix point of view, we expect 98 billion of wrong cells. In other words, a presence/absence binary matrix representing the presence/absence of all k-mers across samples contains 98 billions wrong bits (wrong 1s). The classical filtering discards 8-9x too many k-mers by removing 756 billion of cells, while the rescue strategy filters out 86 billion of cells.

On such datasets, k-mer rescuing seems to yield good results by allowing to overcome the classical filtering, which rejects an order of magnitude too many k-mers. Note that although these results look promising, it is essential to remember that there is no way to verify that the rescued k-mers are real ones. Indeed, the sequence content of this type of dataset remains largely unknown, and we lack access to the truth to make any assessments. The validation of these rare but shared k-mers from a biological point of view is one of the open questions brought by this work.

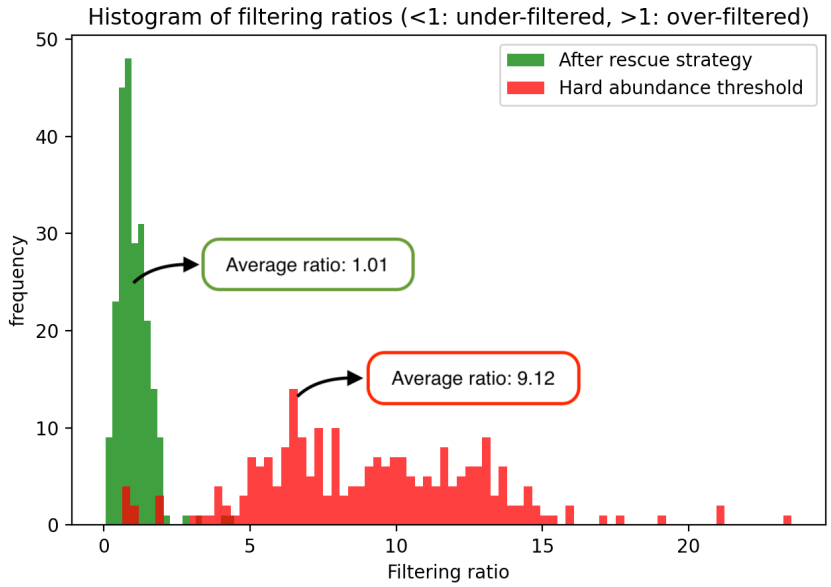

Figure IV.6 – Histogram of filtering ratios, reprinted from [131]. For each of the 241 Tara samples, the filtering ratio reports the number of filtered k-mers divided by the expected number of erroneous k-mers (the closer to 1, the better). The green (resp. red) histogram shows the filtering ratios of samples using the `kmtricks` rescue procedure (resp. using classical removal of k-mers occurring only once).

3 Queries

Like all AMQF-based indexes, our index is sensitive to false positives. The practical impact of false positives on the queries has been discussed in sections I.1.D.2.b.ii [page 26] and I.1.D.3.b [page 34]. As a reminder, approximate k-mer queries remain a good approximation of the truth when the queries are long enough. In addition, using query algorithms like `findere` theoretically allows working with high false positive rates while keeping good results.

We compared the query results of an exact index without false positives and an index produced by `kmtricks`. We performed 10,000 queries of size 100 on both indexes. For each query, we compared the reported shared k-mer rates for a sample showing a false positive rate of 20%, a really high rate. The objective was to verify the possibility of working with such a high rate, reducing index size, construction and query times. We compare the results for only one sample because the exact indexing cost of Tara samples, containing billions of k-mers, is really high.

Figure IV.7 [page 78] shows that 20% of false positives is too high. The percentage of shared k-mers is always strongly overestimated. However, using `findere` under the same conditions, the correlation between the exact index and the `kmtricks` index is remarkably high, as shown in figure IV.8 [page 78].

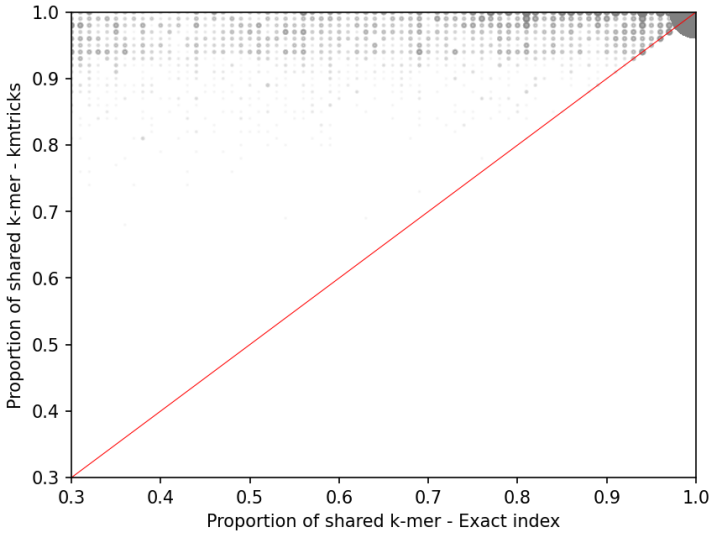

Figure IV.7 – Comparison of the proportion of shared k-mers between 10,000 queries and two indexes: an exact index and an approximate `kmtricks` index without `findere` algorithm. The red line represents the perfect correlation. The false positive rate of the approximate index is 0.2. The proportion of shared k-mers is always overestimated by the approximate index.

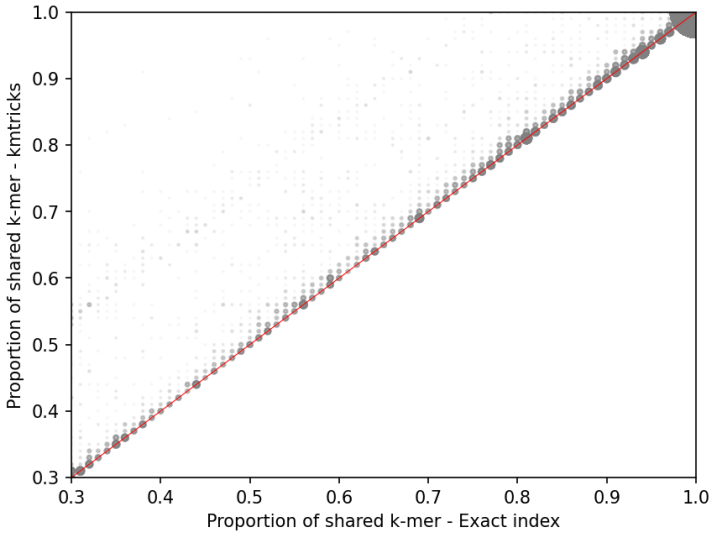

Figure IV.8 – Comparison of the proportion of shared k-mers between 10,000 queries and two indexes: an exact index and an approximate `kmtricks` index with `findere` algorithm. The red line represents the perfect correlation. The false positive rate of the approximate index is 0.2. Compared to IV.7 [page 78], exact and approximate indexes give close results.

In summary, we propose an end-to-end indexing pipeline capable of managing a dataset like Tara Ocean. The construction is achievable in a reasonable time, less than two days, while

considering about 9x more k-mers than traditional indexing tools. On the usage side, performing thousands of queries is possible in a few minutes.

The combination with `findere` algorithm reveals a great synergy, which can be exploited in two ways depending on the downstream analyses: 1. A better efficiency. Reducing the BF sizes results in more efficient construction and query while maintaining a usable precision, as shown by the previous experiment. 2. A better accuracy. Increasing the BF sizes while using `fnidere` could lead to a near-exact index at the cost of higher construction and query times.

The perspectives of future use of our indexing pipeline, implemented in `kmtricks` (see Chapter V [page 81]), are discussed at the end of the manuscript.

Chapter V

KMTRICKS: A k-MER MATRIX FRAMEWORK

Preamble

Since k-mer matrices are versatile representations with applications in various analyses, we are convinced that a generic tool dedicated to their construction and analysis could be helpful for the bioinformatics community. Consequently, we propose a tool, `kmtricks`, along with some utilities to work with such matrices. This last chapter presents `kmtricks` features and implementation details, along with some use cases.

Contents

1 Rationale

As stated in this manuscript, k-mer matrices go beyond the k-mer counting and can have applications in numerous bioinformatic analyses. Although other tools have already used the concept of k-mer matrices, there was no generic tool formalizing it. Each software had its approach, sometimes inefficient, to perform k-mer matrix construction. The primary purpose of `kmtricks` is to propose a tool allowing to build various types of k-mer matrices in an efficient, generic, and easy way. In addition, we accompanied it with different tools to facilitate its use and the exploitation of the results. In summary, `kmtricks` is a tool allowing joint k-mer counting on a set of FASTA/Q files, compressed or not, and exploiting the results via various utilities. In this section, I present its different components as well as concrete use cases. The implementation is available at https://github.com/tlemane/kmtricks.

2 Features

`kmtricks` is composed of 5 main components: **1.** A pipeline for easy end-to-end usage. **2.** A set of modules for step-by-step usage. **3.** A C++ library for the construction of new matrix-based tools. **4.** A C++ plugin system for easily extending its features. **5.** A formalized interface for sequence indexing. All of these components have different purposes and allow modular usages of `kmtricks`. For example, `kmdiff` (see Section III.1 [page 52]) needs a view on each row of the matrix. The whole representation of the matrix, expensive in terms of time and space, is usually not required, only the matrix streaming is. As a result, `kmdiff` uses the pipeline to obtain partitioned counted k-mers before using the library for parallel streaming of the sub-matrices. Another tool, decOM (see Section VI.3 [page 108]), relies on `kmtricks` to provide a pre-built matrix and to allow end-users to make matrix comparisons thanks to the modules.

A Pipeline

The pipeline is central part of `kmtricks`. It allows the joint execution of the different modules (see Section V.2.B [page 84]) in order to produce different types of structures such as abundance matrices, presence/absence matrices, or BFs. However, the intermediate structures used to build the final objects are interesting, independently of our use case. Consequently, the pipeline can be stopped at any stage for further usage of these intermediate structures in other tools or analyses. For example, the computation may stop at the counting stage to obtain abundance tables for each sample. In this case, `kmtricks` acts as a multi-sample k-mer counter. The results provided by the pipeline are exploitable using the utility modules or the library presented in sections V.2.B.2 [page 85] and V.2.C [page 87], respectively.

The chaining of the different modules within the pipeline is presented in the figure V.1 [page

83]. We can notably see the paths allowing to build an index using or not the k-mer rescue procedure.

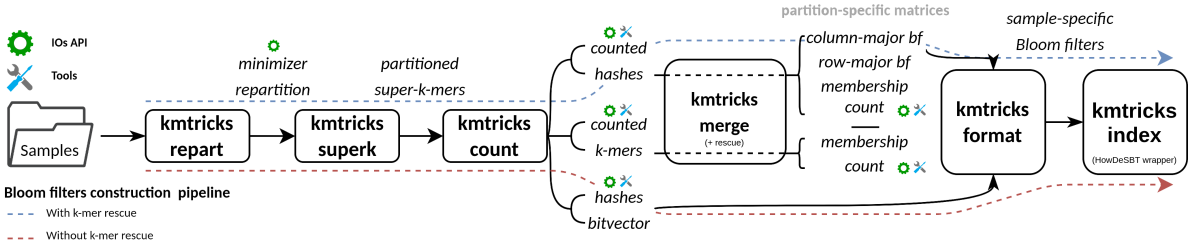

Figure V.1 – Overview of `kmtricks` modules. The blue and red arrows correspond to the BF construction pipeline with or without k-mer rescue, respectively.

At the end of the execution, all configurations and results are stored in a single directory corresponding to a run, e.g. usually a matrix. It contains all information relative to a run, potentially useful for downstream analyses. The directory structure is described in table V.1 [page 83].

build_infos.txt	Informations about kmtricks compilation
options.txt	Input parameters
run_infos.txt	Execution time and peak memory
hash.info	Storage of the partitioned hash function
kmtricks.fof	Copy of kmtricks input fof
config_gatb	Storage of the configuration
repartition_gatb	Storage of the minimizer repartition
minimizers	Storage of the list of minimizers
superkmers	Storage of super-k-mers
counts	Storage of counted hashes/k-mers
matrices	Storage of the hash/k-mer/Bloom matrices
partition_infos	Storage of partition statistics
merge_infos	Storage of merge statistics
filters	Storage of Bloom filters
fpr	Storage of false positive rates statistics
histograms	Storage of k-mer histogram
howde_index	Storage of the HowDeSBT index

Table V.1 – Structure of the `kmtricks` directory.

B Modules

1 Computation modules

As stated before, the `kmtricks` pipeline is composed of successive stages, producing various objects potentially interesting in other tools. As a result, we propose different command line modules that allow a step-by-step construction of matrices and that can also be used as building blocks by other tools or pipelines. Each module produces results that are then exploitable through the utilities (Section V.2.B.2 [page 85]) or the library (Section V.2.C [page 87]). Note that all modules presented in this section are linearly dependent.

The purpose of modules is to perform specific tasks not directly allowed by the pipeline, such as processing a single partition from a given sample. For example, they can be used to construct a matrix using only one specific partition for sub-sampling purposes. Such features was exploited initially by decOM (see Section VI.3 [page 108]) and finally integrated into the pipeline for convenience.

In addition, step-by-step computation enables the usage of distributed systems, i.e. simultaneously performing various stages on several computation nodes. SLURM [158] support was implemented and will be part of future releases after further testing.

kmtricks repart

The execution of `kmtricks` requires a particular runtime environment, i.e. a directory with a specific arborescence containing all the configurations and output files of `kmtricks`. This first module, `kmtricks repart`, is in charge of the configuration and must be used before any other `kmtricks` module. In addition, it computes the repartition scheme of the minimizers accordingly to the k-mer content of each sample. Outside the `kmtricks` framework, it can be used to compute a balanced minimizer distribution for further usage by another tool.

kmtricks superk

In the context of sequence processing, several tools use super-k-mers as an intermediate representation but do not allow to get them for later usage. The `kmtricks superk` allows splitting a sample into super-k-mers and storing them on disk in partitions accordingly to the pre-computed repartition scheme.

kmtricks count / kmtricks merge

`kmtricks count` and `kmtricks merge` allow to count k-mers and then merge abundance tables to obtain matrices. Since their specific features were included directly in the pipeline over time, they have no direct relevance for end-users. However, they are useful in the context of distributed computing and were used to implement SLURM support.

kmtricks format

`kmtricks format` is the last module of the pipeline. It is used only in hash mode to construct BFs. Independently of the pipeline, it allows to extract sample-specific BFs from the matrices and dumping them in different formats like the widely-used SDSL [150] format.

2 Utility modules

The utility modules are intended to be used on the data produced by pipeline/modules. They are mainly dedicated to output manipulations.

kmtricks dump

All `kmtricks` files are usually compressed binary files. `kmtricks dump` dumps them in a human-readable format for further reading by external scripts, for example. The files, such as the abundance tables and the matrices, are dumped in a sorted way since the counting algorithm implies sorted results.

kmtricks aggregate

Internally, the objects produced by `kmtricks` are stored following the partitioning scheme. In other words, the results are dispatched in many files. `kmtricks aggregate` aggregates these files to produce a unique and complete file in binary or plain text format. In addition, it can provide sorted outputs thanks to the counting algorithm, which implies that each partition is sorted. Some tools, like `HAWK` (see Section I.2 [page 35]), need a sorted k-mer abundance table in plain text format as inputs. Classically, abundance tables are dumped in plain text before being sorted. Using `kmtricks aggregate`, the tables remain stored in an efficient format. They are sorted and transferred on the fly to another tool, saving time and space. An example of a command line is presented below.

```
# Build the matrix
kmtricks pipeline --run-dir km_dir --file km.fof --hard-min 2 --until count
# Aggregate and stream abundance table of the ID sample to 'my_tool'
kmtricks aggregate --run-dir km_dir --count ID:kmer --format text --sorted --cpr-in | my_tool
```

kmtricks filter

`kmtricks filter` filters an existing input matrix according to the k-mer content of another new sample. Different outputs are possible: **1.** An abundance table that contains the k-mers present in the new sample but absent in the input matrix. **2.** A new matrix corresponding to the intersection between the new sample and the input matrix. In count mode, the new matrix contains an additional column corresponding to the abundances of k-mers from the new sample.

3. A column bit-vector representing the presence/absence of k-mers from the new sample in the input matrix.

3 Indexing modules

kmtricks index

The `kmtricks index` module is basically a wrapper around `howdesbt build`. From a `kmtricks` run containing a set of BFs, it is able to construct various HowDeSBT [114] indexes. It corresponds to the last step of the k-mer indexing pipeline and must be executed before any query.

kmtricks query

The `kmtricks query` module is wrapper around `howdesbt query` but it offers additional features. As described in section I.1.B [page 13], k-mer queries are heuristics. A match is defined by the number of shared k-mers between the query and an indexed sample. The meanings of matches with the same ratio of shared k-mers are not necessarily equivalent. Indeed, a match consisting of a large number of consecutive and overlapping k-mer hits does not have the same meaning as a match with an equivalent number of positive k-mer hits scattered along the query. In addition to the usual threshold, i.e. the proportion of shared k-mers (S_k), we implemented another threshold corresponding to the proportion of covered bases (S_b). The combination of both thresholds enables a more robust interpretation of the results, i.e. a low S_k and a high S_b indicates that k-mer hits are scattered along the query. To facilitate the interpretation, an optional "graphical" output is available and represents the shared bases on the query sequence by using '+' or '-' characters, as described below. The output indicates that the query *1* is found in the sample *D1* with 21% of shared k-mers and 58% of bases covered by at least one positive k-mer.

```
* [1]
[D1] ----------------------------++++++++++++++++++++++++++++++++++++ 0.21 0.58
```

In future releases, we plan to implement another threshold corresponding to the proportion of bases covered by at least N k-mers.

Finally, the query module implements the `findere` algorithm presented in the sections I.1.D.3.b [page 34] and IV.3.C.3 [page 77]. As a reminder, this algorithm drastically reduces false positives at query time without modifying the underlying index.

4 SOCKS interface

Nowadays, numerous tools offer k-mer indexing and propose different features and trade-offs, making the choice complicated from a user point of view. Recently, a resource has been

proposed to overcome this problem, the SOCKS interface (https://gi.cebitec.uni-bielefeld.de/research/panbench), which defines a set of features and standards for tools dealing with colored k-mer sets. In addition, a website with the documentation of various tools, automatic benchmarking, and online testing is planned but not yet available.

The SOCKS interface is supported by `kmtricks` via the `kmtricks-socks` command. Indexing a collection of samples is possible using a single and simple command with limited parameters. In the same spirit, the query procedure is streamlined and produces a standardized output. The documentation is available at: https://github.com/tlemane/kmtricks/wiki/kmtricks-socks-interface and https://gitlab.ub.uni-bielefeld.de/gi/socks.

C API

1 Sequence

`kmtricks` provides a templated k-mer class for an efficient 2-bits representation of k-mers. Efficient class specializations are available for small k-mers of sizes smaller than 32 or 64, encoded using 64-bits or 128-bits native unsigned integers, respectively. Larger k-mers are represented using an array of 64-bits unsigned integers. The right implementation is chosen at runtime thanks to recursive templates. Behind the scene, the code is compiled for each class of sizes and the right implementation is selected at runtime according to the input k-mer size. This mechanism is used in an example in section V.3 [page 91]. A standalone implementation is also available at https://github.com/tlemane/kmercpp.

2 I/Os

For efficiency reasons, `kmtricks` uses only in-house binary formats that can lead to interoperability problems. In complement of command line tools, a set of classes and functions are available for reading or writing each type of `kmtricks` file. The documentation is available at https://github.com/tlemane/kmtricks/wiki/IOs-api. Note that `kmtricks` supports also a standard format for storing k-mer sets: K-mer File Format (KFF) (see Section VI.2 [page 106]).

3 Matrix streaming

The matrix streaming API enables streaming and processing of the matrix on-the-fly from a collection of counted partitions. For example, it is used by `kmdiff` to merge and stream the different partitions in parallel and apply statistical tests (see Section III.1 [page 52]).

4 Task system

In `kmtricks` the elements of each partition are independent and can be processed in parallel. To facilitate parallelization, we provide a simple task system. It consists in a pool that will

execute the tasks in parallel according to a given number of threads and priorities. Moreover, it uses a callback system allowing the finished tasks to launch new ones. This task pool is used in `kmtricks` to execute the pipeline and manage the dependencies between the different components. An example is presented in section V.3 [page 91].

D Plugins

The matrices produced by `kmtricks` are sometimes very large. Their simple transformation into plain text followed by a reading from other tools, such as python scripts, leads to poor performance. The library allows to solve this problem but it requires a good understanding of C++. Moreover, it is necessary to consider I/O or multi-threading, even if many components are provided for that.

To facilitate the addition of new features, we have implemented a simple plugin system. The idea is to implement a class respecting a specific interface that will be able to apply an operation on each row of the matrix. In other words, this operation is applied directly when creating the matrix, reducing the running times.

The class is compiled as a shared library and linked at execution time using runtime dynamic linking, a technique that is widely used by famous libraries such as OpenMP. The required symbols are loaded at runtime and recompilation of `kmtricks` is therefore not required. The plugin interface is described in Implementation D.1 [page 89].

The user has to overload some functions to implement his custom features:

— `set_kmer_size` allows to set the k-mer size and have to be overloaded in the case of templated plugins.

— `configure` allows to configure your plugin. It takes a string passed from the command line with `--plugin-config`. Usually, this string should correspond to a path to a config file.

— `process_kmer and process_hash` are the functions applied on each line of the matrix. The return value indicates if the line is kept or not. The `count_vector` parameter corresponds to the abundance vector of the current line. This way, features like k-mer rescue can be implemented through the plugin system.

In addition to the class, some functions must be implemented to make the plugin loadable and are described below:

— `plugin_name` returns a string corresponding the name of the plugin.

— `use_template` returns 1 if the plugin is a templated plugin, 0 otherwise.

— `create0` constructs the plugin and returns it.

— `destroy` calls the destructor of the plugin.

Note that these functions must be marked as `extern "C"` to avoid the name mangling.

Implementation D.1: `kmtricks` plugin interface

```
class IMergePlugin
{
 using count_type = typename km::selectC<DMAX_C>::type;
 public:
  IMergePlugin() = default;
  virtual ~IMergePlugin() {}

  virtual void set_out_dir(const std::string& s) final {
   m_output_directory = s; }

  virtual void set_partition(size_t p) final {
   m_partition = p; }

  virtual void set_kmer_size(const std::size_t kmer_size) {
   m_kmer_size = kmer_size; }

  virtual void configure(const std::string& s) {}

  virtual bool process_kmer(const std::uint64_t* kmer_data,
                            std::vector<count_type>&  count_vector) {
   return true; }

  virtual bool process_hash(std::uint64_t h,
                            std::vector<count_type>& count_vector) {
   return true; }

 protected:
  std::string m_output_directory;
  size_t m_kmer_size;
  size_t m_partition;
};
```

To illustrate the usage of plugins, we consider an elementary example: One wants to build a k-mer abundance matrix and keep only the rows for which k-mers have an abundance greater than a given threshold in the first sample. For that, we need to overload the `process_kmer` function to make it return false when the condition is not satisfied. The corresponding source code is presented below.

Implementation D.2: Basic plugin

```
#include <kmtricks/plugin.hpp>

using count_type = typename km::selectC<DMAX_C>::type;
class Plugin : public km::IMergePlugin
{
 public:
  Plugin() = default;

  bool process_kmer(const uint64_t* kmer_data,
                    std::vector<count_type>& count_vector) override {
   return counts[0] > m_threshold; }

  void configure(const std::string& s) override {
   m_threshold = std::stoll(s); }

 private:
  unsigned int m_threshold {0};
};

extern "C" std::string plugin_name() { return "Plugin"; }
extern "C" int use_template() { return 0; }
extern "C" km::IMergePlugin* create0() { return new Plugin(); }
extern "C" void destroy(km::IMergePlugin* p) { delete p; }
```

Once written, the plugin must be compiled as a shared library. The `kmtricks` repository contains the necessary resources for automatically compuling plugins. The source code must be placed in the plugin folder and can then be compiled via:

```
install.sh -p -q
```

Assuming that the plugin source code is contained in a plugin.cpp file, the previous command will produce a shared *libplugin.so*. It can then be used in the following way:

```
kmtricks pipeline --plugin libplugin.so --plugin-config 42      \
                  --run-dir km_dir --file fof.txt --hard-min 2 \
                  --kmer-size 31 0
```

Once this command is executed, the final matrix in text format can be obtained using:

```
kmtricks aggregate --run-dir km_dir --matrix kmer > matrix.txt
```

In this example, only the abundances are required, i.e. the k-mer sequences are not considered. To benefit from the k-mer utilities, a templated plugin is needed. The documentation is available at https://github.com/tlemane/kmtricks/wiki/plugins.

3 Practical usage example

In comparative genomics, the comparison of k-mer content between pairs of samples is commonly used to reflect the similarities within an extensive collection. For example, such a technique was used by Simka [63, 64] on the Tara Ocean data and has allowed to identify genomic provinces around the oceans [159]. This work is based on the computation of ecological distances between k-mer sets such as the Jaccard Index [77], which could be easily computed from k-mer matrix. In this section, I present two ways to implement this type of analyses with `kmtricks` to illustrate the features of the library and the plugin system. Note that the code snippets presented in this section are designed to be succinct and easy to understand `kmtricks` functionalities. They are illustrative and not designed to be the most efficient.

A Using the API

In this section, I present the implementation of a standalone tool called `jaccard`, performing pairwise Jaccard index [77] computation on a set of sequencing samples. The Jaccard index is a statistic representing the similarity of two sets. It is given by $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$.

Using the `kmtricks` pipeline, we obtain partitioned abundance tables for each sample. The Jaccard index is then computed by merging partitions in parallel using the library to accumulate the counting information allowing the subsequent computation of the Jaccard index. Note that the whole matrix is not represented but only streamed on-the-fly from the abundance tables. The implementation is a combination of the pipeline and the library.

Such tool can be implemented as follows:

1. **Implement a JaccardObserver (see Implementation A.1 [page 93]).** The JaccardObserver class is passed to the `kmtricks` merging algorithm for collecting the counting information needed for the subsequent computation of the Jaccard index. The `process` function is called on each row of the matrix. The `compute_distance` is called at the end

of the matrix streaming and computes the Jaccard matrix representing the Jaccard index for each pair of samples.

2. **Implement a JaccardTask (see Implementation A.2 [page 94]).** Internally, abundance tables are partitioned. The JaccardTask class allows streaming of a sub-matrix from the equivalent partitions of each sample. The JaccardObserver collect the counting information. Since partitions are independent, multiple sub-matrices can be streamed in parallel. Note that the sub-matrices are not stored. Each row is computed on-the-fly from the abundance tables, used by the JaccardObserver, and finally forgotten. All parallel tasks share the same instance of the JaccardObserver. Consequently, a lock is used to avoid race condition in the `process` function. Such implementation was chosen to present a concise and simple example. Indeed, it leads to high thread contention and is therefore not optimal. To be efficient, each task should have its own JaccardObserver instance that will be aggregated at the end.
3. **Implement a JaccardEntryPoint (see Implementation A.3 [page 95]).** The JaccardEntryPoint acts as a templated main function used to select the correct k-mer implementation at runtime (see Section V.2.C.1 [page 87]). The tasks are created, one per partition, and executed in parallel using the Task pool (see Section V.2.C.4 [page 87]). When all tasks are done, the Jaccard matrix is computed.
4. **Implement the main function (see Implementation A.4 [page 96]).** The main function collects the user-defined parameters and runs the computations. Note that the k-mer size is a user-defined parameter, i.e. a runtime parameter, while the k-mer implementation is selected according to a template parameter, i.e. a compile-time parameter. The JaccardEntryPoint is therefore executed using the `const_loop_executor` to select the correct implementation at runtime. Behind the scene, this utility has the effect of compiling various representations of k-mers and choose the right afterwards one, according to the requested k-mer size. Thanks to this feature, `kmtricks` supports any k-mer size without recompilation and performance penalties. An upper bound (usually 256) is still defined at compilation time.

Implementation A.1: JaccardObserver

```cpp
template<size_t MAX_K, size_t MAX_C>
class JaccardObserver : public km::IMergeObserver<MAX_K, MAX_C> {
 using ctype = typename km::selectC<MAX_C>::type;
 public:
  JaccardObserver(size_t nb_samples) : nb_samples(nb_samples) {
   m_matrix.resize(nb_samples, std::vector<double>(nb_samples, 0.0));
   m_shared_kmers.resize(nb_samples, std::vector<uint64_t>(nb_samples, 0));
   m_solid_per_sample.resize(nb_samples, 0); }

  void process(km::Kmer<MAX_K>& kmer, std::vector<ctype>& counts) override {
   std::unique_lock<std::mutex> lock(m_mutex);
   for (size_t i=0; i<nb_samples; i++) {
    m_solid_per_sample[i] += counts[i];
    for (size_t j=i+1; j<nb_samples; j++) {
     m_shared_kmers[i][j] += counts[i];
     m_shared_kmers[j][i] += counts[j]; } } }

  void compute_distance() {
   for (size_t i=0; i<nb_samples; i++) {
    for (size_t j=0; j<nb_samples; j++) {
     uint64_t num = m_shared_kmers[i][j] + m_shared_kmers[j][i];
     uint64_t den = m_solid_per_sample[i] + m_solid_per_sample[j];
     if (den == 0) {
      m_matrix[i][j] = 1.0; }
     else {
      m_matrix[i][j] = 1.0 - (num / static_cast<double>(den)); } } } }

  void write(const std::string& path) {
   std::ofstream out(path, std::ios::out);
   for (size_t i=0; i<nb_samples; i++) {
    for (size_t j=0; j<nb_samples; j++) {
     out << m_matrix[i][j] << " "; }
    out << "\n"; } }

 private:
  size_t nb_samples;
  std::mutex m_mutex;
  std::vector<std::vector<uint64_t>> m_shared_kmers;
  std::vector<uint64_t> m_solid_per_sample;
  std::vector<std::vector<double>> m_matrix;
};
```

Implementation A.2: JaccardTask

```cpp
template<size_t MAX_K, size_t MAX_C>
using obs_t = std::shared_ptr<JaccardObserver<MAX_K, MAX_C>>;

template<size_t MAX_K, size_t MAX_C>
class JaccardTask : public km::ITask {
 public:
  JaccardTask(std::vector<std::string>& paths,
          std::vector<uint32_t>& abundance_min,
          size_t kmer_size,
          obs_t<MAX_K, MAX_C> obs)
    : ITask(0), m_paths(paths), m_ab_min(abundance_min),
      m_kmer_size(kmer_size), m_obs(obs) {}

  void exec() override {
    km::KmerMerger<MAX_K, MAX_C> merger(
      m_paths, m_ab_min, m_kmer_size, 1, 0);
    merger.merge(m_obs); }

 private:
  std::vector<std::string>& m_paths;
  std::vector<uint32_t>& m_ab_min;
  size_t m_kmer_size;
  obs_t<MAX_K, MAX_C> m_obs;
};
```

Implementation A.3: JaccardEntryPoint

```cpp
template<size_t MAX_K>
struct JaccardEntryPoint {
 void operator()(const std::vector<std::string>& idx,
            const std::string& dir,
            size_t kmer_size,
            size_t nb_parts,
            size_t threads,
            const std::string& output) {

  km::TaskPool pool(threads);
  std::vector<uint32_t> thresholds(idx.size(), 2);
  obs_t<MAX_K, 255> obs =
   std::make_shared<JaccardObserver<MAX_K, 255>>(idx.size());

  for (size_t p=0; p<nb_parts; p++) {
   std::vector<std::string> paths;
   for (auto& id: idx) {
    std::stringstream ss;
    ss << dir << "/counts/partition_" << p << "/" << id << ".kmer";
    paths.push_back(ss.str());
   }
   pool.add_task(
    std::make_shared<JaccardTask<MAX_K, 255>>(
     paths, thresholds, kmer_size, obs));
  }
  pool.join_all(); obs->compute_distance(); obs->write(output);
 }
};
```

Implementation A.4: Jaccard Main

```cpp
int main(int argc, char* argv[])
{
  if (argc < 7) {
    std::cerr << "Usage: jaccard <kmer_size> <nb_parts> <km_dir>"
          << " <idx_file> <nb_threads> <output>" << std::endl;
    exit(EXIT_FAILURE); }

  size_t kmer_size = std::stoll(argv[1]);
  size_t nb_parts = std::stoll(argv[2]);
  std::string kmdir = argv[3];
  std::string idx_path = argv[4];
  size_t threads = std::stoll(argv[5]);
  std::string output = argv[6];

  km::Timer jaccard_time;
  std::vector<std::string> sample_idx;
  std::ifstream in(idx_path, std::ios::in);
  for (std::string line; std::getline(in, line);)
    sample_idx.push_back(line);

  km::const_loop_executor<0, KMER_N>::exec<JaccardEntryPoint>(
    kmer_size, sample_idx, kmdir, kmer_size, nb_parts, threads, output);

  std::cerr << "Done in "
        << jaccard_time.elapsed<std::chrono::seconds>().count()
        << " seconds." << std::endl;
  return 0;
}
```

Once compiled, the `jaccard` program can be used on a `kmtricks` run containing abundance tables in the following way:

```
# Generate the abundance tables
kmtricks pipeline --run-dir jaccard_dir --file fof.txt --hard-min 2 \
                  --kmer-size 31 --nb-partitions 10 --until count
# Compute pairwise Jaccard indexes
jaccard 31 10 jaccard_dir idx.txt 20 jaccard_matrix.txt
```

B Using the plugin system

Unlike the previous example, we do not create an independent program. The idea is to implement the Jaccard index feature through a plugin that will be loaded at runtime and directly executed by the `kmtricks` pipeline. This requires less efforts because it allows to get rid of some steps, such as multi-threading management.

Using the plugin system, the Jaccard indexes can be computed in the following way:

1. **Implement a JaccardMatrix.** The JaccardMatrix class is a storage class accumulating information from the different plugin instances. Each plugin instance shares the same JaccardMatrix instance, in the same spirit as the JaccardObserver (see Section V.3.A [page 91]). The abundances collected by the different plugin instances, one per partition, are aggregated by the JaccardMatrix. Before its destruction, it computes the final Jaccard matrix representing all pairwise Jaccard indexes and writes it to disk.
2. **Implement a JaccardPlugin.** The JaccardPlugin overloads the `process_kmer` function of the plugin interface to update the JaccardMatrix from each row of the matrix. Note that the function returns *false* to discard all rows, i.e. the rows are built on-the-fly, and the whole matrix is never stored. As the JaccardObserver (see Section V.3.A [page 91]), the function uses a lock to avoid race conditions. This choice was made for the same reasons as in the previous example, i.e. simplicity and conciseness.

Implementation B.1: Jaccard Matrix

```
class JaccardMatrix {
 public:
  void compute_distance() {
   for (size_t i=0; i<nb_samples; i++) {
    for (size_t j=0; j<nb_samples; j++) {
     uint64_t num = m_shared_kmers[i][j] + m_shared_kmers[j][i];
     uint64_t den = m_solid_per_sample[i] + m_solid_per_sample[j];
     if (den == 0) m_matrix[i][j] = 1.0;
     else m_matrix[i][j] = 1.0 - (num / static_cast<double>(den)); } } }

  void write() {
   std::ofstream out(output, std::ios::out);
   for (size_t i=0; i<nb_samples; i++) {
    for (size_t j=0; j<nb_samples; j++) {
     out << m_matrix[i][j] << " "; }
    out << "\n"; } }

  void init(std::size_t nb_samples) {
    if (!ready) {
     m_matrix.resize(nb_samples, std::vector<double>(nb_samples, 0.0));
     m_shared_kmers.resize(
        nb_samples, std::vector<uint64_t>(nb_samples, 0));
     m_solid_per_sample.resize(nb_samples, 0);
     ready = true; } }

  void set_output_path(const std::string& p) { output = p; }
  ~JaccardMatrix() { compute_distance(); write(); }

 public:
  std::vector<std::vector<uint64_t>> m_shared_kmers;
  std::vector<uint64_t> m_solid_per_sample;
  std::vector<std::vector<double>> m_matrix;
  std::string output; bool ready {false};
};
```

Implementation B.2: Jaccard Plugin

```cpp
#include <kmtricks/io/fof.hpp>
#include <kmtricks/plugin.hpp>
class JaccardPlugin : public km::IMergePlugin
{
 public:
  JaccardPlugin() = default;

  bool process_kmer(const uint64_t* kmer_data,
                    std::vector<count_type>& count_vector) override {
   std::unique_lock<std::mutex> lock(m_mutex);
   for (size_t i=0; i<nb_samples; i++) {
    m.m_solid_per_sample[i] += counts[i];
    for (size_t j=i+1; j<nb_samples; j++) {
     m.m_shared_kmers[i][j] += counts[i];
     m.m_shared_kmers[j][i] += counts[j]; } }
   return false; }

  void configure(const std::string& s) override {
   nb_samples = km::Fof(this->m_output_directory+"/kmtricks/fof").size();
   m.init(nb_samples);
   m.set_output_path(this->m_output_directory + "/" + s); }

 private:
  static JaccardMatrix m;
  unsigned int nb_samples {0}; std::mutex m_mutex;
};

extern "C" std::string plugin_name() { return "JaccardPlugin"; }
extern "C" int use_template() { return 0; }
extern "C" km::IMergePlugin* create0() { return new Plugin(); }
extern "C" void destroy(km::IMergePlugin* p) { delete p; }
```

Once compiled, the plugin can be used in the following way:

```
kmtricksp pipeline --plugin libjaccard.so --plugin-config jaccard_matrix.txt \
                   --run-dir jaccard_dir --file fof.txt --hard-min 2       \
                   --kmer-size 31 --threads 20
```

The Jaccard matrix is finally available at 'jaccard_matrix.txt'.

Finally, the plugin system allows extending `kmtricks` features relatively simply while keeping the pipeline efficiency. The first use case of plugins was matrix filtering, even if `kmtricks` already supports basic filters. However, plugins could be used for more complex things, as shown by this example. The `kmdiff` tool (see Chapter II [page 41]) was implemented independently for convenience, but it could be only a plugin. Note that it supports a similar system allowing easy and fast testing of new statistical models.

4 Implementation details

A I/O

The objective of `kmtricks` is to allow multi-sample analysis by generating k-mer matrices from a relatively large number of samples. This operation being impossible to perform in memory on current machines, `kmtricks` relies heavily on disk. In the last few years, the speeds have been significantly improved with the Solid State Drives (SSD) and the Redundant Array of Inexpensive Disks (RAID) technologies. It is common to observe throughputs close to the gigabyte per second, both in reading and writing. Such speed allow building disk-based tools while keeping competitive performances. However, these speeds remain significantly lower than memory ones. In other words, I/O operations must be implemented carefully to guarantee optimal performance.

As described previously, the sequences can be encoded using a 2-bit encoding scheme which improves both space and processing time. The encoded k-mers are packed into one or more 64-bit unsigned integers according to their sizes and always stored with an alignment of 8 bytes allowing optimal read and write. As k-mers and super-k-mers can be encoded on several integers, the classical integer compression techniques were hardly applicable. Although theoretically possible, there are few implementations for large integers (≥ 64 bits). We decided to use a generic compression algorithm designed for speed to compress super-k-mers and the pairs k-mers/abundances. The k-mers from a given partition share a subset of minimizers and are lexicographically sorted. We can then expect a certain redundancy between close k-mers. This type of layout is perfectly adapted to the dictionary compression. We used the lz4 algorithm [160], a lossless compression algorithm from the LZ77 family [161] which supports streaming compression. Contrary to its competitors, it uses only a dictionary-matching stage without any further entropy stage [162].

The BFs construction pipeline allows a more efficient compression, in terms of space and time. In this case, the stored entities are hash values represented by 64-bit integers. As for k-mers, hash values are always sorted in `kmtricks` allowing the use of delta encoding. The deltas are then compressed using a PFOR [163] compression algorithm. The compression routines come from TurboPFor-Integer-Compression [164], a SIMD-accelerated library for integers compression.

Unlike hash values, the abundances are not sorted, and are therefore encoded via the ZigZag coding [165]. The result is compressed using the PFOR algorithm. Similar techniques could be used for small k-mers ($k \leq 32$) as they are encoded on 64 bits.

Sample-specific BFs are obtained by rearranging bit-vector matrices, as described in the section IV.1.B.3.b [page 66]. To ensure the efficiency of this operation, these vectors are not compressed (see Section V.4.C [page 101])

Compression of temporary and output files is optional and controlled by a parameter at runtime. The numerous uses of `kmtricks` have shown that running times are often lower when compression is enabled. This indicates that at least some parts of the pipeline are I/O-bound. Further benchmarks would be needed to clearly identify the affected parts before possible improvements.

B Bit-matrix transposition

Nowadays, bit-array manipulations can be greatly accelerated through vectorization using SIMDs. These techniques are used in `kmtricks` to perform efficient transposition of BF submatrices to obtain the sample-specific BFs required to build the final index. More specifically, the intrinsics `_mm_slli_epi64` and `_mm_movemask_epi8` [92] are used for bit-shifting and masking, respectively. These intrinsics work on 128-bit registers, allowing the transposition by block of 16 x 8 bits. Intrinsics working with larger registers (256 and 512 bits) are available on modern architectures and could allow an even more efficient implementation. However, benchmarks show that the matrix transposition is not a bottleneck in `kmtricks` and does not require additional developments for the moment.

C Concatenation of partitioned Bloom filter

The generation of sample-specific BFs consists of rearranging BF matrices via a copy of bit-vectors into individual files. Initially, each partition contains a part of the filter of each sample, stored in a row-major layout.

The classic file copy procedure involves both CPU-copy and Direct Memory Access (DMA) [166] copy, as well as a non-negligible number of context switching. When the CPU is involved in I/O operations, it is not available for perform other operations. DMA allows transferring data between devices without involving the CPU. As described in the figure V.2 [page 102], a classic copy consists in four copies and four context switches. The first system call causes a first context switch, and the data are copied from the device to the read buffer in the kernel space with a DMA copy. The data are then copied to the user buffer with a CPU copy and the system call returns, causing a second context switch. The writing of data into the socket buffer is performed by another system call (again, a context switch and a CPU copy). Finally, the socket buffer content is copied into the disk with a DMA copy and the system call returns causing the last

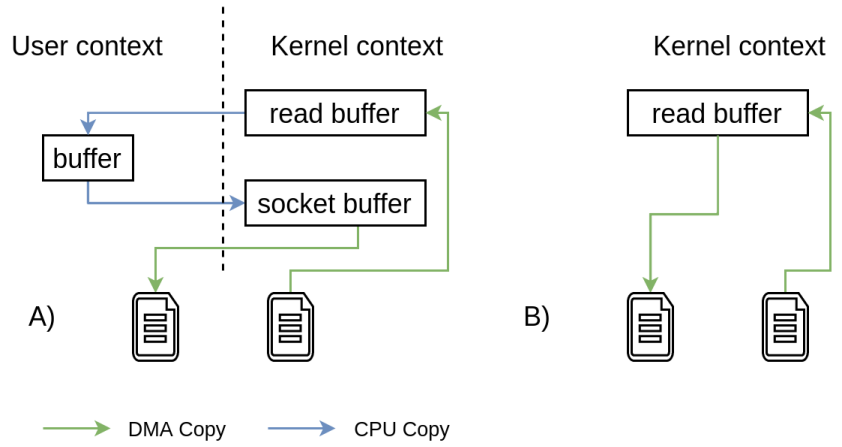

Figure V.2 – The difference between a classic copy (A) and a copy using the zero-copy (B) hardware feature.

context switch. Using zero-copy, the operation is completely delegated to the kernel. As a result, there are only two context switches, and the CPU copies are eliminated. Note that a CPU copy can still happen if the device does not support gather operations.

The Linux kernel provides several system calls allowing zero-copy as sendfile(2), splice(2) or copy_file_range(2). Unfortunately, there is no system call allowing direct zero-copy between files on macOS. As a result, some steps are significantly slower in the macOS version. However, it is probably possible to implement zero-copy using Memory Mapped I/O on such systems. In large experiments like Tara Ocean, the use of zero-copy reduced the running time of this step from a few hours to a few minutes.

Chapter VI

Other contributions

Preamble

During my thesis, I had the opportunity to collaborate on other projects, obviously related to the k-mer world. In ORI (Section VI.1), I participated to the development of the indexing aspect of the method. The other projects use `kmtricks` and have resulted in new works and developments. In KFF (Section VI.2), `kmtricks` was used to provide a proof of concept related to compact representation of k-mer sets. I contributed to implement one of the proposed representations in `kmtricks`. In decOM (Section VI.3), I implemented new features in `kmtricks`, related to sub-sampling and matrix comparisons. This chapter presents an overview of these collaborations.

1 Identification of isolated or mixed strains from long reads: a challenge met on Streptococcus thermophilus using a MinION sequencer [5]

In recent years, high throughput sequencing has enabled a significant breakthrough in the analysis of microbial communities without prior culturing. Numerous methods and tools have appeared for the characterization of bacterial samples from sequencing data with applications in many fields such as ecology, agri-food or even rapid diagnosis of clinically relevant bacteria.

There exists currently a stable and efficient set of tools allowing bacterial identification from short reads, but these fail when considering strain level identification because of the lack of long-range information. Regarding long reads, the number of tools is limited [167, 168, 169] and techniques used constantly evolve. Although these data contain long-range information, the identification at the strain level remains challenging because of the numerous sequencing errors present in long reads. If the identification at the genus or species levels is sufficient for some applications, it is sometimes a critical point in other fields such as health care with the differentiation of pathogenic vs non-pathogenic strains for example.

Consequently, this project aimed to develop a method for strain identification from long reads, and more specifically from MinION [170] reads. This sequencer produces relatively long

but noisy reads (see Section .1.A.3). In the context of strain identification, sequencing errors are problematic because they complicate the identification of similar strains. To address this issue, we use spaced seeds which are less sensitive to errors than k-mers [171]. Spaced seeds are sequences that allow wildcard positions that can be matches or mismatches.

The proposed method, called `ORI` [5], is a combination of indexing and Answer Set Programming (ASP) [172]. It is composed of three main steps as described in the Figure VI.1:

1. **Index construction**. A collection of genomes is used to build an index based on a modified version of HowDeSBT. The new version supports spaced seeds indexing and genome merging. Very similar strains present a high sequence identity making them hardly differentiable. These strains can be merged according to a user-defined threshold to avoid noisy identification. The spaced seeds support and the new merging features of HowDeSBT are my main contributions to this work.
2. **Query**. `ORI` is able to identify a strain mixture from a collection of MinION reads (usually, 4000 to 8000 reads are sufficient, depending on the dataset). First, the reads that are too noisy or too short are discarded. An affiliation matrix *reads* X *strains* is then constructed by querying all the reads against the index.
3. **Identification**. The objective of this step is to find a subset of strains that best explains the queried reads. A strain explains a read if the fraction of shared spaced seeds is greater than a given threshold. The reads explained by too many strains belong to the core genome are not considered. Finding the minimal subset of strains corresponds to a set cover optimization problem and is solved using ASP.

`ORI` was tested against other identification tools as `Kraken2` [169] or `StrainSeeker` [173]. `StrainSeeker` and `ORI` are the two methods that reach the strain level. However, `StrainSeeker` hardly scales up and can produce very different results depending on the number of input reads. In terms of time, `Kraken2` is still the best but fails to identify strains. This is explained by its minimizer-based method, which does not capture the slight variation between strains. `ORI` seems the most efficient for strain and mixed strains identifications.

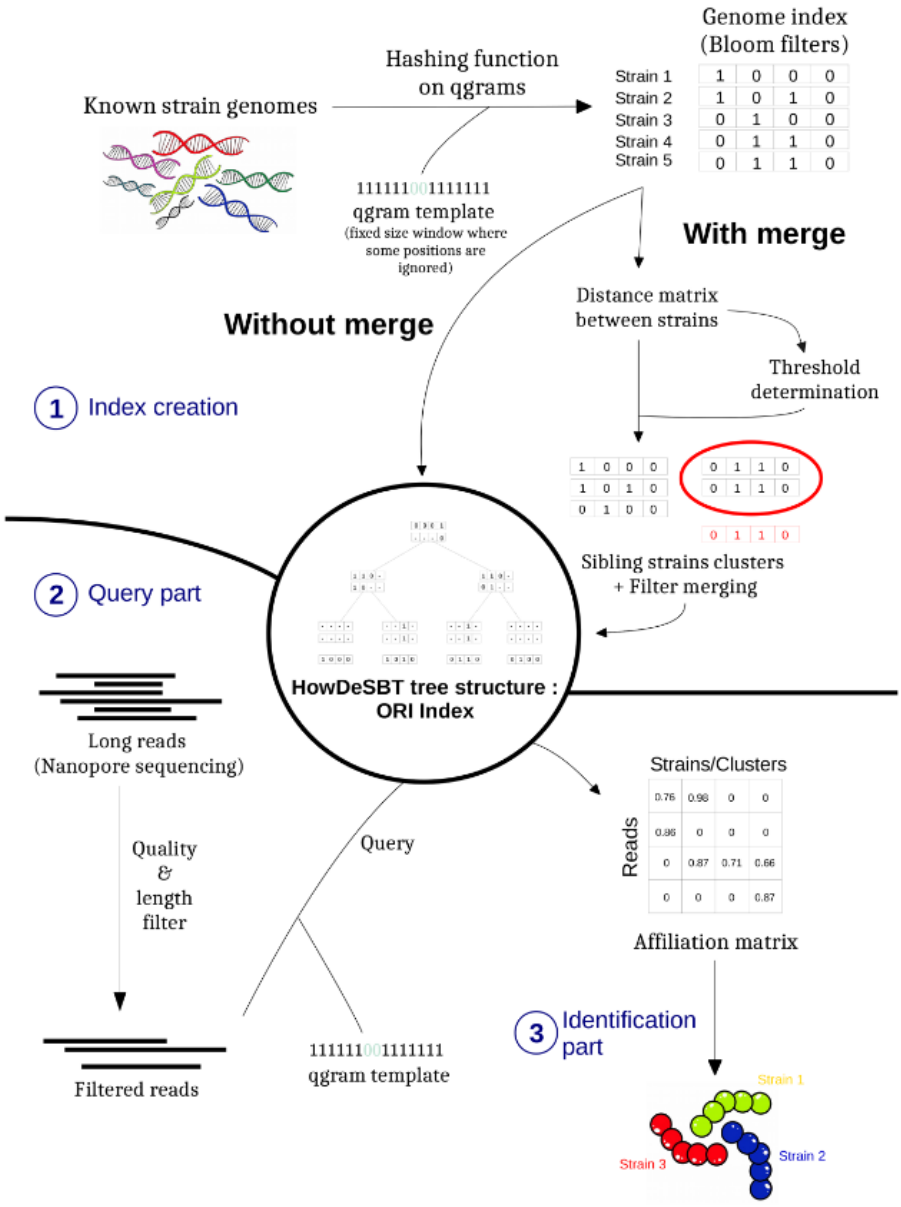

Figure VI.1 – Overview of `ORI`, reprinted from [5]. 1. Genomes are split into spaced seeds which are indexed using a Bloom filter. A distance matrix is computed for each pair of strains and similar strains are possibly merged according to a user-defined threshold. Bloom filters are then organized into a HowDeSBT index. 2. Long reads are filtered according to quality and length. Each read is then split into spaced seeds and queries are perfomed to build an affiliation matrix which represent the percentage of shared spaced seeds between reads and indexed strains. 3. An ASP script is used to determine the smallest subset of strains that best explains all the reads.

2 The K-mer File Format : a standardized and compact disk representation of sets of k-mers [6]

As described in this document, many analyses and tools are based on k-mers. Curiously, there was no standard format for storing k-mer sets, with or without abundances. As a result, each tool uses its own format, from simple plain text to complex binary format, causing interoperability and efficiency problems.

Standard formats are proving to be really useful and efficient in bioinformatics. The most famous example is probably the SAM and BAM formats [174], which are used to represent sequence alignment. In addition to the specification, these formats offer many libraries and tools [175] for analysis and file handling. This rich collection of features, as well as the efficiency of the format justify their ubiquity in sequence bioinformatics.

In the same spirit, we propose a binary file format for storing k-mer sets, the K-mer File Format (KFF), along with a library (C++ and Rust) and a set of tools allowing the processing of files. KFF stores k-mers in binary format and takes advantage of overlaps and redundancy between sequences to represent k-mer sets compactly without compression, i.e. bases are simply packed thanks to overlaps. It allows storage of variable-sized sequences consisting of overlapping k-mers with an additional space-optimization when the k-mers constituting the sequence share the same minimizer (super-k-mers for example). A description of the format is available in figure VI.2.

As `kmtricks` contains basic components related to minimizers and super-k-mers, we used it to produce the super-k-mer representation. This required a modification of the counting step (see Section II.2), which I have adapted as follows: the counted k-mers are no longer stored on disk but in an abundance table. Then, the super-k-mers are read again and associated with abundance vectors thanks to the abundance table. When a k-mer is absent from the table, it has already been seen in another super-k-mer. The current super-k-mer is then split into new super-k-mers containing only new k-mers. This representation is sub-optimal because it does not consider maximal overlaps and processes super-k-mers in the order seen in the reads. It remains a reasonable compromise between a naive representation and a near-optimal representation which is expensive to build as spectrum-preserving string set [67, 66].

To illustrate the KFF efficiency, we have benchmarked different formats on various sequencing samples: plain-text encoding, KMC format, KFF naive (one k-mer per block, no usage of overlaps), KFF sk (stored as super-k-mer from `kmtricks`) and KFF spss (stored as a string-preserving string set from ESS-Compress [67]). The results (see Table 1 in [6]) on various sequencing samples show good performances in terms of space, both for raw (KFF spss) and minimizer (KFF sk) sections. However, the most advanced representations are more time-consuming. The running time to obtain the k-mer set from a *G. gallus* sequencing sample in KFF format was 9

min for the naive version, 113 minutes for sk, and 900 minutes for spss.

All resources related to the project are available on github at https://github.com/Kmer-File-Format/.

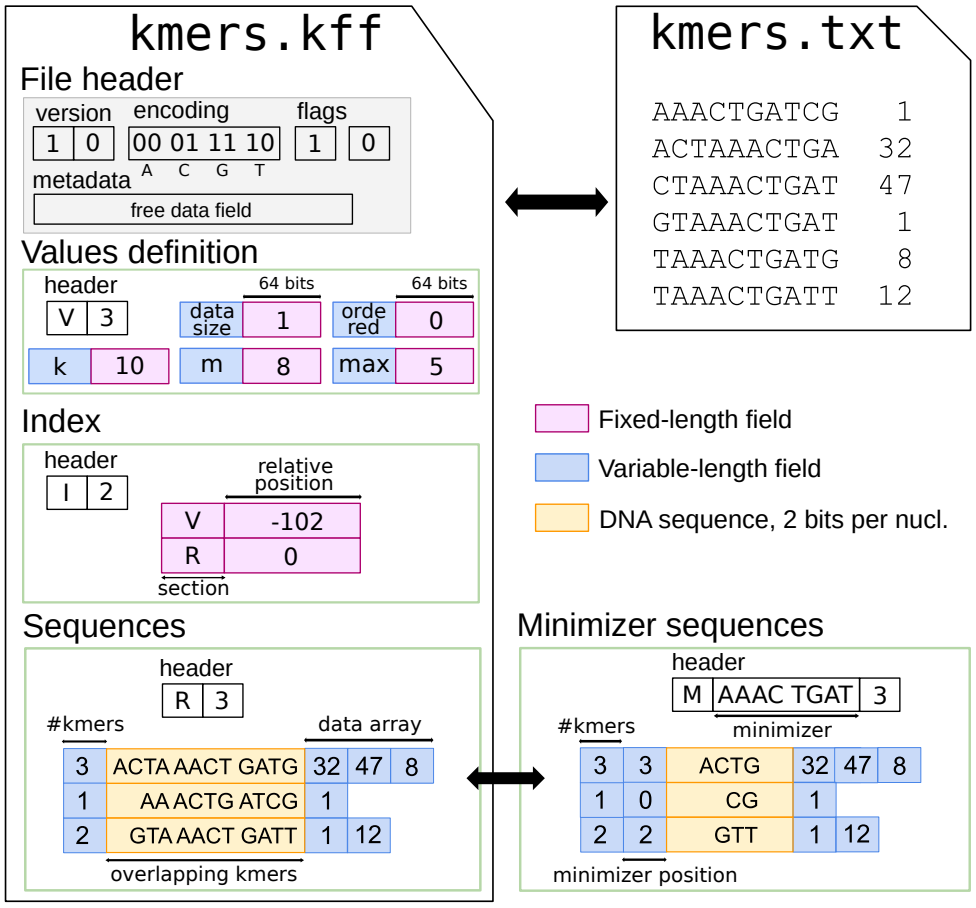

Figure VI.2 – Overview of the KFF specifications, reprinted from [6]. The header section stores the magic number to identify the format, the nucleotide encoding, and some flags related to k-mer canonicity. A variable-sized block is also allowed for eventual metadata. The value section **V** is a scoped (until a new **V** section is reached) key-value section. It defines variables required by the subsequent data section. The index section **I** stores relative addresses allowing to jump between sections. Finally, two data sections are allowed: raw (**R**) and minimizer (**M**) sections. The raw section allows to store variable-sized overlapping k-mers associated or not with an abundance vector. The minimizer section allows to store sequence of overlapping k-mers that share the same minimizer (e.g. super-k-mers). The sequence of the minimizer is then stored once per section and the storage consists in storing sequence without minimizer associated with the minimizer position.

3 decOM: Similarity-based microbial source tracking of ancient oral samples using k-mer-based methods [7]

The relatively long life span of DNA molecules allows to perform genetic analysis on very old samples. However, these molecules still suffer degradations due to time and environmental conditions [176]. In addition, the studied sample frequently contains modern or ancient contaminations unrelated to the study. These contaminations are the consequence of the immediate environment of the sample but also sometimes of the collecting itself, despite the numerous precautions. In other words, contaminations are extremely frequent and probably inevitable [177].

The characterization and quantification of the contaminations are essential to guarantee the quality of the downstream analyses. In the context of metagenomics, this procedure is called Microbial Source Tracking (MST) and consists of the quantification of different sources in the samples, i.e. the different microbial environments. Different techniques [178, 179] exist and rely on reference databases to track the sources and usually require a fine parameter tuning to obtain optimal results [180].

As a result, this project aims to provide a new MST reference-free method, called OM, and not surprisingly relies on k-mers. The idea is to represent possible contaminants by a set of real metagenomic samples (called the sources) and to use it to compute the contributions of such contaminants in an external group of samples (called the sinks). In our case, a collection of samples corresponding to Sediment/Soil, skin, ancient oral microbiome (aOral) and modern oral microbiome (mOral) was used to quantity the possible contaminations of an aOral sample. The reference datasets is composed of 360 metagenomes from different databases [181, 182, 183].

`OM` requires a binary presence/absence matrix M representing sources, where each $M(i, j)$ corresponds to the presence/absence of a k-mer i in a sample j. According to the metadata, each column has a label corresponding to the environments, i.e. aOral, mOral, Skin, Sediment/Soil. The matrix is compared to one or more sinks represented by column vectors. Using the binary matrix, `OM` counts the number of k-mers from a sink that fit in each environmental condition. For that, each sink vector which is compared to each source vector. At the end, the number of k-mers in each label is used to estimate the proportions of each source in the studied sink. An overview of the pipeline is presented in figure VI.3.

`OM` relies on `kmtricks` and benefits from several features. The minimizer partitioning allows speeding the computations by sub-sampling k-mers. Indeed, the estimated proportions of sources in sinks are close, whatever the number of partitions considered. In the aOral experiment, only one partition is considered, allowing to build a matrix with only 14 million of k-mers instead of 9 billion. In addition, the same contaminant matrix can be used for characterizing different sinks and can consequently be constructed only once. If the sinks are counted using the same partitioning scheme as the sources, the comparison is easy because identical k-mers from sources

and sinks belong to the same partition. Thanks to the sub-sampling of partitions, the comparison is also really fast because only one or a few partitions of the sinks have to be counted. The need for comparisons between a matrix (sources) and a new k-mer table (sink) resulted in the `kmtricks filter` command presented in the section V.2.B.1. I originally developed both features, sub-sampling and `kmtricks filter`, for this project.

On the aOral experiments, `OM` have shown better results than the state-of-the-art tools, `mSourceTracker` [178] and `FEAST` [179]. The different evaluations and validations show significantly higher accuracy, precision, and recall for `OM`. The computational performances are also in favor of `OM`. The paper is currently in the review process. The tool is available on github at https://github.com/CamilaDuitama/decOM.

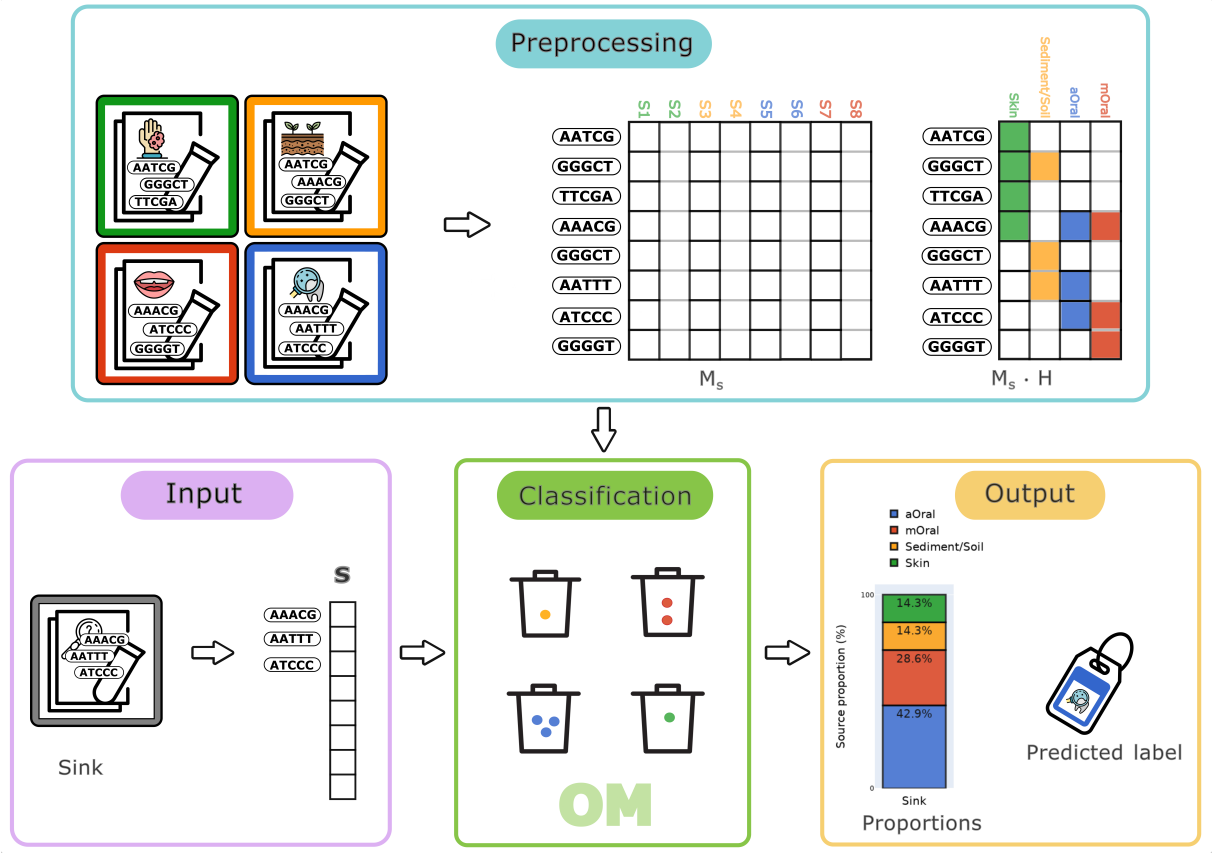

Figure VI.3 – Overview of the decOM method, reprinted from [7]. 1. Sources from different environments (aOral, mOral, Skin, Sediment/Soil) are used to produce a binary presence/absence matrix. 2. The sink to characterize is transformed into a k-mer column vector. 3. The k-mers from the sink are associated with environmental labels according to the binary matrix. 4. The proportions of sources in the sink are estimated according to k-mer labelling of the previous step.

Conclusions and perspectives

In this work, we have explored the ability of k-mer matrices to address scaling problems in various contexts. We showed that such representations can be involved at different levels of reference-free sequencing data processing while allowing new operations such as k-mer rescue.

We propose new construction and streaming methods for different types of matrices for analysis or indexing purposes, allowing the processing of datasets up to tens of terabytes in size. Such efficient construction and streaming capabilities allowed us to exhibit two major applications:

1. A fast differential k-mer analysis. Differential k-mer analysis, consisting of finding differentially represented k-mers between phenotypes, was limited by the k-mer enumeration. Such an experiment does not require the complete representation of the matrix, but merely a view on each row. Consequently, partitioned and parallel streaming of the matrix has significantly accelerated a state-of-the-art method, i.e. has enabled the processing of 80 human sequencing samples in a few hours compared to several days. In addition, our tools use fewer computing resources than previous methods, facilitating their usage. Note that k-mer processing is still the bottleneck of differential k-mer analysis.

2. An end-to-end indexing pipeline. Our method allowed to build efficiently the Tara Ocean index while considering more k-mers than other tools thanks to the k-mer rescue. In addition, our implementation is the first to benefit from `findere`, allowing more accurate queries. Consequently, it was possible to reduce the index size, resulting in time and space savings while keeping equivalent precision. However, for large database indexing, I think a paradigm shift is mandatory. The improvement of data structures and indexing methods have a noticeable impact on medium-sized datasets, at project-scale such as Tara Ocean for example. Regarding database contents, e.g. SRA, these improvements remain relatively limited. I think one possible direction would be multi-scale indexing, i.e. indexes with different and more and more precise layers to guide the queries. Recently, a new pre-filtering tool, `Raptor` [184], has emerged. Instead of associating a query with specific samples, it is associated with bins containing many samples. Based on minimizers, such queries are less computationally expensive than k-mer queries and could allow to select sample-level sub-indexes. I am strongly convinced that such approaches should be further explored. Currently, the main drawback is that `Raptor` requires prior knowledge on the data to constitute the bins.

Finally, we propose efficient and extensible implementations of our works. Our tools were already used by other projects, with or without our direct participation, sometimes leading to

new developments. We hope that they will be beneficial to the bioinformatic community.

Although k-mer matrices are generic objects allowing a variety of analyses, our work opens direct perspectives and future works, which are discussed below.

Towards k-mer-based variants detection

The differential k-mer analysis is only a preliminary step towards k-mer-based GWAS-like studies. Indeed, k-mers detected by such techniques should be characterized to bring a better biological value. The challenge consists in moving from a significant k-mer sets, potentially incomplete, to a well-characterized set of genetic variations. Several directions are possible from alignment to k-mer graphs, each with its own advantages and drawbacks. Anyway, such problem requires a fine understanding of genetic variants through the k-mer prism and constitutes an important future work.

Towards faster k-mer queries

Beyond the construction, query time is a crucial consideration of indexing methods and bounds the possible downstream analyses. The particular layout of the filters produced by our method could improve the current query times.

The BFs produced by `kmtricks` are partitioned filters, i.e. each consecutive section of the filter corresponds to a specific set of minimizers. This particular layout could lead to a new variant of the HowDeSBT index. As stated previously, HowDeSBT resolves queries by walking down in the SBT until they are resolved or simply unresolvable. In the current implementation, the whole bit-vectors representing a node are loaded from disk to memory at each step. As a result, the query time is bounded by I/O operations when indexes are large. For example, our tests indicates that performing 1 or 10,000 queries on the large Tara Ocean index takes approximately the same amount of time. In other words, reducing the I/O costs could drastically reduces the query time.

At each node, the partitioned layout could enable the loading of a few filter ranges, corresponding to the subset of minimizers involved in the query. In this case, the implementation could benefit from memory-mapped files. In addition, each partition could be an independent SBT with its own topology, allowing an efficient parallelization and probably a better compression. The current implementation of HowDeSBT does not allow to reach these targets easily. A consequent engineering work would be necessary but would result in a significant improvement at every level from construction to querying.

A public Tara Ocean index

We showed that our methods can build indexes for relatively large sequencing projects like Tara Ocean. The next step is to make the index accessible and searchable by everyone. We are currently working with the OGA team in this direction. As a reminder, OGA (https://tara-oceans.mio.osupytheas.fr/ocean-gene-atlas/) is a web service allowing to query a part of the data produced by the Tara Foundation, i.e. the assembled data. An index allowing direct queries against the sequencing samples would allow to query and exploit a larger part of the data. OGA and our index are based on different paradigms, alignment versus k-mer queries, respectively. Consequently, the deployment of the k-mer index requires the development of new interfaces and representations. The project is currently in a testing phase and limited to the Tara Ocean index. In the future, the other sequencing cohorts of the Tara foundation could also be indexed. In addition, the Tara dataset might be interesting to explore multi-scale indexing since it is composed of several sequencing campaigns with their own specificities. Likewise, groups could be constituted within sequencing cohorts, by sampling depth for example. In summary, it would consist in constituting different levels of indexing to drive the queries. Currently, no indexing methods allows such type of layered resolution, requiring further methodological developments.

BIBLIOGRAPHY

[1] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, July 1970. ISSN 0001-0782. doi: 10.1145/362686.362692. URL https://doi.org/10.1145/362686.362692. [pages 1 and 16.]

[2] Atif Rahman, Ingileif Hallgrímsdóttir, Michael Eisen, and Lior Pachter. Association mapping from sequencing reads using k-mers. *eLife*, 7:e32920, June 2018. ISSN 2050-084X. doi: 10.7554/eLife.32920. URL https://doi.org/10.7554/eLife.32920. Publisher: eLife Sciences Publications, Ltd. [pages 4, 5, 9, 36, 50, 53, 54, 56, and 58.]

[3] Zakaria Mehrab, Jaiaid Mobin, Ibrahim Asadullah Tahmid, and Atif Rahman. Efficient association mapping from k-mers—An application in finding sex-specific sequences. *PLOS ONE*, 16(1):e0245058, January 2021. ISSN 1932-6203. doi: 10.1371/journal.pone.0245058. URL https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0245058. Publisher: Public Library of Science. [pages 4, 9, 36, and 58.]

[4] Lucas Robidou and Pierre Peterlongo. findere: Fast and Precise Approximate Membership Query. In Thierry Lecroq and Hélène Touzet, editors, *String Processing and Information Retrieval*, Lecture Notes in Computer Science, pages 151–163, Cham, 2021. Springer International Publishing. ISBN 978-3-030-86692-1. doi: 10.1007/978-3-030-86692-1_13. [pages 5, 34, and 35.]

[5] Grégoire Siekaniec, Emeline Roux, Téo Lemane, Eric Guédon, and Jacques Nicolas. Identification of isolated or mixed strains from long reads: a challenge met on Streptococcus thermophilus using a MinION sequencer. *Microbial Genomics*, 7(11):000654, November 2021. ISSN 2057-5858. doi: 10.1099/mgen.0.000654. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8743539/. [pages , 103, 104, and 105.]

[6] Yoann Dufresne, Teo Lemane, Pierre Marijon, Pierre Peterlongo, Amatur Rahman, Marek Kokot, Paul Medvedev, Sebastian Deorowicz, and Rayan Chikhi. The K-mer File Format: a standardized and compact disk representation of sets of k-mers. *Bioinformatics*, page btac528, July 2022. ISSN 1367-4803. doi: 10.1093/bioinformatics/btac528. URL https://doi.org/10.1093/bioinformatics/btac528. [pages , 106, and 107.]

[7] Camila Gonzàles, Duitama, Riccardo Vicedomini, Téo Lemane, Nicolas Rascovan, Hugues Richard, and Rayan Chikhi. Microbial source tracking for contamination assessment of an-

cient oral samples using k-mer-based methods. *Submitted to Microbiome Journal*. [pages , 108, and 109.]

[8] F. Sanger, S. Nicklen, and A. R. Coulson. DNA sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences of the United States of America*, 74(12):5463–5467, December 1977. ISSN 0027-8424. doi: 10.1073/pnas.74.12.5463. [page 1.]

[9] S. Anderson, A. T. Bankier, B. G. Barrell, M. H. L. de Bruijn, A. R. Coulson, J. Drouin, I. C. Eperon, D. P. Nierlich, B. A. Roe, F. Sanger, P. H. Schreier, A. J. H. Smith, R. Staden, and I. G. Young. Sequence and organization of the human mitochondrial genome. *Nature*, 290(5806):457–465, April 1981. ISSN 1476-4687. doi: 10.1038/290457a0. URL https://www.nature.com/articles/290457a0. Number: 5806 Publisher: Nature Publishing Group. [page 1.]

[10] J. Craig Venter, Mark D. Adams, Eugene W. Myers, Peter W. Li, Richard J. Mural, Granger G. Sutton, Hamilton O. Smith, Mark Yandell, Cheryl A. Evans, Robert A. Holt, Jeannine D. Gocayne, Peter Amanatides, Richard M. Ballew, Daniel H. Huson, Jennifer Russo Wortman, Qing Zhang, Chinnappa D. Kodira, Xiangqun H. Zheng, Lin Chen, Marian Skupski, Gangadharan Subramanian, Paul D. Thomas, Jinghui Zhang, George L. Gabor Miklos, Catherine Nelson, Samuel Broder, Andrew G. Clark, Joe Nadeau, Victor A. McKusick, Norton Zinder, Arnold J. Levine, Richard J. Roberts, Mel Simon, Carolyn Slayman, Michael Hunkapiller, Randall Bolanos, Arthur Delcher, Ian Dew, Daniel Fasulo, Michael Flanigan, Liliana Florea, Aaron Halpern, Sridhar Hannenhalli, Saul Kravitz, Samuel Levy, Clark Mobarry, Knut Reinert, Karin Remington, Jane Abu-Threideh, Ellen Beasley, Kendra Biddick, Vivien Bonazzi, Rhonda Brandon, Michele Cargill, Ishwar Chandramouliswaran, Rosane Charlab, Kabir Chaturvedi, Zuoming Deng, Valentina Di Francesco, Patrick Dunn, Karen Eilbeck, Carlos Evangelista, Andrei E. Gabrielian, Weiniu Gan, Wangmao Ge, Fangcheng Gong, Zhiping Gu, Ping Guan, Thomas J. Heiman, Maureen E. Higgins, Rui-Ru Ji, Zhaoxi Ke, Karen A. Ketchum, Zhongwu Lai, Yiding Lei, Zhenya Li, Jiayin Li, Yong Liang, Xiaoying Lin, Fu Lu, Gennady V. Merkulov, Natalia Milshina, Helen M. Moore, Ashwinikumar K Naik, Vaibhav A. Narayan, Beena Neelam, Deborah Nusskern, Douglas B. Rusch, Steven Salzberg, Wei Shao, Bixiong Shue, Jingtao Sun, Zhen Yuan Wang, Aihui Wang, Xin Wang, Jian Wang, Ming-Hui Wei, Ron Wides, Chunlin Xiao, Chunhua Yan, Alison Yao, Jane Ye, Ming Zhan, Weiqing Zhang, Hongyu Zhang, Qi Zhao, Liansheng Zheng, Fei Zhong, Wenyan Zhong, Shiaoping C. Zhu, Shaying Zhao, Dennis Gilbert, Suzanna Baumhueter, Gene Spier, Christine Carter, Anibal Cravchik, Trevor Woodage, Feroze Ali, Huijin An, Aderonke Awe, Danita Baldwin, Holly Baden, Mary Barnstead, Ian Barrow, Karen Beeson, Dana Busam, Amy Carver, Angela Center, Ming Lai Cheng, Liz Curry, Steve Danaher, Lionel Davenport, Raymond Desilets,

Susanne Dietz, Kristina Dodson, Lisa Doup, Steven Ferriera, Neha Garg, Andres Gluecksmann, Brit Hart, Jason Haynes, Charles Haynes, Cheryl Heiner, Suzanne Hladun, Damon Hostin, Jarrett Houck, Timothy Howland, Chinyere Ibegwam, Jeffery Johnson, Francis Kalush, Lesley Kline, Shashi Koduru, Amy Love, Felecia Mann, David May, Steven McCawley, Tina McIntosh, Ivy McMullen, Mee Moy, Linda Moy, Brian Murphy, Keith Nelson, Cynthia Pfannkoch, Eric Pratts, Vinita Puri, Hina Qureshi, Matthew Reardon, Robert Rodriguez, Yu-Hui Rogers, Deanna Romblad, Bob Ruhfel, Richard Scott, Cynthia Sitter, Michelle Smallwood, Erin Stewart, Renee Strong, Ellen Suh, Reginald Thomas, Ni Ni Tint, Sukyee Tse, Claire Vech, Gary Wang, Jeremy Wetter, Sherita Williams, Monica Williams, Sandra Windsor, Emily Winn-Deen, Keriellen Wolfe, Jayshree Zaveri, Karena Zaveri, Josep F. Abril, Roderic Guigó, Michael J. Campbell, Kimmen V. Sjolander, Brian Karlak, Anish Kejariwal, Huaiyu Mi, Betty Lazareva, Thomas Hatton, Apurva Narechania, Karen Diemer, Anushya Muruganujan, Nan Guo, Shinji Sato, Vineet Bafna, Sorin Istrail, Ross Lippert, Russell Schwartz, Brian Walenz, Shibu Yooseph, David Allen, Anand Basu, James Baxendale, Louis Blick, Marcelo Caminha, John Carnes-Stine, Parris Caulk, Yen-Hui Chiang, My Coyne, Carl Dahlke, Anne Deslattes Mays, Maria Dombroski, Michael Donnelly, Dale Ely, Shiva Esparham, Carl Fosler, Harold Gire, Stephen Glanowski, Kenneth Glasser, Anna Glodek, Mark Gorokhov, Ken Graham, Barry Gropman, Michael Harris, Jeremy Heil, Scott Henderson, Jeffrey Hoover, Donald Jennings, Catherine Jordan, James Jordan, John Kasha, Leonid Kagan, Cheryl Kraft, Alexander Levitsky, Mark Lewis, Xiangjun Liu, John Lopez, Daniel Ma, William Majoros, Joe McDaniel, Sean Murphy, Matthew Newman, Trung Nguyen, Ngoc Nguyen, Marc Nodell, Sue Pan, Jim Peck, Marshall Peterson, William Rowe, Robert Sanders, John Scott, Michael Simpson, Thomas Smith, Arlan Sprague, Timothy Stockwell, Russell Turner, Eli Venter, Mei Wang, Meiyuan Wen, David Wu, Mitchell Wu, Ashley Xia, Ali Zandieh, and Xiaohong Zhu. The Sequence of the Human Genome. *Science*, 291(5507):1304–1351, February 2001. doi: 10.1126/science.1058040. URL https://www.science.org/doi/abs/10.1126/science.1058040. Publisher: American Association for the Advancement of Science. [page 1.]

[11] E. S. Lander, L. M. Linton, B. Birren, C. Nusbaum, M. C. Zody, J. Baldwin, K. Devon, K. Dewar, M. Doyle, W. FitzHugh, R. Funke, D. Gage, K. Harris, A. Heaford, J. Howland, L. Kann, J. Lehoczky, R. LeVine, P. McEwan, K. McKernan, J. Meldrim, J. P. Mesirov, C. Miranda, W. Morris, J. Naylor, C. Raymond, M. Rosetti, R. Santos, A. Sheridan, C. Sougnez, Y. Stange-Thomann, N. Stojanovic, A. Subramanian, D. Wyman, J. Rogers, J. Sulston, R. Ainscough, S. Beck, D. Bentley, J. Burton, C. Clee, N. Carter, A. Coulson, R. Deadman, P. Deloukas, A. Dunham, I. Dunham, R. Durbin, L. French, D. Grafham, S. Gregory, T. Hubbard, S. Humphray, A. Hunt, M. Jones, C. Lloyd, A. McMurray, L. Matthews, S. Mercer, S. Milne, J. C. Mullikin, A. Mungall, R. Plumb, M. Ross,

R. Shownkeen, S. Sims, R. H. Waterston, R. K. Wilson, L. W. Hillier, J. D. McPherson, M. A. Marra, E. R. Mardis, L. A. Fulton, A. T. Chinwalla, K. H. Pepin, W. R. Gish, S. L. Chissoe, M. C. Wendl, K. D. Delehaunty, T. L. Miner, A. Delehaunty, J. B. Kramer, L. L. Cook, R. S. Fulton, D. L. Johnson, P. J. Minx, S. W. Clifton, T. Hawkins, E. Branscomb, P. Predki, P. Richardson, S. Wenning, T. Slezak, N. Doggett, J. F. Cheng, A. Olsen, S. Lucas, C. Elkin, E. Uberbacher, M. Frazier, R. A. Gibbs, D. M. Muzny, S. E. Scherer, J. B. Bouck, E. J. Sodergren, K. C. Worley, C. M. Rives, J. H. Gorrell, M. L. Metzker, S. L. Naylor, R. S. Kucherlapati, D. L. Nelson, G. M. Weinstock, Y. Sakaki, A. Fujiyama, M. Hattori, T. Yada, A. Toyoda, T. Itoh, C. Kawagoe, H. Watanabe, Y. Totoki, T. Taylor, J. Weissenbach, R. Heilig, W. Saurin, F. Artiguenave, P. Brottier, T. Bruls, E. Pelletier, C. Robert, P. Wincker, D. R. Smith, L. Doucette-Stamm, M. Rubenfield, K. Weinstock, H. M. Lee, J. Dubois, A. Rosenthal, M. Platzer, G. Nyakatura, S. Taudien, A. Rump, H. Yang, J. Yu, J. Wang, G. Huang, J. Gu, L. Hood, L. Rowen, A. Madan, S. Qin, R. W. Davis, N. A. Federspiel, A. P. Abola, M. J. Proctor, R. M. Myers, J. Schmutz, M. Dickson, J. Grimwood, D. R. Cox, M. V. Olson, R. Kaul, C. Raymond, N. Shimizu, K. Kawasaki, S. Minoshima, G. A. Evans, M. Athanasiou, R. Schultz, B. A. Roe, F. Chen, H. Pan, J. Ramser, H. Lehrach, R. Reinhardt, W. R. McCombie, M. de la Bastide, N. Dedhia, H. Blöcker, K. Hornischer, G. Nordsiek, R. Agarwala, L. Aravind, J. A. Bailey, A. Bateman, S. Batzoglou, E. Birney, P. Bork, D. G. Brown, C. B. Burge, L. Cerutti, H. C. Chen, D. Church, M. Clamp, R. R. Copley, T. Doerks, S. R. Eddy, E. E. Eichler, T. S. Furey, J. Galagan, J. G. Gilbert, C. Harmon, Y. Hayashizaki, D. Haussler, H. Hermjakob, K. Hokamp, W. Jang, L. S. Johnson, T. A. Jones, S. Kasif, A. Kaspryzk, S. Kennedy, W. J. Kent, P. Kitts, E. V. Koonin, I. Korf, D. Kulp, D. Lancet, T. M. Lowe, A. McLysaght, T. Mikkelsen, J. V. Moran, N. Mulder, V. J. Pollara, C. P. Ponting, G. Schuler, J. Schultz, G. Slater, A. F. Smit, E. Stupka, J. Szustakowki, D. Thierry-Mieg, J. Thierry-Mieg, L. Wagner, J. Wallis, R. Wheeler, A. Williams, Y. I. Wolf, K. H. Wolfe, S. P. Yang, R. F. Yeh, F. Collins, M. S. Guyer, J. Peterson, A. Felsenfeld, K. A. Wetterstrand, A. Patrinos, M. J. Morgan, P. de Jong, J. J. Catanese, K. Osoegawa, H. Shizuya, S. Choi, Y. J. Chen, J. Szustakowki, and International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, Feb 2001. ISSN 0028-0836. doi: 10.1038/35057062. [page 1.]

[12] Jay Shendure and Hanlee Ji. Next-generation DNA sequencing. *Nature Biotechnology*, 26(10):1135–1145, October 2008. ISSN 1546-1696. doi: 10.1038/nbt1486. URL https://www.nature.com/articles/nbt1486. Number: 10 Publisher: Nature Publishing Group. [page 1.]

[13] Darius Wen-Shuo Koh, Kwok-Fong Chan, Weiling Wu, and Samuel Ken-En Gan. Yet

Another Quick Assembly, Analysis and Trimming Tool (YAQAAT): A Server for the Automated Assembly and Analysis of Sanger Sequencing Data. *Journal of Biomolecular Techniques : JBT*, pages jbt.2021–3202–003, July 2021. ISSN 1524-0215. doi: 10.7171/jbt.2021-3202-003. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7861051/. [page 1.]

[14] Yuriy O. Alekseyev, Roghayeh Fazeli, Shi Yang, Raveen Basran, Thomas Maher, Nancy S. Miller, and Daniel Remick. A Next-Generation Sequencing Primer—How Does It Work and What Can It Do? *Academic Pathology*, 5:2374289518766521, May 2018. ISSN 2374-2895. doi: 10.1177/2374289518766521. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5944141/. [page 2.]

[15] Rupesh Kanchi Ravi, Kendra Walton, and Mahdieh Khosroheidari. MiSeq: A Next Generation Sequencing Platform for Genomic Analysis. In Johanna K. DiStefano, editor, *Disease Gene Identification: Methods and Protocols*, Methods in Molecular Biology, pages 223–232. Springer, New York, NY, 2018. ISBN 978-1-4939-7471-9. doi: 10.1007/978-1-4939-7471-9_12. URL https://doi.org/10.1007/978-1-4939-7471-9_12. [page 2.]

[16] Barry Merriman, Ion Torrent R&D Team, and Jonathan M. Rothberg. Progress in Ion Torrent semiconductor chip based sequencing. *ELECTROPHORESIS*, 33(23): 3397–3417, 2012. ISSN 1522-2683. doi: 10.1002/elps.201200424. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/elps.201200424. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/elps.201200424. [page 2.]

[17] Michael A. Quail, Miriam Smith, Paul Coupland, Thomas D. Otto, Simon R. Harris, Thomas R. Connor, Anna Bertoni, Harold P. Swerdlow, and Yong Gu. A tale of three next generation sequencing platforms: comparison of Ion Torrent, Pacific Biosciences and Illumina MiSeq sequencers. *BMC Genomics*, 13(1):341, July 2012. ISSN 1471-2164. doi: 10.1186/1471-2164-13-341. URL https://doi.org/10.1186/1471-2164-13-341. [page 2.]

[18] Takeru Nakazato, Tazro Ohta, and Hidemasa Bono. Experimental Design-Based Functional Mining and Characterization of High-Throughput Sequencing Data in the Sequence Read Archive. *PLOS ONE*, 8(10):e77910, October 2013. ISSN 1932-6203. doi: 10.1371/journal.pone.0077910. URL https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0077910. Publisher: Public Library of Science. [page 2.]

[19] Neil I. Weisenfeld, Vijay Kumar, Preyas Shah, Deanna M. Church, and David B. Jaffe. Direct determination of diploid genome sequences. *Genome Research*, 27(5):757–767, May 2017. ISSN 1088-9051, 1549-5469. doi: 10.1101/gr.214874.116. URL https://genome.csh

lp.org/content/27/5/757. Company: Cold Spring Harbor Laboratory Press Distributor: Cold Spring Harbor Laboratory Press Institution: Cold Spring Harbor Laboratory Press Label: Cold Spring Harbor Laboratory Press Publisher: Cold Spring Harbor Lab. [page 2.]

[20] Zhanshan (Sam) Ma, Lianwei Li, Chengxi Ye, Minsheng Peng, and Ya-Ping Zhang. Hybrid assembly of ultra-long Nanopore reads augmented with 10x-Genomics contigs: Demonstrated with a human genome. *Genomics*, 111(6):1896–1901, December 2019. ISSN 0888-7543. doi: 10.1016/j.ygeno.2018.12.013. URL https://www.sciencedirect.com/science/article/pii/S0888754318305603. [page 2.]

[21] Pierre Morisse, Fabrice Legeai, and Claire Lemaitre. LEVIATHAN: efficient discovery of large structural variants by leveraging long-range information from Linked-Reads data, March 2021. URL https://www.biorxiv.org/content/10.1101/2021.03.25.437002v1. Pages: 2021.03.25.437002 Section: New Results. [page 2.]

[22] Martin O. Pollard, Deepti Gurdasani, Alexander J. Mentzer, Tarryn Porter, and Manjinder S. Sandhu. Long reads: their purpose and place. *Human Molecular Genetics*, 27(R2): R234–R241, August 2018. ISSN 1460-2083. doi: 10.1093/hmg/ddy177. [page 2.]

[23] Rachael E. Workman, Alison D. Tang, Paul S. Tang, Miten Jain, John R. Tyson, Roham Razaghi, Philip C. Zuzarte, Timothy Gilpatrick, Alexander Payne, Joshua Quick, Norah Sadowski, Nadine Holmes, Jaqueline Goes de Jesus, Karen L. Jones, Cameron M. Soulette, Terrance P. Snutch, Nicholas Loman, Benedict Paten, Matthew Loose, Jared T. Simpson, Hugh E. Olsen, Angela N. Brooks, Mark Akeson, and Winston Timp. Nanopore native RNA sequencing of a human poly(A) transcriptome. *Nature Methods*, 16(12):1297–1305, December 2019. ISSN 1548-7091, 1548-7105. doi: 10.1038/s41592-019-0617-2. URL https://www.nature.com/articles/s41592-019-0617-2. [page 2.]

[24] Clara Delahaye and Jacques Nicolas. Sequencing DNA with nanopores: Troubles and biases. *PLOS ONE*, 16(10):e0257521, October 2021. ISSN 1932-6203. doi: 10.1371/journal.pone.0257521. URL https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0257521. Publisher: Public Library of Science. [page 2.]

[25] Levene Mj, Korlach J, Turner Sw, Foquet M, Craighead Hg, and Webb Ww. Zero-mode waveguides for single-molecule analysis at high concentrations. *Science (New York, N.Y.)*, 299(5607), January 2003. ISSN 1095-9203. doi: 10.1126/science.1079700. URL https://pubmed.ncbi.nlm.nih.gov/12560545/. Publisher: Science. [page 3.]

[26] Anthony Rhoads and Kin Fai Au. PacBio Sequencing and Its Applications. *Genomics, Proteomics & Bioinformatics*, 13(5):278–289, October 2015. ISSN 1672-0229. doi: 10.1016/

j.gpb.2015.08.002. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4678779/. [page 3.]

[27] Ting Hon, Kristin Mars, Greg Young, Yu-Chih Tsai, Joseph W. Karalius, Jane M. Landolin, Nicholas Maurer, David Kudrna, Michael A. Hardigan, Cynthia C. Steiner, Steven J. Knapp, Doreen Ware, Beth Shapiro, Paul Peluso, and David R. Rank. Highly accurate long-read HiFi sequencing data for five complex genomes. *Scientific Data*, 7(1): 399, November 2020. ISSN 2052-4463. doi: 10.1038/s41597-020-00743-4. URL https://www.nature.com/articles/s41597-020-00743-4. Number: 1 Publisher: Nature Publishing Group. [page 3.]

[28] Dmitry Antipov, Anton Korobeynikov, Jeffrey S. McLean, and Pavel A. Pevzner. hybridSPAdes: an algorithm for hybrid assembly of short and long reads. *Bioinformatics*, 32 (7):1009–1015, April 2016. ISSN 1367-4803. doi: 10.1093/bioinformatics/btv688. URL https://doi.org/10.1093/bioinformatics/btv688. [page 3.]

[29] Leena Salmela and Eric Rivals. LoRDEC: accurate and efficient long read error correction. *Bioinformatics*, 30(24):3506–3514, December 2014. ISSN 1367-4803. doi: 10.1093/bioinformatics/btu538. URL https://doi.org/10.1093/bioinformatics/btu538. [page 3.]

[30] Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17):9748–9753, August 2001. doi: 10.1073/pnas.171285098. URL https://www.pnas.org/doi/10.1073/pnas.171285098. Publisher: Proceedings of the National Academy of Sciences. [page 3.]

[31] Jacob F. Degner, John C. Marioni, Athma A. Pai, Joseph K. Pickrell, Everlyne Nkadori, Yoav Gilad, and Jonathan K. Pritchard. Effect of read-mapping biases on detecting allele-specific expression from RNA-sequencing data. *Bioinformatics*, 25(24):3207–3212, December 2009. ISSN 1367-4803. doi: 10.1093/bioinformatics/btp579. URL https://doi.org/10.1093/bioinformatics/btp579. [page 4.]

[32] Débora Y C Brandt, Vitor R C Aguiar, Bárbara D Bitarello, Kelly Nunes, Jérôme Goudet, and Diogo Meyer. Mapping Bias Overestimates Reference Allele Frequencies at the HLA Genes in the 1000 Genomes Project Phase I Data. *G3 Genes|Genomes|Genetics*, 5(5): 931–941, May 2015. ISSN 2160-1836. doi: 10.1534/g3.114.015784. URL https://doi.org/10.1534/g3.114.015784. [page 4.]

[33] Mazdak Salavati, Stephen J. Bush, Sergio Palma-Vera, Mary E. B. McCulloch, David A. Hume, and Emily L. Clark. Elimination of Reference Mapping Bias Reveals Robust Immune Related Allele-Specific Expression in Crossbred Sheep. *Frontiers in Genetics*, 10,

2019. ISSN 1664-8021. URL https://www.frontiersin.org/articles/10.3389/fgene.2019.00863. [page 4.]

[34] Deepti Gurdasani, Tommy Carstensen, Fasil Tekola-Ayele, Luca Pagani, Ioanna Tachmazidou, Konstantinos Hatzikotoulas, Savita Karthikeyan, Louise Iles, Martin O. Pollard, Ananyo Choudhury, Graham R. S. Ritchie, Yali Xue, Jennifer Asimit, Rebecca N. Nsubuga, Elizabeth H. Young, Cristina Pomilla, Katja Kivinen, Kirk Rockett, Anatoli Kamali, Ayo P. Doumatey, Gershim Asiki, Janet Seeley, Fatoumatta Sisay-Joof, Muminatou Jallow, Stephen Tollman, Ephrem Mekonnen, Rosemary Ekong, Tamiru Oljira, Neil Bradman, Kalifa Bojang, Michele Ramsay, Adebowale Adeyemo, Endashaw Bekele, Ayesha Motala, Shane A. Norris, Fraser Pirie, Pontiano Kaleebu, Dominic Kwiatkowski, Chris Tyler-Smith, Charles Rotimi, Eleftheria Zeggini, and Manjinder S. Sandhu. The African Genome Variation Project shapes medical genetics in Africa. *Nature*, 517(7534): 327–332, January 2015. ISSN 1476-4687. doi: 10.1038/nature13997. URL https://www.nature.com/articles/nature13997. Number: 7534 Publisher: Nature Publishing Group. [page 4.]

[35] Erik Garrison, Jouni Sirén, Adam M. Novak, Glenn Hickey, Jordan M. Eizenga, Eric T. Dawson, William Jones, Shilpa Garg, Charles Markello, Michael F. Lin, Benedict Paten, and Richard Durbin. Variation graph toolkit improves read mapping by representing genetic variation in the reference. *Nature Biotechnology*, 36(9):875–879, October 2018. ISSN 1546-1696. doi: 10.1038/nbt.4227. URL https://www.nature.com/articles/nbt.4227. Number: 9 Publisher: Nature Publishing Group. [page 4.]

[36] Nae-Chyun Chen, Brad Solomon, Taher Mun, Sheila Iyer, and Ben Langmead. Reference flow: reducing reference bias using multiple population genomes. *Genome Biology*, 22(1): 8, January 2021. ISSN 1474-760X. doi: 10.1186/s13059-020-02229-3. URL https://doi.org/10.1186/s13059-020-02229-3. [page 4.]

[37] Ben Langmead and Steven L. Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature Methods*, 9(4):357–359, April 2012. ISSN 1548-7105. doi: 10.1038/nmeth.1923. URL https://www.nature.com/articles/nmeth.1923. Number: 4 Publisher: Nature Publishing Group. [page 4.]

[38] Heng Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM, May 2013. URL http://arxiv.org/abs/1303.3997. arXiv:1303.3997 [q-bio]. [page 4.]

[39] Heng Li and Richard Durbin. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics (Oxford, England)*, 25(14):1754–1760, July 2009. ISSN 1367-4811. doi: 10.1093/bioinformatics/btp324. [page 4.]

[40] Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18): 3094–3100, September 2018. ISSN 1367-4803. doi: 10.1093/bioinformatics/bty191. URL https://doi.org/10.1093/bioinformatics/bty191. [page 4.]

[41] Jang-il Sohn and Jin-Wu Nam. The present and future of de novo whole-genome assembly. *Briefings in Bioinformatics*, 19(1):23–40, January 2018. ISSN 1477-4054. doi: 10.1093/bib/bbw096. URL https://doi.org/10.1093/bib/bbw096. [page 5.]

[42] Raluca Uricaru, Guillaume Rizk, Vincent Lacroix, Elsa Quillery, Olivier Plantard, Rayan Chikhi, Claire Lemaitre, and Pierre Peterlongo. Reference-free detection of isolated SNPs. *Nucleic Acids Research*, 43(2):e11, January 2015. ISSN 0305-1048. doi: 10.1093/nar/gku1187. URL https://doi.org/10.1093/nar/gku1187. [page 5.]

[43] Mark J. Chaisson, Dumitru Brinza, and Pavel A. Pevzner. De novo fragment assembly with short mate-paired reads: Does the read length matter? *Genome Research*, 19(2): 336–346, February 2009. ISSN 1088-9051. doi: 10.1101/gr.079053.108. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2652199/. [page 5.]

[44] Jie Ren, Nathan A. Ahlgren, Yang Young Lu, Jed A. Fuhrman, and Fengzhu Sun. VirFinder: a novel k-mer based tool for identifying viral sequences from assembled metagenomic data. *Microbiome*, 5(1):69, July 2017. ISSN 2049-2618. doi: 10.1186/s40168-017-0283-5. URL https://doi.org/10.1186/s40168-017-0283-5. [page 5.]

[45] Rob Patro, Stephen M. Mount, and Carl Kingsford. Sailfish enables alignment-free isoform quantification from RNA-seq reads using lightweight algorithms. *Nature Biotechnology*, 32 (5):462–464, May 2014. ISSN 1546-1696. doi: 10.1038/nbt.2862. URL https://www.nature.com/articles/nbt.2862. Number: 5 Publisher: Nature Publishing Group. [page 5.]

[46] Yoav Voichek and Detlef Weigel. Identifying genetic variants underlying phenotypic variation in plants without complete genomes. *Nature Genetics*, 52(5):534–540, May 2020. ISSN 1546-1718. doi: 10.1038/s41588-020-0612-7. URL https://www.nature.com/articles/s41588-020-0612-7. Number: 5 Publisher: Nature Publishing Group. [pages 5, 9, 36, 50, and 56.]

[47] Rasko Leinonen, Hideaki Sugawara, and Martin Shumway. The Sequence Read Archive. *Nucleic Acids Research*, 39(Database issue):D19–D21, January 2011. ISSN 0305-1048. doi: 10.1093/nar/gkq1019. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3013647/. [page 5.]

[48] Gaëtan Benoit, Claire Lemaitre, Dominique Lavenier, Erwan Drezen, Thibault Dayris, Raluca Uricaru, and Guillaume Rizk. Reference-free compression of high throughput

sequencing data with a probabilistic de Bruijn graph. *BMC Bioinformatics*, 16(1): 288, September 2015. ISSN 1471-2105. doi: 10.1186/s12859-015-0709-7. URL https://doi.org/10.1186/s12859-015-0709-7. [page 6.]

[49] Łukasz Roguski, Idoia Ochoa, Mikel Hernaez, and Sebastian Deorowicz. FaStore: a space-saving solution for raw sequencing data. *Bioinformatics*, 34(16):2748–2756, August 2018. ISSN 1367-4803. doi: 10.1093/bioinformatics/bty205. URL https://doi.org/10.1093/bioinformatics/bty205. [page 6.]

[50] Marek Kokot, Adam Gudyś, Heng Li, and Sebastian Deorowicz. CoLoRd: compressing long reads. *Nature Methods*, 19(4):441–444, April 2022. ISSN 1548-7105. doi: 10.1038/s41592-022-01432-3. URL https://www.nature.com/articles/s41592-022-01432-3. Number: 4 Publisher: Nature Publishing Group. [page 6.]

[51] Ibrahim Numanagić, James K. Bonfield, Faraz Hach, Jan Voges, Jörn Ostermann, Claudio Alberti, Marco Mattavelli, and S. Cenk Sahinalp. Comparison of high-throughput sequencing data compression tools. *Nature Methods*, 13(12):1005–1008, December 2016. ISSN 1548-7105. doi: 10.1038/nmeth.4037. URL https://www.nature.com/articles/nmeth.4037. Number: 12 Publisher: Nature Publishing Group. [page 6.]

[52] Adam Klie, Brian Y Tsui, Shamim Mollah, Dylan Skola, Michelle Dow, Chun-Nan Hsu, and Hannah Carter. Increasing metadata coverage of SRA BioSample entries using deep learning–based named entity recognition. *Database*, 2021:baab021, September 2021. ISSN 1758-0463. doi: 10.1093/database/baab021. URL https://doi.org/10.1093/database/baab021. [page 7.]

[53] Robert C. Edgar, Jeff Taylor, Victor Lin, Tomer Altman, Pierre Barbera, Dmitry Meleshko, Dan Lohr, Gherman Novakovsky, Benjamin Buchfink, Basem Al-Shayeb, Jillian F. Banfield, Marcos de la Peña, Anton Korobeynikov, Rayan Chikhi, and Artem Babaian. Petabase-scale sequence alignment catalyses viral discovery. *Nature*, 602(7895): 142–147, February 2022. ISSN 1476-4687. doi: 10.1038/s41586-021-04332-2. URL https://www.nature.com/articles/s41586-021-04332-2. Number: 7895 Publisher: Nature Publishing Group. [page 8.]

[54] The 1000 Genomes Project Consortium. A global reference for human genetic variation. *Nature 2015 526:7571*, 526(7571):68–74, September 2015. ISSN 1476-4687. doi: 10.1038/nature15393. URL https://www.nature.com/articles/nature15393. Publisher: Nature Publishing Group. [pages 8, 35, and 58.]

[55] Lita M. Proctor, Heather H. Creasy, Jennifer M. Fettweis, Jason Lloyd-Price, Anup Mahurkar, Wenyu Zhou, Gregory A. Buck, Michael P. Snyder, Jerome F. Strauss,

George M. Weinstock, Owen White, Curtis Huttenhower, and The Integrative HMP (iHMP) Research Network Consortium. The Integrative Human Microbiome Project. *Nature*, 569(7758):641–648, May 2019. ISSN 1476-4687. doi: 10.1038/s41586-019-1238-8. URL https://www.nature.com/articles/s41586-019-1238-8. Number: 7758 Publisher: Nature Publishing Group. [pages 8 and 35.]

[56] Brad Solomon and Carl Kingsford. Fast search of thousands of short-read sequencing experiments. *Nature Biotechnology*, 34(3):300–302, March 2016. ISSN 1546-1696. doi: 10.1038/nbt.3442. URL https://www.nature.com/articles/nbt.3442. Number: 3 Publisher: Nature Publishing Group. [pages 8, 15, 21, 27, 28, 29, and 32.]

[57] Christiam Camacho, George Coulouris, Vahram Avagyan, Ning Ma, Jason Papadopoulos, Kevin Bealer, and Thomas L. Madden. BLAST+: architecture and applications. *BMC Bioinformatics*, 10(1):421, December 2009. ISSN 1471-2105. doi: 10.1186/1471-2105-10-421. URL https://doi.org/10.1186/1471-2105-10-421. [page 8.]

[58] Alexander Dobin, Carrie A. Davis, Felix Schlesinger, Jorg Drenkow, Chris Zaleski, Sonali Jha, Philippe Batut, Mark Chaisson, and Thomas R. Gingeras. STAR: ultrafast universal RNA-seq aligner. *Bioinformatics (Oxford, England)*, 29(1):15–21, January 2013. ISSN 1367-4811. doi: 10.1093/bioinformatics/bts635. [page 8.]

[59] Nina Luhmann, Guillaume Holley, and Mark Achtman. BlastFrost: fast querying of 100,000s of bacterial genomes in Bifrost graphs. *Genome Biology*, 22(1):30, January 2021. ISSN 1474-760X. doi: 10.1186/s13059-020-02237-3. URL https://doi.org/10.1186/s13059-020-02237-3. [pages 8 and 15.]

[60] Guillaume Holley and Páll Melsted. Bifrost: highly parallel construction and indexing of colored and compacted de Bruijn graphs. *Genome Biology*, 21(1):249, September 2020. ISSN 1474-760X. doi: 10.1186/s13059-020-02135-8. URL https://doi.org/10.1186/s13059-020-02135-8. [page 8.]

[61] Jian Ye, Scott McGinnis, and Thomas L. Madden. BLAST: improvements for better sequence analysis. *Nucleic Acids Research*, 34(suppl_2):W6–W9, July 2006. ISSN 0305-1048. doi: 10.1093/nar/gkl164. URL https://doi.org/10.1093/nar/gkl164. [pages 8 and 72.]

[62] Roland Wittler. Alignment- and reference-free phylogenomics with colored de Bruijn graphs. *Algorithms for Molecular Biology*, 15(1):4, April 2020. ISSN 1748-7188. doi: 10.1186/s13015-020-00164-3. URL https://doi.org/10.1186/s13015-020-00164-3. [page 9.]

[63] Gaëtan Benoit, Pierre Peterlongo, Mahendra Mariadassou, Erwan Drezen, Sophie Schbath, Dominique Lavenier, and Claire Lemaitre. Multiple comparative metagenomics using multiset k-mer counting. *PeerJ Computer Science*, 2:e94, November 2016. ISSN 2376-5992. doi: 10.7717/peerj-cs.94. URL https://peerj.com/articles/cs-94. Publisher: PeerJ Inc. [pages 9, 47, 49, and 91.]

[64] Gaëtan Benoit, Mahendra Mariadassou, Stéphane Robin, Sophie Schbath, Pierre Peterlongo, and Claire Lemaitre. SimkaMin: fast and resource frugal de novo comparative metagenomics. *Bioinformatics*, 36(4):1275–1276, February 2020. ISSN 1367-4803. doi: 10.1093/bioinformatics/btz685. URL https://doi.org/10.1093/bioinformatics/btz685. [pages 9, 49, and 91.]

[65] International Union of Pure and Applied Chemistry. IUPAC Compendium of Chemical Terminology – The Gold Book, 2009. URL http://goldbook.iupac.org/. [page 12.]

[66] Karel Břinda, Michael Baym, and Gregory Kucherov. Simplitigs as an efficient and scalable representation of de Bruijn graphs. *Genome Biology*, 22(1):96, April 2021. ISSN 1474-760X. doi: 10.1186/s13059-021-02297-z. URL https://doi.org/10.1186/s13059-021-02297-z. [pages 13 and 106.]

[67] Amatur Rahman and Paul Medvedev. Representation of $$k$$-mer Sets Using Spectrum-Preserving String Sets. In Russell Schwartz, editor, *Research in Computational Molecular Biology*, Lecture Notes in Computer Science, pages 152–168, Cham, 2020. Springer International Publishing. ISBN 978-3-030-45257-5. doi: 10.1007/978-3-030-45257-5_10. [pages 13 and 106.]

[68] Sebastian Schmidt, Shahbaz Khan, Jarno Alanko, and Alexandru I. Tomescu. Matchtigs: minimum plain text representation of kmer sets, February 2022. URL https://www.biorxiv.org/content/10.1101/2021.12.15.472871v2. Pages: 2021.12.15.472871 Section: New Results. [page 13.]

[69] Rayan Chikhi, Antoine Limasset, Shaun Jackman, Jared T. Simpson, and Paul Medvedev. On the Representation of de Bruijn Graphs. In Roded Sharan, editor, *Research in Computational Molecular Biology*, Lecture Notes in Computer Science, pages 35–55, Cham, 2014. Springer International Publishing. ISBN 978-3-319-05269-4. doi: 10.1007/978-3-319-05269-4_4. [page 13.]

[70] Michael Burrows and David Wheeler. A Block-Sorting Lossless Data Compression Algorithm. 1994. URL https://citeseerx.ist.psu.edu/viewdoc/citations;jsessionid=4979FEEE5BD966B330E81C59DE112C95?doi=10.1.1.37.6774. [page 14.]

[71] P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 390–398, November 2000. doi: 10.1109/SFCS.2000.892127. ISSN: 0272-5428. [page 14.]

[72] Heng Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM, May 2013. URL http://arxiv.org/abs/1303.3997. arXiv:1303.3997 [q-bio]. [page 14.]

[73] Brian D. Ondov, Todd J. Treangen, Páll Melsted, Adam B. Mallonee, Nicholas H. Bergman, Sergey Koren, and Adam M. Phillippy. Mash: fast genome and metagenome distance estimation using MinHash. *Genome Biology*, 17(1):132, June 2016. ISSN 1474-760X. doi: 10.1186/s13059-016-0997-x. URL https://doi.org/10.1186/s13059-016-0997-x. [page 14.]

[74] Daniel N. Baker and Ben Langmead. Dashing: fast and accurate genomic distances with HyperLogLog. *Genome Biology*, 20(1):265, December 2019. ISSN 1474-760X. doi: 10.1186/s13059-019-1875-0. URL https://doi.org/10.1186/s13059-019-1875-0. [page 14.]

[75] A.Z. Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)*, pages 21–29, June 1997. doi: 10.1109/SEQUEN.1997.666900. [page 14.]

[76] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, USA, 2nd edition, 2014. ISBN 978-1-107-07723-2. [page 14.]

[77] Paul Jaccard. The Distribution of the Flora in the Alpine Zone.1. *New Phytologist*, 11(2):37–50, 1912. ISSN 1469-8137. doi: 10.1111/j.1469-8137.1912.tb05611.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1469-8137.1912.tb05611.x. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1469-8137.1912.tb05611.x. [pages 14 and 91.]

[78] Fatemeh Almodaresi, Hirak Sarkar, Avi Srivastava, and Rob Patro. A space and time-efficient index for the compacted colored de Bruijn graph. *Bioinformatics*, 34(13):i169–i177, July 2018. ISSN 1367-4803. doi: 10.1093/bioinformatics/bty292. URL https://doi.org/10.1093/bioinformatics/bty292. [page 15.]

[79] Camille Marchet, Mael Kerbiriou, and Antoine Limasset. BLight: efficient exact associative structure for k-mers. *Bioinformatics*, 37(18):2858–2865, September 2021. ISSN 1367-4803. doi: 10.1093/bioinformatics/btab217. URL https://doi.org/10.1093/bioinformatics/btab217. [page 15.]

[80] Guillaume Holley and Páll Melsted. Bifrost: highly parallel construction and indexing of colored and compacted de Bruijn graphs. *Genome Biology*, 21(1):249, September 2020.

ISSN 1474-760X. doi: 10.1186/s13059-020-02135-8. URL https://doi.org/10.1186/s13059-020-02135-8. [page 15.]

[81] Jarno N. Alanko, Simon J. Puglisi, and Jaakko Vuohtoniemi. Succinct k-mer Set Representations Using Subset Rank Queries on the Spectral Burrows-Wheeler Transform (SBWT), May 2022. URL https://www.biorxiv.org/content/10.1101/2022.05.19.492613v1. Pages: 2022.05.19.492613 Section: New Results. [page 15.]

[82] Rayan Chikhi and Guillaume Rizk. Space-Efficient and Exact de Bruijn Graph Representation Based on a Bloom Filter. In Ben Raphael and Jijun Tang, editors, *Algorithms in Bioinformatics*, Lecture Notes in Computer Science, pages 236–248, Berlin, Heidelberg, 2012. Springer. ISBN 978-3-642-33122-0. doi: 10.1007/978-3-642-33122-0_19. [page 15.]

[83] Andrey Prjibelski, Dmitry Antipov, Dmitry Meleshko, Alla Lapidus, and Anton Korobeynikov. Using SPAdes De Novo Assembler. *Current Protocols in Bioinformatics*, 70(1):e102, 2020. ISSN 1934-340X. doi: 10.1002/cpbi.102. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cpbi.102. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpbi.102. [page 15.]

[84] Jamshed Khan and Rob Patro. Cuttlefish: fast, parallel and low-memory compaction of de Bruijn graphs from large-scale genome collections. *Bioinformatics*, 37(Supplement_1): i177–i186, July 2021. ISSN 1367-4803. doi: 10.1093/bioinformatics/btab309. URL https://doi.org/10.1093/bioinformatics/btab309. [page 15.]

[85] Jamshed Khan, Marek Kokot, Sebastian Deorowicz, and Rob Patro. Scalable, ultra-fast, and low-memory construction of compacted de Bruijn graphs with Cuttlefish 2, June 2022. URL https://www.biorxiv.org/content/10.1101/2021.12.14.472718v2. Pages: 2021.12.14.472718 Section: New Results. [page 15.]

[86] Mikhail Karasikov, Harun Mustafa, Daniel Danciu, Marc Zimmermann, Christopher Barber, Gunnar Rätsch, and André Kahles. MetaGraph: Indexing and Analysing Nucleotide Archives at Petabase-scale. preprint, Bioinformatics, October 2020. URL http://biorxiv.org/lookup/doi/10.1101/2020.10.01.322164. [page 15.]

[87] Brad Solomon and Carl Kingsford. Improved Search of Large Transcriptomic Sequencing Databases Using Split Sequence Bloom Trees. *Journal of Computational Biology*, 25(7): 755–765, July 2018. ISSN 1066-5277. doi: 10.1089/cmb.2017.0265. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6067102/. [pages 15, 30, and 31.]

[88] Prashant Pandey, Michael A. Bender, Rob Johnson, and Rob Patro. A General-Purpose Counting Filter: Making Every Bit Count. In *Proceedings of the 2017 ACM International*

Conference on Management of Data, SIGMOD '17, pages 775–787, New York, NY, USA, May 2017. Association for Computing Machinery. ISBN 978-1-4503-4197-4. doi: 10.1145/3035918.3035963. URL https://doi.org/10.1145/3035918.3035963. [pages 15, 21, and 22.]

[89] Prashant Pandey, Michael A Bender, Rob Johnson, and Rob Patro. Squeakr: an exact and approximate k-mer counting system. *Bioinformatics*, 34(4):568–575, February 2018. ISSN 1367-4803. doi: 10.1093/bioinformatics/btx636. URL https://doi.org/10.1093/bioinformatics/btx636. [pages 15, 21, 23, and 37.]

[90] William Stallings. *Computer Organization and Architecture: Designing for Performance*. Prentice Hall Press, USA, 8th edition, 2009. ISBN 978-0-13-607373-4. [page 16.]

[91] Adam Kirsch and Michael Mitzenmacher. Less hashing, same performance: Building a better Bloom filter. *Random Structures and Algorithms*, 33(2):187–218, September 2008. ISSN 10429832, 10982418. doi: 10.1002/rsa.20208. URL https://onlinelibrary.wiley.com/doi/10.1002/rsa.20208. [page 16.]

[92] Intel® Intrinsics Guide, . URL https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html. [pages 17 and 101.]

[93] Rasmus Pagh and Flemming Friche Rodler. Cuckoo Hashing. In Friedhelm Meyer auf der Heide, editor, *Algorithms — ESA 2001*, Lecture Notes in Computer Science, pages 121–133, Berlin, Heidelberg, 2001. Springer. ISBN 978-3-540-44676-7. doi: 10.1007/3-540-44676-1_10. [pages 17 and 18.]

[94] Jens Zentgraf, Henning Timm, and Sven Rahmann. Cost-optimal assignment of elements in genome-scale multi-way bucketed Cuckoo hash tables. In *2020 Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX)*, Proceedings, pages 186–198. Society for Industrial and Applied Mathematics, December 2019. doi: 10.1137/1.9781611976007.15. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611976007.15. [page 18.]

[95] Bin Fan, Dave G. Andersen, Michael Kaminsky, and Michael D. Mitzenmacher. Cuckoo Filter: Practically Better Than Bloom. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, CoNEXT '14, pages 75–88, New York, NY, USA, December 2014. Association for Computing Machinery. ISBN 978-1-4503-3279-8. doi: 10.1145/2674005.2674994. URL https://doi.org/10.1145/2674005.2674994. [pages 18 and 19.]

[96] J.G. Clerry. Compact Hash Tables Using Bidirectional Linear Probing. *IEEE Transactions on Computers*, C-33(9):828–834, September 1984. ISSN 1557-9956. doi: 10.1109/TC.1984.1676499. Conference Name: IEEE Transactions on Computers. [page 19.]

[97] Michael A. Bender, Martin Farach-Colton, Rob Johnson, Russell Kraner, Bradley C. Kuszmaul, Dzejla Medjedovic, Pablo Montes, Pradeep Shetty, Richard P. Spillane, and Erez Zadok. Don't thrash: how to cache your hash on flash. *Proceedings of the VLDB Endowment*, 5(11):1627–1637, July 2012. ISSN 2150-8097. doi: 10.14778/2350229.2350275. URL https://dl.acm.org/doi/10.14778/2350229.2350275. [pages 19 and 20.]

[98] Donald Knuth. *The Art Of Computer Programming, vol. 3: Sorting And Searching*. Addison-Wesley, 1973. [page 19.]

[99] Gonzalo Navarro and Eliana Providel. Fast, Small, Simple Rank/Select on Bitmaps. In Ralf Klasing, editor, *Experimental Algorithms*, Lecture Notes in Computer Science, pages 295–306, Berlin, Heidelberg, 2012. Springer. ISBN 978-3-642-30850-5. doi: 10.1007/978-3-642-30850-5_26. [page 21.]

[100] Prashant Pandey, Fatemeh Almodaresi, Michael A. Bender, Michael Ferdman, Rob Johnson, and Rob Patro. Mantis: A Fast, Small, and Exact Large-Scale Sequence-Search Index. *Cell Systems*, 7(2):201–207.e4, August 2018. ISSN 2405-4712. doi: 10.1016/j.cels.2018.05.021. URL https://www.cell.com/cell-systems/abstract/S2405-4712(18)30239-4. Publisher: Elsevier. [pages 21, 23, and 24.]

[101] Rajeev Raman, Venkatesh Raman, and Srinivasa Rao Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Transactions on Algorithms*, 3(4):43–es, November 2007. ISSN 1549-6325. doi: 10.1145/1290672.1290680. URL https://doi.org/10.1145/1290672.1290680. [pages 21 and 28.]

[102] Daniel Lemire, Owen Kaser, Nathan Kurz, Luca Deri, Chris O'Hara, François Saint-Jacques, and Gregory Ssi-Yan-Kai. Roaring Bitmaps: Implementation of an Optimized Software Library. *Software: Practice and Experience*, 48(4):867–895, April 2018. ISSN 00380644. doi: 10.1002/spe.2560. URL http://arxiv.org/abs/1709.07821. arXiv:1709.07821 [cs]. [page 21.]

[103] Thomas Mueller Graf and Daniel Lemire. Xor Filters: Faster and Smaller Than Bloom and Cuckoo Filters. *ACM Journal of Experimental Algorithmics*, 25:1–16, December 2020. ISSN 1084-6654, 1084-6654. doi: 10.1145/3376122. URL http://arxiv.org/abs/1912.08258. arXiv:1912.08258 [cs]. [page 22.]

[104] Justin Zobel, Alistair Moffat, and Kotagiri Ramamohanarao. Inverted files versus signature files for text indexing. *ACM Transactions on Database Systems*, 23(4):453–490, December

1998. ISSN 0362-5915. doi: 10.1145/296854.277632. URL https://doi.org/10.1145/296854.277632. [page 23.]

[105] Camille Marchet, Christina Boucher, Simon J. Puglisi, Paul Medvedev, Mikaël Salson, and Rayan Chikhi. Data structures based on k-mers for querying large collections of sequencing data sets. *Genome Research*, 31(1):1–12, January 2021. ISSN 1088-9051, 1549-5469. doi: 10.1101/gr.260604.119. URL https://genome.cshlp.org/content/31/1/1. Company: Cold Spring Harbor Laboratory Press Distributor: Cold Spring Harbor Laboratory Press Institution: Cold Spring Harbor Laboratory Press Label: Cold Spring Harbor Laboratory Press Publisher: Cold Spring Harbor Lab. [pages 23 and 33.]

[106] Harry K. T. Wong, Hsiu-Fen Liu, Frank Olken, Doron Rotem, and Linda Wong. Bit transposed files. In *Proceedings of the 11th international conference on Very Large Data Bases - Volume 11*, VLDB '85, pages 448–457, Stockholm, Sweden, August 1985. VLDB Endowment. [page 24.]

[107] Phelim Bradley, Henk C. den Bakker, Eduardo P. C. Rocha, Gil McVean, and Zamin Iqbal. Ultrafast search of all deposited bacterial and viral genomic data. *Nature Biotechnology*, 37(2):152–159, February 2019. ISSN 1546-1696. doi: 10.1038/s41587-018-0010-1. URL https://www.nature.com/articles/s41587-018-0010-1. Number: 2 Publisher: Nature Publishing Group. [pages 24 and 25.]

[108] Timo Bingmann, Phelim Bradley, Florian Gauger, and Zamin Iqbal. COBS: A Compact Bit-Sliced Signature Index. In Nieves R. Brisaboa and Simon J. Puglisi, editors, *String Processing and Information Retrieval*, Lecture Notes in Computer Science, pages 285–303, Cham, 2019. Springer International Publishing. ISBN 978-3-030-32686-9. doi: 10.1007/978-3-030-32686-9_21. [pages 25 and 26.]

[109] Sanjay K. Srikakulam, Sebastian Keller, Fawaz Dabbaghie, Robert Bals, and Olga V. Kalinina. MetaProFi: A protein-based Bloom filter for storing and querying sequence data for accurate identification of functionally relevant genetic variants, August 2021. URL https://www.biorxiv.org/content/10.1101/2021.08.12.456081v1. Pages: 2021.08.12.456081 Section: New Results. [page 26.]

[110] Zarr — zarr 2.12.0 documentation, . URL https://zarr.readthedocs.io/en/stable/. [page 26.]

[111] Zstandard - Real-time data compression algorithm, . URL https://facebook.github.io/zstd/. [page 26.]

[112] Donald E. Knuth. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*. Addison-Wesley, Reading, Mass., third edition, 1997. ISBN 0-201-89683-4 978-0-201-89683-1. [page 26.]

[113] Chen Sun, Robert S. Harris, Rayan Chikhi, and Paul Medvedev. AllSome Sequence Bloom Trees. *Journal of Computational Biology*, 25(5):467–479, May 2018. doi: 10.1089/cmb.2017.0258. URL https://www.liebertpub.com/doi/10.1089/cmb.2017.0258. Publisher: Mary Ann Liebert, Inc., publishers. [pages 29 and 30.]

[114] Robert S Harris and Paul Medvedev. Improved representation of sequence bloom trees. *Bioinformatics*, 36(3):721–727, February 2020. ISSN 1367-4803. doi: 10.1093/bioinformatics/btz662. URL https://doi.org/10.1093/bioinformatics/btz662. [pages 31, 32, 33, and 86.]

[115] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, July 1948. ISSN 0005-8580. doi: 10.1002/j.1538-7305.1948.tb01338.x. Conference Name: The Bell System Technical Journal. [page 32.]

[116] Rayan Chikhi, Jan Holub, and Paul Medvedev. Data Structures to Represent a Set of k-long DNA Sequences. *ACM Computing Surveys*, 54(1):17:1–17:22, March 2021. ISSN 0360-0300. doi: 10.1145/3445967. URL https://doi.org/10.1145/3445967. [page 33.]

[117] Jérôme Audoux, Nicolas Philippe, Rayan Chikhi, Mikaël Salson, Mélina Gallopin, Marc Gabriel, Jérémy Le Coz, Emilie Drouineau, Thérèse Commes, and Daniel Gautheret. DE-kupl: exhaustive capture of biological variation in RNA-seq data through k-mer decomposition. *Genome Biology*, 18(1):243, December 2017. ISSN 1474-760X. doi: 10.1186/s13059-017-1372-2. URL https://doi.org/10.1186/s13059-017-1372-2. [pages 36 and 50.]

[118] Guillaume Marçais and Carl Kingsford. A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics*, 27(6):764–770, March 2011. ISSN 1367-4803. doi: 10.1093/bioinformatics/btr011. URL https://doi.org/10.1093/bioinformatics/btr011. [pages 37 and 68.]

[119] Páll Melsted and Jonathan K. Pritchard. Efficient counting of k-mers in DNA sequences using a bloom filter. *BMC Bioinformatics*, 12(1):333, August 2011. ISSN 1471-2105. doi: 10.1186/1471-2105-12-333. URL https://doi.org/10.1186/1471-2105-12-333. [page 37.]

[120] Maurice Herlihy and J. Eliot B. Moss. Transactional memory: architectural support for lock-free data structures. In *Proceedings of the 20th annual international symposium on*

computer architecture, ISCA '93, pages 289–300, New York, NY, USA, May 1993. Association for Computing Machinery. ISBN 978-0-8186-3810-7. doi: 10.1145/165123.165164. URL https://doi.org/10.1145/165123.165164. [page 37.]

[121] Guillaume Collet, Guillaume Rizk, Rayan Chikhi, and Dominique Lavenier. MINIA on a Raspberry Pi, Assembling a 100 Mbp Genome on a Credit Card Sized Computer. page 2. [page 37.]

[122] Michael Roberts, Brian R. Hunt, James A. Yorke, Randall A. Bolanos, and Arthur L. Delcher. A Preprocessor for Shotgun Assembly of Large Genomes. *Journal of Computational Biology*, 11(4):734–752, August 2004. doi: 10.1089/cmb.2004.11.734. URL https://www.liebertpub.com/doi/abs/10.1089/cmb.2004.11.734. Publisher: Mary Ann Liebert, Inc., publishers. [pages 38 and 43.]

[123] Jason R. Miller, Arthur L. Delcher, Sergey Koren, Eli Venter, Brian P. Walenz, Anushka Brownley, Justin Johnson, Kelvin Li, Clark Mobarry, and Granger Sutton. Aggressive assembly of pyrosequencing reads with mates. *Bioinformatics*, 24(24):2818–2824, December 2008. ISSN 1367-4803. doi: 10.1093/bioinformatics/btn548. URL https://doi.org/10.1093/bioinformatics/btn548. [page 38.]

[124] Guillaume Rizk, Dominique Lavenier, and Rayan Chikhi. DSK: k-mer counting with very low memory usage. *Bioinformatics (Oxford, England)*, 29(5):652–653, March 2013. ISSN 1367-4811. doi: 10.1093/bioinformatics/btt020. [pages 38 and 43.]

[125] Sebastian Deorowicz, Agnieszka Debudaj-Grabysz, and Szymon Grabowski. Disk-based k-mer counting on a PC. *BMC Bioinformatics*, 14(1):160, May 2013. ISSN 1471-2105. doi: 10.1186/1471-2105-14-160. URL https://doi.org/10.1186/1471-2105-14-160. [pages 38 and 43.]

[126] Sebastian Deorowicz, Marek Kokot, Szymon Grabowski, and Agnieszka Debudaj-Grabysz. KMC 2: fast and resource-frugal k-mer counting. *Bioinformatics*, 31(10):1569–1576, May 2015. ISSN 1367-4803. doi: 10.1093/bioinformatics/btv022. URL https://doi.org/10.1093/bioinformatics/btv022. [pages 38 and 44.]

[127] Marek Kokot, Maciej Długosz, and Sebastian Deorowicz. KMC 3: counting and manipulating k-mer statistics. *Bioinformatics*, 33(17):2759–2761, September 2017. ISSN 1367-4803. doi: 10.1093/bioinformatics/btx304. URL https://doi.org/10.1093/bioinformatics/btx304. [pages 38 and 68.]

[128] David R. Musser. Introspective Sorting and Selection Algorithms. *Software: Practice and Experience*, 27(8):983–993, 1997. ISSN 1097-024X. doi:

10.1002/(SICI)1097-024X(199708)27:8<983::AID-SPE117>3.0.CO;2-#. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/%28SICI%291097-024X%28199708%2927%3A8%3C983%3A%3AAID-SPE117%3E3.0.CO%3B2-%23. [page 44.]

[129] David Laehnemann, Arndt Borkhardt, and Alice Carolyn McHardy. Denoising DNA deep sequencing data—high-throughput sequencing errors and their correction. *Briefings in Bioinformatics*, 17(1):154–179, January 2016. ISSN 1467-5463. doi: 10.1093/bib/bbv029. URL https://doi.org/10.1093/bib/bbv029. [page 46.]

[130] Alexey Vorobev, Marion Dupouy, Quentin Carradec, Tom O. Delmont, Anita Annamalé, Patrick Wincker, and Eric Pelletier. Transcriptome reconstruction and functional analysis of eukaryotic marine plankton communities via high-throughput metagenomics and metatranscriptomics. *Genome Research*, 30(4):647–659, April 2020. ISSN 1088-9051, 1549-5469. doi: 10.1101/gr.253070.119. URL https://genome.cshlp.org/content/30/4/647. Company: Cold Spring Harbor Laboratory Press Distributor: Cold Spring Harbor Laboratory Press Institution: Cold Spring Harbor Laboratory Press Label: Cold Spring Harbor Laboratory Press Publisher: Cold Spring Harbor Lab. [page 47.]

[131] Téo Lemane, Paul Medvedev, Rayan Chikhi, and Pierre Peterlongo. kmtricks: Efficient and flexible construction of Bloom filters for large sequencing data collections. *Bioinformatics Advances*, page vbac029, April 2022. ISSN 2635-0041. doi: 10.1093/bioadv/vbac029. URL https://doi.org/10.1093/bioadv/vbac029. [pages 48, 64, 67, 70, 74, and 77.]

[132] Daniel J Richter, Romain Watteaux, Thomas Vannier, Jade Leconte, Paul Frémont, Gabriel Reygondeau, Nicolas Maillet, Nicolas Henry, Gaëtan Benoit, Antonio Fernandez-Guerra, Samir Suweis, Romain Narci, Cédric Berney, Damien Eveillard, Frédérick F. Gavory, Lionel Guidi, Karine Labadie, Eric Mahieu, Julie Poulain, Sarah Romac, Simon Roux, Céline Dimier, Stefanie Kandels, Marc Picheral, Sarah Searson, Stéphane Pesant, Jean-Marc Aury, Jennifer Brum, Claire Lemaitre, Eric Pelletier, Peer Bork, Shinichi Sunagawa, Lee Karp-Boss, Chris Bowler, Matthew Sullivan, Eric Karsenti, Mahendra Mariadassou, Ian Probert, Pierre Peterlongo, Patrick Wincker, Colomban de Vargas, Maurizio Ribera d'Alcalà, Daniele Iudicone, and Olivier Jaillon. Genomic evidence for global ocean plankton biogeography shaped by large-scale current systems. *eLife*, 2022. doi: 10.1101/867739. URL https://hal.inria.fr/hal-02399723. Publisher: eLife Sciences Publication. [page 49.]

[133] Arthur Korte and Ashley Farlow. The advantages and limitations of trait analysis with GWAS: a review. *Plant Methods*, 9(1):29, July 2013. ISSN 1746-4811. doi: 10.1186/1746-4811-9-29. URL https://doi.org/10.1186/1746-4811-9-29. [page 49.]

[134] Annalisa Buniello, Jacqueline A L MacArthur, Maria Cerezo, Laura W Harris, James Hayhurst, Cinzia Malangone, Aoife McMahon, Joannella Morales, Edward Mountjoy, Elliot Sollis, Daniel Suveges, Olga Vrousgou, Patricia L Whetzel, Ridwan Amode, Jose A Guillen, Harpreet S Riat, Stephen J Trevanion, Peggy Hall, Heather Junkins, Paul Flicek, Tony Burdett, Lucia A Hindorff, Fiona Cunningham, and Helen Parkinson. The NHGRI-EBI GWAS Catalog of published genome-wide association studies, targeted arrays and summary statistics 2019. *Nucleic Acids Research*, 47(D1):D1005–D1012, January 2019. ISSN 0305-1048. doi: 10.1093/nar/gky1120. URL https://doi.org/10.1093/nar/gky1120. [page 49.]

[135] Pawe\l Stankiewicz and James R. Lupski. Structural Variation in the Human Genome and its Role in Disease. *Annual Review of Medicine*, 61(1):437–455, 2010. doi: 10.1146/annurev-med-100708-204735. URL https://doi.org/10.1146/annurev-med-100708-204735. _eprint: https://doi.org/10.1146/annurev-med-100708-204735. [page 50.]

[136] Ryan L. Collins, Harrison Brand, Konrad J. Karczewski, Xuefang Zhao, Jessica Alföldi, Laurent C. Francioli, Amit V. Khera, Chelsea Lowther, Laura D. Gauthier, Harold Wang, Nicholas A. Watts, Matthew Solomonson, Anne O'Donnell-Luria, Alexander Baumann, Ruchi Munshi, Mark Walker, Christopher W. Whelan, Yongqing Huang, Ted Brookings, Ted Sharpe, Matthew R. Stone, Elise Valkanas, Jack Fu, Grace Tiao, Kristen M. Laricchia, Valentin Ruano-Rubio, Christine Stevens, Namrata Gupta, Caroline Cusick, Lauren Margolin, Kent D. Taylor, Henry J. Lin, Stephen S. Rich, Wendy S. Post, Yii-Der Ida Chen, Jerome I. Rotter, Chad Nusbaum, Anthony Philippakis, Eric Lander, Stacey Gabriel, Benjamin M. Neale, Sekar Kathiresan, Mark J. Daly, Eric Banks, Daniel G. MacArthur, and Michael E. Talkowski. A structural variation reference for medical and population genetics. *Nature*, 581(7809):444–451, May 2020. ISSN 1476-4687. doi: 10.1038/s41586-020-2287-8. URL https://www.nature.com/articles/s41586-020-2287-8. Number: 7809 Publisher: Nature Publishing Group. [page 50.]

[137] Nick Patterson, Alkes L. Price, and David Reich. Population Structure and Eigenanalysis. *PLOS Genetics*, 2(12):e190, December 2006. ISSN 1553-7404. doi: 10.1371/journal.pgen.0020190. URL https://journals.plos.org/plosgenetics/article?id=10.1371/journal.pgen.0020190. Publisher: Public Library of Science. [page 54.]

[138] Alkes L. Price, Nick J. Patterson, Robert M. Plenge, Michael E. Weinblatt, Nancy A. Shadick, and David Reich. Principal components analysis corrects for stratification in genome-wide association studies. *Nature Genetics*, 38(8):904–909, August 2006. ISSN 1546-1718. doi: 10.1038/ng1847. URL https://www.nature.com/articles/ng1847. Number: 8 Publisher: Nature Publishing Group. [page 54.]

[139] Bonferroni C. Teoria statistica delle classi e calcolo delle probabilita. *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commericiali di Firenze*, 8:3–62, 1936. URL https://cir.nii.ac.jp/crid/1570009749360424576. [page 54.]

[140] Zbyněk Šidák. Rectangular Confidence Regions for the Means of Multivariate Normal Distributions. *Journal of the American Statistical Association*, 62(318):626–633, June 1967. ISSN 0162-1459. doi: 10.1080/01621459.1967.10482935. URL https://doi.org/10.1080/01621459.1967.10482935. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/01621459.1967.10482935. [page 54.]

[141] Yoav Benjamini and Yosef Hochberg. Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. *Journal of the Royal Statistical Society. Series B (Methodological)*, 57(1):289–300, 1995. ISSN 0035-9246. URL https://www.jstor.org/stable/2346101. Publisher: [Royal Statistical Society, Wiley]. [page 54.]

[142] Xiang Zhou and Matthew Stephens. Genome-wide efficient mixed-model analysis for association studies. *Nature Genetics*, 44(7):821–824, July 2012. ISSN 1546-1718. doi: 10.1038/ng.2310. URL https://www.nature.com/articles/ng.2310. Number: 7 Publisher: Nature Publishing Group. [page 55.]

[143] Sarah G. Earle, Chieh-Hsi Wu, Jane Charlesworth, Nicole Stoesser, N. Claire Gordon, Timothy M. Walker, Chris C. A. Spencer, Zamin Iqbal, David A. Clifton, Katie L. Hopkins, Neil Woodford, E. Grace Smith, Nazir Ismail, Martin J. Llewelyn, Tim E. Peto, Derrick W. Crook, Gil McVean, A. Sarah Walker, and Daniel J. Wilson. Identifying lineage effects when controlling for population structure improves power in bacterial association studies. *Nature Microbiology*, 1(5):1–8, April 2016. ISSN 2058-5276. doi: 10.1038/nmicrobiol.2016.41. URL https://www.nature.com/articles/nmicrobiol201641. Number: 5 Publisher: Nature Publishing Group. [pages 56 and 57.]

[144] Téo Lemane, Rayan Chikhi, and Pierre Peterlongo. kmdiff, large-scale and user-friendly differential k-mer analyses. *Bioinformatics*, Oct 2022. ISSN 1367-4803. doi: 10.1093/bioinformatics/btac689. [page 57.]

[145] Sajad Babakhani and Mana Oloomi. Transposons: the agents of antibiotic resistance in bacteria. *Journal of Basic Microbiology*, 58(11):905–917, 2018. ISSN 1521-4028. doi: 10.1002/jobm.201800204. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/jobm.201800204. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/jobm.201800204. [page 57.]

[146] Shaun D. Jackman, Benjamin P. Vandervalk, Hamid Mohamadi, Justin Chu, Sarah Yeo, S. Austin Hammond, Golnaz Jahesh, Hamza Khan, Lauren Coombe, Rene L. Warren,

and Inanc Birol. ABySS 2.0: resource-efficient assembly of large genomes using a Bloom filter. *Genome Research*, 27(5):768–777, May 2017. ISSN 1088-9051, 1549-5469. doi: 10.1101/gr.214346.116. URL https://genome.cshlp.org/content/27/5/768. Company: Cold Spring Harbor Laboratory Press Distributor: Cold Spring Harbor Laboratory Press Institution: Cold Spring Harbor Laboratory Press Label: Cold Spring Harbor Laboratory Press Publisher: Cold Spring Harbor Lab. [page 57.]

[147] Eric W. Sayers, Evan E. Bolton, J. Rodney Brister, Kathi Canese, Jessica Chan, Donald C. Comeau, Ryan Connor, Kathryn Funk, Chris Kelly, Sunghwan Kim, Tom Madej, Aron Marchler-Bauer, Christopher Lanczycki, Stacy Lathrop, Zhiyong Lu, Francoise Thibaud-Nissen, Terence Murphy, Lon Phan, Yuri Skripchenko, Tony Tse, Jiyao Wang, Rebecca Williams, Barton W. Trawick, Kim D. Pruitt, and Stephen T. Sherry. Database resources of the national center for biotechnology information. *Nucleic Acids Research*, 50(D1): D20–D26, January 2022. ISSN 1362-4962. doi: 10.1093/nar/gkab1112. [page 57.]

[148] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402, September 1997. ISSN 0305-1048. doi: 10.1093/nar/25.17.3389. [page 57.]

[149] Yann Collet. xxHash - Extremely fast hash algorithm, August 2022. URL https://github.com/Cyan4973/xxHash. original-date: 2014-04-30T23:32:49Z. [page 65.]

[150] Simon Gog. SDSL - Succinct Data Structure Library, 2013. URL https://github.com/simongog/sdsl-lite. original-date: 2013-02-28T22:34:07Z. [pages 65 and 85.]

[151] Eric Karsenti, Silvia G. Acinas, Peer Bork, Chris Bowler, Colomban De Vargas, Jeroen Raes, Matthew Sullivan, Detlev Arendt, Francesca Benzoni, Jean-Michel Claverie, Mick Follows, Gaby Gorsky, Pascal Hingamp, Daniele Iudicone, Olivier Jaillon, Stefanie Kandels-Lewis, Uros Krzic, Fabrice Not, Hiroyuki Ogata, Stéphane Pesant, Emmanuel Georges Reynaud, Christian Sardet, Michael E. Sieracki, Sabrina Speich, Didier Velayoudon, Jean Weissenbach, Patrick Wincker, and the Tara Oceans Consortium. A Holistic Approach to Marine Eco-Systems Biology. *PLOS Biology*, 9(10):e1001177, October 2011. ISSN 1545-7885. doi: 10.1371/journal.pbio.1001177. URL https://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.1001177. Publisher: Public Library of Science. [page 71.]

[152] Emilie Villar, Thomas Vannier, Caroline Vernette, Magali Lescot, Miguelangel Cuenca, Aurélien Alexandre, Paul Bachelerie, Thomas Rosnet, Eric Pelletier, Shinichi Sunagawa, and Pascal Hingamp. The Ocean Gene Atlas: exploring the biogeography of plankton

genes online. *Nucleic Acids Research*, 46(W1):W289–W295, July 2018. ISSN 0305-1048. doi: 10.1093/nar/gky376. URL https://doi.org/10.1093/nar/gky376. [page 72.]

[153] Caroline Vernette, Julien Lecubin, Pablo Sánchez, Tara Oceans Coordinators, Shinichi Sunagawa, Tom O Delmont, Silvia G Acinas, Eric Pelletier, Pascal Hingamp, and Magali Lescot. The Ocean Gene Atlas v2.0: online exploration of the biogeography and phylogeny of plankton genes. *Nucleic Acids Research*, 50(W1):W516–W526, July 2022. ISSN 0305-1048. doi: 10.1093/nar/gkac420. URL https://doi.org/10.1093/nar/gkac420. [page 72.]

[154] S R Eddy. Profile hidden Markov models. *Bioinformatics*, 14(9):755–763, January 1998. ISSN 1367-4803. doi: 10.1093/bioinformatics/14.9.755. URL https://doi.org/10.1093/bioinformatics/14.9.755. [page 72.]

[155] Benjamin Buchfink, Chao Xie, and Daniel H. Huson. Fast and sensitive protein alignment using DIAMOND. *Nature Methods*, 12(1):59–60, January 2015. ISSN 1548-7105. doi: 10.1038/nmeth.3176. URL https://www.nature.com/articles/nmeth.3176. Number: 1 Publisher: Nature Publishing Group. [page 72.]

[156] Travis J. Wheeler and Sean R. Eddy. nhmmer: DNA homology search with profile HMMs. *Bioinformatics*, 29(19):2487–2489, October 2013. ISSN 1367-4803. doi: 10.1093/bioinformatics/btt403. URL https://doi.org/10.1093/bioinformatics/btt403. [page 72.]

[157] Stéphane Pesant, Fabrice Not, Marc Picheral, Stefanie Kandels-Lewis, Noan Le Bescot, Gabriel Gorsky, Daniele Iudicone, Eric Karsenti, Sabrina Speich, Romain Troublé, Céline Dimier, and Sarah Searson. Open science resources for the discovery and analysis of Tara Oceans data. *Scientific Data*, 2(1):150023, May 2015. ISSN 2052-4463. doi: 10.1038/sdata.2015.23. URL https://www.nature.com/articles/sdata201523. Number: 1 Publisher: Nature Publishing Group. [page 72.]

[158] Andy B. Yoo, Morris A. Jette, and Mark Grondona. SLURM: Simple Linux Utility for Resource Management. In Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, Lecture Notes in Computer Science, pages 44–60, Berlin, Heidelberg, 2003. Springer. ISBN 978-3-540-39727-4. doi: 10.1007/10968987_3. [page 84.]

[159] Daniel J Richter, Romain Watteaux, Thomas Vannier, Jade Leconte, Paul Frémont, Gabriel Reygondeau, Nicolas Maillet, Nicolas Henry, Gaëtan Benoit, Ophélie Da Silva, Tom O Delmont, Antonio Fernàndez-Guerra, Samir Suweis, Romain Narci, Cédric Berney, Damien Eveillard, Frederick Gavory, Lionel Guidi, Karine Labadie, Eric Mahieu, Julie Poulain, Sarah Romac, Simon Roux, Céline Dimier, Stefanie Kandels, Marc Picheral,

Sarah Searson, Tara Oceans Coordinators, Stéphane Pesant, Jean-Marc Aury, Jennifer R Brum, Claire Lemaitre, Eric Pelletier, Peer Bork, Shinichi Sunagawa, Fabien Lombard, Lee Karp-Boss, Chris Bowler, Matthew B Sullivan, Eric Karsenti, Mahendra Mariadassou, Ian Probert, Pierre Peterlongo, Patrick Wincker, Colomban de Vargas, Maurizio Ribera d'Alcalà, Daniele Iudicone, and Olivier Jaillon. Genomic evidence for global ocean plankton biogeography shaped by large-scale current systems. *eLife*, 11:e78129, August 2022. ISSN 2050-084X. doi: 10.7554/eLife.78129. URL https://doi.org/10.7554/eLife.78129. Publisher: eLife Sciences Publications, Ltd. [page 91.]

[160] Yann Collet. LZ4 - Extremely fast compression, August 2022. URL https://github.com/lz4/lz4. original-date: 2014-03-25T15:52:21Z. [page 100.]

[161] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, May 1977. ISSN 1557-9654. doi: 10.1109/TIT.1977.1055714. Conference Name: IEEE Transactions on Information Theory. [page 100.]

[162] Alistair Moffat. Huffman Coding. *ACM Computing Surveys*, 52(4):85:1–85:35, August 2019. ISSN 0360-0300. doi: 10.1145/3342555. URL https://doi.org/10.1145/3342555. [page 100.]

[163] M. Zukowski, S. Heman, N. Nes, and P. Boncz. Super-Scalar RAM-CPU Cache Compression. In *22nd International Conference on Data Engineering (ICDE'06)*, pages 59–59, 2006. doi: 10.1109/ICDE.2006.150. [page 100.]

[164] powturbo. powturbo/TurboPFor-Integer-Compression, July 2022. URL https://github.com/powturbo/TurboPFor-Integer-Compression. original-date: 2014-10-28T21:17:07Z. [page 100.]

[165] Protocol Buffers, . URL https://developers.google.com/protocol-buffers. [page 101.]

[166] Michael Kistler, Michael Perrone, and Fabrizio Petrini. Cell Multiprocessor Communication Network: Built for Speed. *IEEE Micro*, 26(03):10–23, May 2006. ISSN 0272-1732. doi: 10.1109/MM.2006.49. URL https://www.computer.org/csdl/magazine/mi/2006/03/m3010/13rRUzphDAE. Publisher: IEEE Computer Society. [page 101.]

[167] Alexander T. Dilthey, Chirag Jain, Sergey Koren, and Adam M. Phillippy. Strain-level metagenomic assignment and compositional estimation for long reads with MetaMaps. *Nature Communications*, 10(1):3066, July 2019. ISSN 2041-1723. doi: 10.1038/s41467-019-10934-2. URL https://www.nature.com/articles/s41467-019-10934-2. Number: 1 Publisher: Nature Publishing Group. [page 103.]

[168] Daehwan Kim, Li Song, Florian P. Breitwieser, and Steven L. Salzberg. Centrifuge: rapid and sensitive classification of metagenomic sequences. *Genome Research*, October 2016. ISSN 1088-9051, 1549-5469. doi: 10.1101/gr.210641.116. URL https://genome.cshlp.org/content/early/2016/11/16/gr.210641.116. Company: Cold Spring Harbor Laboratory Press Distributor: Cold Spring Harbor Laboratory Press Institution: Cold Spring Harbor Laboratory Press Label: Cold Spring Harbor Laboratory Press Publisher: Cold Spring Harbor Lab. [page 103.]

[169] Derrick E. Wood, Jennifer Lu, and Ben Langmead. Improved metagenomic analysis with Kraken 2. *Genome Biology*, 20(1):257, November 2019. ISSN 1474-760X. doi: 10.1186/s13059-019-1891-0. URL https://doi.org/10.1186/s13059-019-1891-0. [pages 103 and 104.]

[170] Miten Jain, Hugh E. Olsen, Benedict Paten, and Mark Akeson. The Oxford Nanopore MinION: delivery of nanopore sequencing to the genomics community. *Genome Biology*, 17(1):239, November 2016. ISSN 1474-760X. doi: 10.1186/s13059-016-1103-0. URL https://doi.org/10.1186/s13059-016-1103-0. [page 103.]

[171] Karel Břinda, Maciej Sykulski, and Gregory Kucherov. Spaced seeds improve k-mer-based metagenomic classification. *Bioinformatics*, 31(22):3584–3592, November 2015. ISSN 1367-4803. doi: 10.1093/bioinformatics/btv419. URL https://doi.org/10.1093/bioinformatics/btv419. [page 104.]

[172] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012. [page 104.]

[173] Märt Roosaare, Mihkel Vaher, Lauris Kaplinski, Märt Möls, Reidar Andreson, Maarja Lepamets, Triinu Kõressaar, Paul Naaber, Siiri Kõljalg, and Maido Remm. StrainSeeker: fast identification of bacterial strains from raw sequencing reads using user-provided guide trees. *PeerJ*, 5:e3353, 2017. ISSN 2167-8359. doi: 10.7717/peerj.3353. [page 104.]

[174] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin, and 1000 Genome Project Data Processing Subgroup. The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, August 2009. ISSN 1367-4803, 1460-2059. doi: 10.1093/bioinformatics/btp352. URL https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/btp352. [page 106.]

[175] James K Bonfield, John Marshall, Petr Danecek, Heng Li, Valeriu Ohan, Andrew Whitwham, Thomas Keane, and Robert M Davies. HTSlib: C library for reading/writ-

ing high-throughput sequencing data. *GigaScience*, 10(2):giab007, February 2021. ISSN 2047-217X. doi: 10.1093/gigascience/giab007. URL https://doi.org/10.1093/gigascience/giab007. [page 106.]

[176] James A. Fellows Yates, Aida Andrades Valtueña, Åshild J. Vågene, Becky Cribdon, Irina M. Velsko, Maxime Borry, Miriam J. Bravo-Lopez, Antonio Fernandez-Guerra, Eleanor J. Green, Shreya L. Ramachandran, Peter D. Heintzman, Maria A. Spyrou, Alexander Hübner, Abigail S. Gancz, Jessica Hider, Aurora F. Allshouse, Valentina Zaro, and Christina Warinner. Community-curated and standardised metadata of published ancient metagenomic samples with AncientMetagenomeDir. *Scientific Data*, 8 (1):31, January 2021. ISSN 2052-4463. doi: 10.1038/s41597-021-00816-y. URL https://www.nature.com/articles/s41597-021-00816-y. Number: 1 Publisher: Nature Publishing Group. [page 108.]

[177] Stéphane Peyrégne and Kay Prüfer. Present-Day DNA Contamination in Ancient DNA Datasets. *BioEssays: News and Reviews in Molecular, Cellular and Developmental Biology*, 42(9):e2000081, September 2020. ISSN 1521-1878. doi: 10.1002/bies.202000081. [page 108.]

[178] Jordan J. McGhee, Nick Rawson, Barbara A. Bailey, Antonio Fernandez-Guerra, Laura Sisk-Hackworth, and Scott T. Kelley. Meta-SourceTracker: application of Bayesian source tracking to shotgun metagenomics. *PeerJ*, 8:e8783, March 2020. ISSN 2167-8359. doi: 10.7717/peerj.8783. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7100590/. [pages 108 and 109.]

[179] Liat Shenhav, Mike Thompson, Tyler A. Joseph, Leah Briscoe, Ori Furman, David Bogumil, Itzhak Mizrahi, Itsik Pe'er, and Eran Halperin. FEAST: fast expectation-maximization for microbial source tracking. *Nature Methods*, 16(7):627–632, July 2019. ISSN 1548-7105. doi: 10.1038/s41592-019-0431-x. URL https://www.nature.com/articles/s41592-019-0431-x. Number: 7 Publisher: Nature Publishing Group. [pages 108 and 109.]

[180] Shahbaz Raza, Jungman Kim, Michael J. Sadowsky, and Tatsuya Unno. Microbial source tracking using metagenomics and other new technologies. *Journal of Microbiology*, 59 (3):259–269, March 2021. ISSN 1976-3794. doi: 10.1007/s12275-021-0668-9. URL https://doi.org/10.1007/s12275-021-0668-9. [page 108.]

[181] Edoardo Pasolli, Lucas Schiffer, Paolo Manghi, Audrey Renson, Valerie Obenchain, Duy Tin Truong, Francesco Beghini, Faizan Malik, Marcel Ramos, Jennifer B. Dowd, Curtis Huttenhower, Martin Morgan, Nicola Segata, and Levi Waldron. Accessible, curated metagenomic data through ExperimentHub. *Nature Methods*, 14(11):1023–1024,

November 2017. ISSN 1548-7105. doi: 10.1038/nmeth.4468. URL https://www.nature.com/articles/nmeth.4468. Number: 11 Publisher: Nature Publishing Group. [page 108.]

[182] Jonas Coelho Kasmanas, Alexander Bartholomäus, Felipe Borim Corrêa, Tamara Tal, Nico Jehmlich, Gunda Herberth, Martin von Bergen, Peter F. Stadler, André Carlos Ponce de Leon Ferreira de Carvalho, and Ulisses Nunes da Rocha. HumanMetagenomeDB: a public repository of curated and standardized metadata for human metagenomes. *Nucleic Acids Research*, 49(D1):D743–D750, January 2021. ISSN 1362-4962. doi: 10.1093/nar/gkaa1031. [page 108.]

[183] Alex L. Mitchell, Alexandre Almeida, Martin Beracochea, Miguel Boland, Josephine Burgin, Guy Cochrane, Michael R. Crusoe, Varsha Kale, Simon C. Potter, Lorna J. Richardson, Ekaterina Sakharova, Maxim Scheremetjew, Anton Korobeynikov, Alex Shlemov, Olga Kunyavskaya, Alla Lapidus, and Robert D. Finn. MGnify: the microbiome analysis resource in 2020. *Nucleic Acids Research*, 48(D1):D570–D578, January 2020. ISSN 1362-4962. doi: 10.1093/nar/gkz1035. [page 108.]

[184] Enrico Seiler, Svenja Mehringer, Mitra Darvish, Etienne Turc, and Knut Reinert. Raptor: A fast and space-efficient pre-filter for querying very large collections of nucleotide sequences. *iScience*, 24(7):102782, July 2021. ISSN 2589-0042. doi: 10.1016/j.isci.2021.102782. URL https://www.sciencedirect.com/science/article/pii/S2589004221007501. [page 111.]

List of publications

[1] **Téo Lemane**, Rayan Chikhi, and Pierre Peterlongo. kmdiff, large-scale and user-friendly differential k-mer analyses. *Bioinformatics*, Oct 2022. ISSN 1367-4803. doi: 10.1093/bioinformatics/btac689.

[2] Camila Duitama Gonzàles, Riccardo Vicedomini, **Téo Lemane**, Nicolas Rascovan, Hugues Richard, and Rayan Chikhi. Microbial source tracking for contamination assessment of ancient oral samples using k-mer-based methods. *Microbiome*, Submitted.

[3] Yoann Dufresne, **Téo Lemane**, Pierre Marijon, Pierre Peterlongo, Amatur Rahman, Marek Kokot, Paul Medvedev, Sebastian Deorowicz, and Rayan Chikhi. The k-mer file format: a standardized and compact disk representation of sets of k-mers. *Bioinformatics*, Jul 2022. ISSN 1367-4803. doi: 10.1093/bioinformatics/btac528.

[4] **Téo Lemane**, Paul Medvedev, Rayan Chikhi, and Pierre Peterlongo. kmtricks: Efficient and flexible construction of Bloom filters for large sequencing data collections. *Bioinformatics Advances*, 2022. ISSN 2635-0041. doi: 10.1093/bioadv/vbac029.

[5] Grégoire Siekaniec, Emeline Roux, **Téo Lemane**, Eric Guédon, and Jacques Nicolas. Identification of isolated or mixed strains from long reads: a challenge met on Streptococcus thermophilus using a MinION sequencer. *Microbial Genomics*, 7(11):654, 2021. ISSN 20575858. doi: 10.1099/MGEN.0.000654.

DOCTORAT BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

Titre : Indexation et analyse de grandes collections de séquençages via des matrices de k-mers

Mot clés : Séquençage, k-mer, matrice de k-mers, indexation

Résumé : Le 21ème siècle subit un tsunami de données dans de nombreux domaines, notamment en bio-informatique. Ce changement de paradigme nécessite le développement de nouvelles méthodes de traitement capables de passer à l'échelle sur de telles données. Ce travail consiste principalement à considérer des jeux de données massifs provenant du séquençage génomique. Une façon courante de traiter ces données est de les représenter comme un ensemble de mots de taille fixe, appelés k-mers. Les k-mers sont très largement utilisés comme éléments de bases par de nombreuses méthodes d'analyses de données de séquençages. L'enjeu est de pouvoir représenter les k-mers et leurs abondances dans un grand nombre de jeux de données. Une possibilité est la matrice de k-mers, où chaque ligne est un k-mer associé à un vecteur d'abondances. Ces k-mers sont erronées en raison des erreurs de séquençage et doivent être filtrés. La technique habituelle consiste à écarter les k-mers peu abondants. Sur des ensembles de données complexes comme les métagénomes, un tel filtre n'est pas efficace et élimine un trop grand nombre de k-mers. La vision des abondances à travers les échantillons permise par la représentation matricielle permet également une nouvelle procédure de détection des erreurs dans les jeux de données complexes. En résumé, nous explorons le concept de matrice de k-mer et montrons ses capacités en termes de passage à l'échelle au travers de diverses applications, de l'indexation à l'analyse, et proposons différents outils à cette fin. Sur le plan de l'indexation, nos outils ont permis d'indexer un grand ensemble métagénomique du projet Tara Ocean tout en conservant des k-mers rares, habituellement écartés par les techniques de filtrage classiques. En matière d'analyse, notre technique de construction de matrices permet d'accélérer d'un ordre de grandeur l'analyse différentielle de k-mers.

Title: Indexing and analysis of large sequencing collections using k-mer matrices

Keywords: sequencing, k-mer, k-mer matrix, indexing

Abstract: The 21st century is bringing a tsunami of data in many fields, especially in bioinformatics. This paradigm shift requires the development of new processing methods capable of scaling up on such data. This work consists mainly in considering massive tera-scaled datasets from genomic sequencing. A common way to process these data is to represent them as a set of words of a fixed size, called k-mers. The k-mers are widely used as building blocks by many sequencing data analysis techniques. The challenge is to be able to represent the k-mers and their abundances in a large number of datasets. One possibility is the k-mer matrix, where each row is a k-mer associated with a vector of abundances and each column corresponds to a sample. Some k-mers are erroneous due to sequencing errors and must be discarded. The usual technique consists in discarding low-abundant k-mers. On complex datasets such as metagenomes, such a filter is not efficient and discards too many k-mers. The holistic view of abundances across samples allowed by the matrix representation also enables a new procedure for error detection on such datasets. In summary, we explore the concept of k-mer matrix and show its scalability in various applications, from indexing to analysis, and propose different tools for this purpose. On the indexing side, our tools have allowed indexing a large metagenomic dataset from the Tara Ocean project while keeping additional k-mers, usually discarded by the classical k-mer filtering technique. The next and important step is to make the index publicly available. On the analysis side, our matrix construction technique enables to speed up a differential k-mer analysis of a state-of-the-art tool by an order of magnitude.