UNIVERSITÉ PARIS-EST

Sketching data structures for alignment-free analysis of genomic sequences

Thèse de doctorat de l'Université Paris-Est

École doctorale: MSTIC
Spécialité de doctorat: Informatique
Unité de recherche : Laboratoire d'Informatique Gaspard-Monge

Thèse présentée et soutenue à l'Université Gustave Eiffel, le 29/11/2022, par :

Yoshihiro SHIBUYA

Composition du Jury

Rayan CHIKHI Chargé de recherche expert, Institut Pasteur, Paris	Rapporteur
Yann PONTY DR CNRS, École Polythechnique de Paris	Rapporteur
Karel BŘINDA Starting faculty, INRIA, Rennes	Examinateur
Cinzia PIZZI Associate Professor, University of Padua	Examinatrice
Éric RIVALS DR CNRS, Université de Montpellier	Examinateur
Irena RUSU-ROBINI Professor, Université de Nantes	Examinateur

Encadrement de la thèse

Gregory KUCHEROV DR CNRS, LIGM, Université Gustave Eiffel	Directeur de thèse

Acknowledgment

I would like to express my gratitude to my advisor Gregory Kucherov who was always present to answer my questions and to provide guidance in many different things, from scientific writing to academic bureaucracy.

I would also like to address special thanks to Djamal Belazzougui whose expertise on a vast array of subjects led to many interesting insights developed in this thesis.

I would like to thank Florent Koechlin, Revekka Kyriakoglu and Pablo Rotondo for helping me settle in Paris when I first arrived in France, and for providing a very friendly work environment during my studies.

Thanks to the people of the LIGM lab, from secretaries to professors for creating a hassle-free working environment.

Thank you to my former supervisor Matteo Comin for having introduced me to this research field, and whose involvement during my Master thesis pushed me to undertake doctoral studies.

Last but not least, I have to thank my own family for the invaluable support they provided me during the Covid pandemic and other difficult times.

Abstract

With the advent of Next Generation Sequencing (NGS) technologies, the amount of sequencing data has started to exponentially grow year by year. This data deluge has made the development of new computationally efficient methods a priority in the big-data era of bioinformatics. Dealing with such huge amount of information can be done in two orthogonal ways: by compressing the input of current tools, or by providing specialized algorithms with better scaling capabilities. In practice, these two approaches are often combined in order to take advantage of their respective strenghts.

This thesis proposes new methods exploring these ideas. To this end, we first provide two approaches for the efficient storage and retrieval of static k-mer count information. k-mer counting is a well-known problem in bioinformatics for which many tools have been proposed over the last decade. However, such approaches usually focus on construction speed, without further optimizing for query efficiency nor space. Count information is usually stored using fixed-width representations, potentially wasting a lot of space when small count values are involved.

Both our methods take advantage of the characteristic power-law distribution of k-mer counts to dramatically improve space while providing fully queryable data structures. The first one is *Set-Min* sketch, a variation of the well-known Count-Min sketch. Unlike Count-Min, Set-Min sketch provides better error-space tradeoffs by bounding the expected *cumulative* error by a fraction of the total number of k-mers in the data set. This is in stark contrast to Count-Min where the same bound holds for point-query errors only.

A further development toward optimized data structures for k-mer counts is represented by our family of data structures that we call *locom*. We demonstate how grouping similar counts together using minimizers as a bucketing technique helps in boosting overall compressibility. One of the ingredients of locom are *Bloom-enhanced Compressed Static Functions* (BCSFs), improved versions of the sole Compressed Static Function implementation available at the time of writing. Finally, the combinination of LSH and BCSFs produces a new family of data structures potentially able to represent counters in less space than their empirical zero-order entropy. To the best of our knowledge, no previous method was able to achieve such compression factors.

Similarity estimation of very similar sequences is the other major problem studied in this work. Current methods estimating pairwise Jaccard similarity, such as MinHash sketching, are space-inefficient when sequence composition is too similar. Here we focus on circumventing this limitation by making use of *Invertible Bloom Lookup Tables* (IBLTs), sketches capable of retrieving the symmetric difference of two sets using space proportional to the difference, and not on the number of elements in each set. We combine IBLTs with syncmers-based sampling for further space reductions. This choice is experimentally justified by showing how, compared to minimizers, syncmers lead to unbiased estimations of Jaccard similarity. Moreover, since IBLTs are able to retrieve differences *exactly* we also explore the application of retrieving super-sets of the symmetric difference of whole k-mer sets.

Résumé

Avec l'introduction des technologies de séquençage à haut débit (en anglais: *High Throughput Sequencing* aussi connues sous le nom de *Next Generation Sequencing*), la quantité de données biologiques produites chaque année a vu une croissance exponentielle. Cette surcharge d'information a rendu prioritaire le développement de nouvelles méthodes plus adaptées pour faire face à cette âge de la bioinformatique où les données massives jouent un rôle fondamental. L'objectif peut être atteint de deux façons différentes et pas mutuellement exclusives: avec compression de données ou avec des algorithmes qui passent à l'échelle.

Cette thèse présente de nouvelles méthodes qui explorent ces idées. Tout d'abord, nous proposons deux approches différentes pour stocker et récupérer les compteurs associés aux k-mers. Le comptage de k-mer est un problème bien connu en bioinformatique pour lequel plusieurs outils ont été proposés. Cependant, ces approches sont habituellement focalisées sur la vitesse du comptage, sans tenir en compte l'efficacité des requêtes ou l'espace occupé par les représentations en sortie. En plus, plusieurs outils emploient des compteurs de taille fixe, ce qui résulte en un gaspillage de plusieurs bits en cas de petites valeurs.

Nos méthodes exploitent le fait que les compteurs de k-mers suivent une distribution dite "power-law", ce qui nous permet d'obtenir des structures plus compactes tout en supportant des requêtes efficaces. Notre première structure s'appelle Set-Min, une esquisse de compteurs de k-mers, variante de la bien connue esquisse Count-Min. Contrairement à Count-Min, où les erreurs de chaque estimation sont bornées par une fraction du nombre de k-mers totales présentes dans l'ensemble d'origine, Set-Min borne la *somme* totale attendue des erreurs. En pratique, cette différence fait que les estimations renvoyées par Set-Min soient beaucoup plus précises que celles obtenues avec une esquisse Count-Min de taille identique.

Dans la deuxième partie de cette thèse nous développons davantage ces idées dans le logiciel *locom*. Nous démontrons comment les minimizers peuvent également être employées pour regrouper les compteurs de k-mers similaires. Cette considération est suivie de l'introduction de nos "*Bloom-enhanced Compressed Static Functions*" (BCSFs), une amélioration de la seule implantation de Fonctions Statiques Compressées ("Compressed Static Functions") disponible au moment de rédiger ces lignes. Finalement, avec une combinaison de LSH et BCSFs nous obtenons dans le même algorithme une nouvelle famille de structures de données capables de stocker des comptages en un nombre de bits inférieur à leur entropie d'ordre zéro. À notre connaissance, aucune méthode précédente n'avait réussi à atteindre une telle efficacité.

Avec le dernier sujet de cette thèse nous abordons le problème d'estimation de la similarité entre séquences très similaires. L'esquissage avec MinHash n'est pas adapté à ce type de situation car, pour avoir une probabilité acceptable d'échantillonner des différences, les esquisses se trouvent à devoir avoir des tailles considérables, avec une conséquente perte d'efficacité. Pour cette raison, nous cherchons de contourner cette limitation en utilisant les *Invertible Bloom Lookup Tables* (IBLTs), des esquisses capables de trouver la différence symétrique de deux ensembles avec espace qui dépend de la taille de la différence, et non pas de la taille des ensembles. Par ailleurs, nous démontrons comment combiner l'échantillonage basé sur les syncmers avec les IBLTs pour réduire davantage la taille de nos esquisses tout en évitant le biais propre aux estimateurs de la similarité de Jaccard basés sur les minimizers. Enfin, puisque les IBLTs sont capables de récupérer les différences *exactes* nous explorons la possibilité de calculer efficacement des super-ensembles de différences entre ensembles de k-mers complets.

Contents

List of Figures

List of Tables

List of Acronyms

CS	CountSketch
CM	Count-Min sketch
SMS	Set-Min Sketch
MPHF	Minimal Perfect Hash Function
CSF	Compressed Static Function
BCSF	Bloom-enhanced Compressed Static Function
IBLT	Invertible Bloom Lookup Table

Publications

Journals

1. Yoshihiro Shibuya, Djamal Belazzougui, and Gregory Kucherov. Set-Min Sketch: A Probabilistic Map for Power-Law Distributions with Application to k-Mer Annotation. *Journal of Computational Biology*, 29(2):140–154, February 2022

2. Yoshihiro Shibuya, Djamal Belazzougui, and Gregory Kucherov. Space-efficient representation of genomic k-mer count tables. *Algorithms for Molecular Biology*, 17(1):5, March 2022

International Conferences

1. Yoshihiro Shibuya, Djamal Belazzougui, and Gregory Kucherov. Efficient reconciliation of genomic datasets of high similarity. *22nd International Workshop on Algorithms in Bioinformatics (WABI 2022). Accepted.*

2. Yoshihiro Shibuya, Djamal Belazzougui, and Gregory Kucherov. Set-Min sketch: a probabilistic map for power-law distributions with application to k-mer annotation. RECOMB 2021. Padova, Italy, 2021, *virtual conference.*

3. Yoshihiro Shibuya, Djamal Belazzougui, and Gregory Kucherov. Space-Efficient Representation of Genomic k-Mer Count Tables. In Alessandra Carbone and Mohammed El-Kebir, editors, *21st International Workshop on Algorithms in Bioinformatics (WABI 2021)*, volume 201 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:19, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik

Preprints

1. Yoshihiro Shibuya, Djamal Belazzougui, and Gregory Kucherov. Efficient reconciliation of genomic datasets of high similarity. *bioRxiv*, page 2022.06.07.495186, June 2022. Type: article

2. Yoshihiro Shibuya and Gregory Kucherov. Set-min Sketch: a Probabilistic Map for Power-Law Distributions with Application to k-mer Annotation. *bioRxiv*, page 2020.11.14.382713, November 2020

Workshops

1. Yoshihiro Shibuya, Djamal Belazzougui, and Gregory Kucherov. Space-efficient sets differences with applications to Jaccard estimation. International workshop “Data Structures in Bioinformatics”, Düsseldorf, Germany.

2. Yoshihiro Shibuya, Djamal Belazzougui, and Gregory Kucherov. Space-efficient representation of genomic k-mer count tables. Workshop "SeqBIM 2021", Lyon, France.

3. Yoshihiro Shibuya, Djamal Belazzougui, and Gregory Kucherov. Succinct k-mer tables in practice. International workshop "Data Structures in Bioinformatics", Milan, Italy.

4. Yoshihiro Shibuya, Djamal Belazzougui, and Gregory Kucherov. Set-min sketch: a probabilistic map for power-law distributions with applications to k-mer annotation. National workshop "SeqBIM 2020", Toulouse, France.

5. Yoshihiro Shibuya, and Gregory Kucherov. uANI: whole genome comparison and phylogeny reconstruction using sketching. National workshop "SeqBIM 2019", Marne-la-Vallée, France.

Other works not included in this thesis

1. Yoshihiro Shibuya and Matteo Comin. Better quality score compression through sequence-based quality smoothing. *BMC Bioinformatics*, 20(9):302, November 2019

2. Yoshihiro Shibuya and Matteo Comin. Indexing k-mers in linear space for quality value compression. *Journal of Bioinformatics and Computational Biology*, 17(05):1940011, October 2019

Résumé détaillé de la thèse en français

Structures d'esquissage pour l'analyse sans alignement de séquences biologiques

Introduction

L'estimation de la similarité entre séquences biologiques est l'un des pilliers de la bioinformatique. Aligner et compter les mutations entre séquences est la méthode classique pour atteindre ce but. Dans le cas général, les alignements multiples de séquences seraient le choix idéal mais, en pratique, leur complexité limite leur utilisation à des cas trop simples pour qu'ils puissent être utiles avec les données d'aujourd'hui. Beaucoup d'outils font face à cette limitation en décomposant chaque alignement multiple en une série d'alignements à deux à deux. Le résultat final est alors obtenu en agrégeant les alignements partiels. Cette technique permet une diminution importante du temps de calcul.

Historiquement, les algorithmes classiques d'alignement sont liés à la définition de distance d'édition de chaînes de caractères. La similarité est alors calculée en base aux opérations fondamentales requises pour transformer une chaîne dans l'autre. À noter que, à différence de la similarité entre chaînes quelconques, la comparaison entre séquences d'ADN, ou d'autres molécules biologiques (p.e. ARN), requiert d'assigner des "scores" adaptés aux couples de caractères alignés, pour mieux capturer leur signification biologique. Utiliser la similarité d'édition dans sa forme base, où les mutations sont vues comme indépendantes et toujours de pèse unitaire, peut porter à des conclusions erronées. En général, ces algorithmes utilisent des approches de programmation dynamique afin d'obtenir une solution optimale par rapport au système de score choisi. Les premiers algorithmes d'alignement [145, 194, 71] ont été proposés il y a plusieurs décennies, mais ils continuent à jouer un rôle de première importance même aujourd'hui.

Cependant, avec l'accumulation de données produites par les technologies de séquençage de première génération (séquençage de Sanger [174]), la simple programmation dynamique commençait à s'avérer trop lente. Ainsi, pour répondre à ces limitations, de nouvelles approches heuristiques ont été proposées en complément de leurs prédécesseurs. L'hypothèse à la base de ces heuristiques est le fait que, si deux séquences sont suffisamment similaires, alors elles doivent forcément partager au moins une région bien conservée. Les outils de la famille de BLAST [151, 206, 4] cherchent ces régions en construisant des indices de sous-chaînes chevauchées de longueur constante k appelées "k-mers". Les parties potentiellement homologues sont alors individuées grâce aux k-mers en commun (graines). Le résultat final est obtenu avec de la programmation dynamique classique limitée aux alentours des blocs trouvés par les indices, ce qui permet une majeure efficacité.

L'introduction de séquenceurs de deuxième génération en 2006 avec un plus haut

débit mit à nouveau en difficulté les algorithmes présents à l'époque. Une nouvelle famille d'algorithmes d'alignement prit alors place. À la place de construire des indices de k-mers, outils comme BWA [112] et similaires [105, 117] emploient de méthodes d'indexation compressée comme les suffix trees/arrays [200, 130], la BWT [28] ou bien des FM-Index [61]. La nature succincte de ces indices permet de réduire considérablement l'espace de stockage. Un autre avantage de ce type de structures est que la recherche de régions conservées n'est plus limitée au graines de longueur k alors qu'elle se peut étendre sans limite. Les indices compressés restent les meilleures méthodes d'alignement de reads affectées par petites erreurs, ce qui est le cas pour la deuxième génération de séquenceurs.

En revanche, les reads produites par la troisième génération de séquenceurs sont plus longues mais aussi plus imprécises. La majeure longueur est avantageuse pour la détection de mutations à large échelle (comme de longues délétions, insertions ou bougement de bases) mais l'augmentation de la probabilité d'erreur, avec conséquente diminution de régions conservées, rend l'indexation compressée presque inutile. Les nouvelles techniques heuristiques pour la troisième génération reprennent alors le concept d'indexation de k-mers en le combinant avec de l'échantillonnage, afin de diminuer les ressources computationnelles requises pour la recherche de graines. Plusieurs méthodes d'echantillionnage existent, parmi elles les bien connues "minimizers" [178, 167, 114, 91, 89] (mais aussi les "syncmers" [52]).

L'alignement n'est guère la seule méthode pour estimer la similarité entre séquences. En effet, une autre possibilité consiste en obtenir une indication de ressemblance depuis la composition des séquences sans chercher d'aligner leurs bases. Cette idée est le fondement de techniques dites "alignment-free" (lit. sans alignement) où les comparaisons sont basées sur des indicateurs de similarité alternatifs, comme, par exemple, la longueur de sous-chaîne partagées ("Average Common Subsequence" [201]) ou le degré de compressibilité d'une chaîne par rapport à l'autre [203, 115]. Parmi les dizaines de méthodes sans alignement pour l'estimation de la similarité, les techniques qui utilisent les k-mer ont eu le plus de succès grâce à leur versatilité [15, 92, 193, 108, 60, 65, 66, 207, 5, 180, 196, 49, 100, 23, 150]. Dans ce cadre, les séquences sont représentées comme ensembles (pesés ou pas) de k-mers. Malgré la grande disponibilité de mesures de distance qui peuvent être calculées sur des ensembles, la plupart d'elles n'ont pas de bases biologiques [224]. Seulement récemment une mesure de distance biologiquement vraisemblable a été introduite par [60], où la fréquence de mutation est estimée à partir de la similarité de Jaccard [142].

Il y a quelques années, même les algorithmes sans alignement ont commencé à montrer leurs limitations à cause de la croissance exponentielle du rendement des moderns séquenceurs. Encore une fois, il a donc fallu adapter de nouvelles techniques à l'analyse des séquences. Le 2016 a signé le début en bioinformatique des algorithmes de "sketching" (esquissage) avec l'introduction de l'outil Mash [147]. L'esquissage consiste à transformer les séquences en représentations réduites qui peuvent être ensuite comparées directement afin de donner une estimation de la grandeur pour laquelle elles ont été conçues. À différence des algorithmes sans alignement classiques, les esquisses contiennent beaucoup moins d'information par rapport aux séquences originelles mais leurs estimations ont une qualité comparable à celle de méthodes plus complètes. En particulier, Mash est une implantation de la technique MinHash [26] visée à la similarité de Jaccard. Plusieurs implantations de MinHash ont suivi Mash [27, 221, 216, 7, 56] ainsi que de nouvelles applications [175, 103, 147, 99]. Un des résultats de cette thèse (**Partie IV**) est l'introduction d'une méthode d'esquissage alternative pour la similarité de Jaccard, dans le cadre de séquences très similaires. Nous utilisons des "Invertible Bloom Lookup Tables" [55, 69] pour atteindre l'objectif. Par ailleurs, nous montrons d'autres applications des IBLTs liées à leur capacité de récupérer exactement la différence symétrique de deux ensembles. En plus, nous démontrons expérimentalement comment l'échantillonnage de syncmers

donne des estimations de la similarité de Jaccard qui ne sont pas biaisées, à différence de l'échantillonnage basé sur les minimizers.

Un autre objectif de notre travail est l'exploration de représentations efficaces pour les fréquences de k-mers. Les fréquences des k-mers sont utiles dans plusieurs analyses [60, 65, 66, 108, 207, 5, 180, 196, 49, 100, 23, 150] mais leurs techniques d'esquissages n'ont pas bénéficié du même succès de MinHash ou d'autres méthodes réservées aux k-mers. À ce propos, nous proposons dans la **Partie II** une technique d'esquissage adaptée aux compteurs de k-mer. Notre algorithme d'esquissage, inspiré aux bien connues CountSketch [33] et Count-Min sketch [42], résulte plus adapté aux distributions de k-mers typique pour des données biologiques, afin de réduire les erreurs qui affectent les estimations ponctuelles.

En revanche, dans la **Partie III** nous proposons plusieurs façons de compresser *exactement* les fréquences de k-mers. Nous apportons des améliorations à la seule implantation de "Compressed Static Functions" (CSFs) [67] disponible au moment de l'écriture de cette thèse afin de créer des *Bloom-enhanced Compressed Static Functions* (BCSFs). La combinaison de BCSFs et le groupage de k-mers avec les minimizers donne de représentations succinctes dont espace dépend de l'entropie de l'ensemble des compteurs. À notre connaissance, nos algorithmes ont été les premiers à produire des structures de données plus petites de l'entropie des pèses y stockés dedans.

Esquissage de compteurs

Compter les nombre d'occurrences de k-mers est une des opérations primordiales en bioinformatique. Cette information est utile pour plusieurs tâches, comme le garnissage de reads [125] ou la détection de variants (sans alignement) [163, 100]. Des nombreaux outils pour le comptage ont été proposés dans la littérature, comme Jellyfish [135], DSK [165] ou bien KMC [101]. Chaque compteur a ses avantages et désavantages afin de balancer vitesse et mémoire. Par exemple, Jellyfish favorise la vitesse alors que DSK ou KMC tendent à être moins exigeants en termes de mémoire (RAM). Les sorties de tous ces outils sont des tables où chaque k-mer est associé à sa fréquence. En général, ce genre de structures associatives peuvent avoir besoin de beaucoup d'espace, surtout s'elles stockent de longues k-mers. Pour donner une idée, il suffit de savoir que le comptage d'un génome humain par KMC donne en sortie une table d'environ 28 Go, pour $k = 32$.

Une technique utilisée dans la pratique est de sauvegarder seulement les compteurs et ignorer les k-mers. Cette idée est supportée par le fait que l'ensemble de k-mers en plusieurs applications est fixe et il ne peut pas changer. Exemples sont la lecture de k-mers depuis des reads partiellement assemblées ou depuis des représentations succinctes d'ensembles de k-mers (comme de graphes de de Bruijn colorés [82] ou de "spectrum-preserving string sets" [29, 162]). L'idée de stocker seulement les compteurs sans leurs k-mers est supportée aussi par le fait que, en général, le nombre de valeurs de comptage est relativement petit par rapport au nombre de k-mers totales.

En effet, pour grandes valeurs de k, les spectres de k-mers obtenus en listant leur nombre en fonction de leurs fréquences suivent une distribution dite "power-law" [44, 36]. Ce sont des distributions très asymétriques où un grand nombre de k-mers a de très petites fréquences, alors que seulement une minorité est très répétitive. Pour k croissant, on s'attend à une diminution du nombre de fréquences distinctes, avec pour cas limite la situation où toutes les k-mers sont uniques.

Utiliser des compteurs de taille fixe peut alors gaspiller beaucoup d'espace car seulement une petite fraction de valeurs a réellement besoin d'un grand nombre de bits. Dans les deux sections qui suivent, nous proposons deux techniques alternatives pour représenter les comptages de manière efficace.

Esquisse Set-Min

La Partie II de cette thèse introduit l'esquisse Set-Min, une variante de la bien connue esquisse Count-Min [42] (elle-même variante de CountSketch [33]). Pour rappel, l'esquisse Count-Min est une matrice A de compteurs de B colonnes et R lignes chacune avec sa propre fonction d'haçage $h_i(\cdot)$. À l'arrivée d'un nouveau pair (clé, valeur) (p, ℓ), l'insertion consiste à exécuter $A(i, h_i(p)) = A(i, h_i(p)) + \ell$ pour chaque ligne i. Les mises à jour sont donc additives ce qui donne des estimations ($\hat{\boldsymbol{a}}(p) = \min_i\{h_i(p)\}$) toujours biaisées par des quantités positives ou nulles. Si Count-Min est dimensionnée avec $R = \lceil \ln(\frac{1}{\delta}) \rceil$ et $B = \lceil \frac{e}{\varepsilon} \rceil$ pour n'importe quelle valeur de $0 < \varepsilon \leq 1$ et $0 < \delta \leq 1$, alors ses estimations sont affectées par une erreur au plus $\varepsilon \|\boldsymbol{a}\|_1$ avec une probabilité $1 - \delta$, où $\|\boldsymbol{a}\|_1$ est la norme L_1 de $\boldsymbol{a}$. Des erreurs d'une magnitude si potentiellement élevée (une fraction du nombre total de k-mers) sur chaque estimation peuvent s'avérer incompatibles avec la bioinformatique.

Cependant, nous démontrons comment Set-Min permet de borner l'erreur *cumulative* totale, calculée comme la somme des erreurs de toutes les k-mers. Il suffit tout simplement de remplacer les compteurs d'une matrice Count-Min avec des ensembles de fréquences. En effet, grâce à la représentation par ensembles, il devient possible de réutiliser un même compteur pour plusieurs k-mers qui tombent dans la même cellule. Puisque les multiplicités des k-mers doivent maintenant être connues avant insertion, Set-Min perd la capacité d'esquisser de séries de mises à jour partielles, comme c'était le cas pour Count-Min. Nous proposons un algorithme heuristique pour le dimensionnement de nos esquisses.

Nos résultats montrent que Set-Min est plus précise d'une esquisse Count-Min de dimensions comparables. Ce résultat est possible grâce à l'exploitation de la distribution asymétrique typique pour les spectres de k-mers. En plus, pour k suffisamment grand et génomes assemblés, Set-Min peut être un ordre de magnitude plus petite par rapport à de solutions basée sur Minimal Perfect Hashing.

Représentation efficace pour les tables de comptage des k-mers

Dans certains cas, il est préférable de n'avoir aucune erreur d'estimation. Par exemple, trouver exactement l'ensemble de k-mers uniques avec une esquisse Set-Min n'est pas, en général, possible, car, même si les estimations sont beaucoup plus précises de celles de Count-Min, elles sont toujours affectées par des erreurs. Avec le grand nombre d'applications qui font usage de fréquences [193, 163, 100, 132, 95, 96, 143], il est alors indispensable de pouvoir disposer d'une structure exacte.

À part l'exactitude, le but de cette partie de la thèse reste inchangé par rapport à la précédente: concevoir une structure qui permet de récupérer de fréquences de k-mers. Seulement les compteurs doivent être traités, avec les k-mers elles mêmes supposées connues. La solution classique à ce problème est d'utiliser des "Minimal Perfect Hash Functions" (MPHFs) [144, 214, 57, 122, 157]. Les MPHFs assignent bijectivement chaque objet d'un ensemble S de dimension $|S|$ une position dans l'intervalle $[0, |S| - 1]$. En utilisant l'ordre imposé par une MPHF pour indexer un tableau, il devient alors possible d'assigner n'importe quelle information supplémentaire aux k-mers. Cependant, cette solution n'est pas optimale: les MPHFs ont besoin de mémoire supplémentaire aux compteurs stockés dans les tableaux.

À ce propos, très récemment, [67] a proposé une implantation pratique de *Compressed Static Functions* (CSFs). La caractéristique qui distingue les CSFs de MPHFs est que le les

premières sont de structures monolithiques dont l'espace dépend de l'entropie des valeurs qui contiennent. Les CSFs n'ont pas ainsi besoin d'un tableau extérieur et, théoriquement, elles peuvent remplacer les MPHFs dans toutes les applications. Toutefois, la solution de [67] comporte une limitation technique qui limite son efficacité à 1 bit/clé. Cette contrainte se révèle particulièrement désavantageuse pour les k-mers, à cause de leur entropie qui est souvent < 1 en raison de leur distribution asymétrique. Notre premier résultat est alors une amélioration de [67] pour distributions d'entropies inférieures à 1.

Pour ce faire, nous utilisons de filtres de Bloom pour éviter de sauvegarder explicitement la plupart de copies du compteur le plus commun, en nous focalisant sur les valeurs les plus rares. La structure résultante, que nous appelons "Bloom-enhanced Compressed Static Function" (BCSF) est ensuite employée comme élément de base pour la suite de nos travaux. En effet, nous démontrons également comment utiliser les minimizers [178, 167] pour regrouper de mêmes compteurs ensemble afin de gagner encore plus d'espace. Le fait que les k-mers voisines tendent à partager leur fréquence a été observée par [132]. Ce constat est combiné avec des BCSFs en deux techniques que nous appelons AMB et FIL.

BCSFs, AMB et FIL sont comparés entre eux et avec BBHash [122], une implantation de MPHF bien connue en bioinformatique. Avec un choix adéquat de k, AMB et FIL produisent de structures plus petites de l'entropie des fréquences qui contiennent. À notre connaissance, nos algorithmes ont été les premiers à atteindre cet objectif.

Computation efficace de différences entre ensembles de k-mers avec les "Invertible Bloom Lookup Tables"

Nous nous éloignons des tables de comptage dans la **Partie IV** de cette thèse. À leur place nous nous concentrons sur le problème de comment esquisser efficacement la similarité de Jaccard pour des séquences (ensembles de k-mers) très similaires. Pour mieux comprendre le problème il faut savoir que la similarité de Jaccard entre deux ensembles A et B est définie comme le nombre d'éléments partagés divisé par le nombre total d'éléments distincts. En termes mathématiques: $J = |A \cap B|/|A \cup B|$.

MinHash est la technique d'esquissage de la similarité de Jaccard [26]. Dans ce cadre, les ensembles sont réduits en esquisses de taille fixe s. Si s est trop petite et A et B ont une similarité élevée, $|A \cap B|$ pourrait contenir pas suffisamment de k-mers pour que MinHash puisse les échantillonner. La similarité de Jaccard résulterait alors égale à 1 même pour des ensembles différents. Pour cette raison, nous remplaçons MinHash avec une technique d'esquissage appelée "Invertible Bloom Lookup Table" (IBLT) [69, 55], initialement conçue pour le problème de réconciliation d'ensembles. Les IBLTs permettent de récupérer la différence exacte entre deux ensembles, ce qui les différencie des esquisses MinHash. Étant donné que la similarité de Jaccard peut être réécrite en termes du nombre d'éléments dans chaque différence ($A \setminus B$ et $B \setminus A$), il devient alors possible d'utiliser des IBLTs à la place de MinHash quand la similarité entre séquence est attendue être élevée. La procédure de récupération est basée sur le pelage de graphes [50].

L'implantation originelle d'IBLTs décrite en [69] est un tableau de cellules, chacune composée d'un compteur C, un champ H et un champ dédié à la somme de clés. Les insertions fonctionnent de manière similaire aux filtres de Bloom classiques, avec r fonctions de hachage qui assignent r indices à chaque clé. Le compteur C compte le nombre de clés contenues dans les cellules, tandis que le champ P stocke leur somme binaire obtenue avec l'opérateur XOR. Les champs H servent à vérifier s'il y a de collisions, ce qui pourrait empêcher la réussite de l'opération de pelage. Ici, nous montrons comment ce dernier contrôle peut être aussi réalisé sans champ H, en regardant seulement les positions données par les r fonctions d'haçage. En plus, inspiré par [159], et à différence de

toute méthode antérieure faisant usage d'IBLTs, nous calculons la différence entre deux IBLTs directement, sans avoir besoin d'un de deux ensembles de départ. Pour cette raison, notre méthodologie devient plus proche à MinHash, où les esquisses sont les seules données utilisées durant les comparaisons. Nous appelons notre implantation "`km-peeler`". Ensuite, nous combinons l'echantillionnage par syncmers avec notre implantation d'IBLT (c-à-d: nous insérons dans les IBLTs des ensembles de syncmers à la place d'ensembles de k-mers). Les syncmers n'introduisent pas de biais dans les estimations de similarité de Jaccard, à différence des minimizers. Avec des esquisses de taille comparable, notre technique résulte en des estimations plus précises de celles produites par MinHash.

Même si les IBLTs peuvent porter à des esquisses efficaces pour la similarité de Jaccard, elles ont été conçues pour récupérer efficacement les différences de façon *exacte*. Afin de mieux démontrer la pleine puissance d'IBLTs, nous les appliquons au problème d'approximation de différences d'ensembles de k-mers. À la place d'échantillonner les syncmers comme contre-mesure au fait que chaque mutation peut comporter jusqu'à k nouvelles k-mers dans la différence, ici nous les utilisons pour regrouper ensemble plusieurs k-mers, afin de calculer leurs différences avec des IBLTs. Les groupements récupérés depuis les différences entre IBLTs sont après décomposés dans les k-mers qui les constituent. L'approximation dérive du fait que les ensembles de k-mers finaux sont en général de super-ensembles des vraies différences.

Introduction

Similarity estimation is a fundamental task in bioinformatics. Classical approaches to this problem consist in aligning sequences and estimating similarity as a function of mutations. Multiple sequence alignment is used to precisely compute similarities of multiple sequences, but finding the best solution is a hard problem. Instead, most tools divide the multiple sequence alignment into multiple pairwise comparisons, which are later combined to give the final result [123, 80, 146, 51]. Similarity between two strings can then be estimated from the alignments listing the elemental operations (substitutions, insertions and deletions) needed to transform one sequence into the other. The dynamic programming algorithms computing alignments [145, 194, 71] are the backbone of many tools still in use nowadays [112, 114]. Such algorithms are very precise at the price of being computationally demanding, and they are not very well suited to deal with long sequences. Further improvements to the topic of (pairwise) sequence comparison came with the tool BLAST in which seeding heuristics were incorporated to classical dynamic programming algorithms as a mean to speed-up homology detection. Seeding refers to the idea of indexing one of the sequences using so-called "seeds", i.e. fixed-length overlapping substrings of the sequence (in the literature also known as k-mers or q-mers). Storing such seeds inside hash tables along with their position in the sequence makes it possible to quickly discover potential sites of similarity. Full alignments/edit-distance computations can then be limited to such regions with a considerable advantage in terms of time. However, such efficiency improvements were quickly outpaced around 2006 with the introduction of Next-Generation Sequencing technologies characterized by much higher throughputs. In order to deal with the increasing amounts of data generated each year, the dawn of the big-data era of bioinformatics spurred interest in finding more memory-efficient replacements of hash tables. This led to the development of new aligners [112, 117, 105] based on compressed representations supporting a wide range of operations (substring search, count). Examples of such structures are suffix trees/arrays [204, 200], BWT [28] and FM-Index [61].

Further disruptions of the bioinformatics landscape came with the introduction of Long-Read sequencers (also known as Third-Generation sequencers). Note that Long-Read sequencers are not yet meant as replacements for the previous generation (which was retroactively renamed Second-Generation). Rather, they complement short, almost error-free reads with longer and noisier ones. Long reads are advantageous in applications where their increased lengths allow detection of structural variations. On the other hand, higher error rates lead to slower searches in succinct indexes, since each mismatched character has to be dealt with during exact searches [112].To solve this issue, the Third-Generation sequencing era has seen the revival of k-mer-based indexes [113, 114]. Unlike full k-mer indexes, new hash table-based dictionaries keep space usage under control by sampling k-mers. Many sampling techniques designed for k-mers have been proposed in recent years, e.g. minimizers [178, 167] or syncmers [52]. Compared to naive sampling, these techniques have the advantage of guaranteeing a maximum distance on two successive sampled k-mers, which greatly improves the sensitivity of homology detection.

Alignments are not the only method to compute sequence similarities: "alignment-

free" algorithms compute sequence similarity by looking at their composition, without the need of performing explicit alignments. The most common type of alignment-free algorithms represent sequences as sets or multi-sets of k-mers. Alignment-free algorithms are usually faster than their alignment-based counterparts given that fewer operations are involved in the computation of similarity (no alignment is produced) and fast set representations exist (i.e. hash tables). However, with the ongoing data deluge in the bioinformatics field, the high memory requirements of alignment-free algorithms are quickly becoming a limiting factor, despite their obvious speed advantage. Improving the current infrastructure with additional storage capabilities is an unrealistic option since sequencing has long outpaced Moore's law [93]. While sampling k-mers remains one of the most straight-forward option to reduce space, sketching techniques have started to play a major role in bioinformatics starting with the introduction of Mash software [148] in 2016. A combination of an alignment-free estimator of mutation rate [60] and the MinHash sketching technique of [26], Mash was the first method to allow for the comparison of all the 54118 bacterial genomes stored in the NCBI RefSeq at the time [148]. Since then, a myriad of sketching techniques have followed (for a non-exhaustive list see Part I).

MinHash-based (and the somehow correlated HyperMinHash [216]) sketches do not take into account k-mer multiplicities, with sequences treated as *simple* sets of k-mers. While doubts have been raised recently about the utility of counts in producing high-quality phylogenies [120], k-mer multiplicities remain useful for a wide range of other applications [132, 100, 49]. Hence, k-mers counting represents another pillar of many bioinformatics pipelines and, as such, has been the subject of an intense research effort over the past years, resulting in multiple counting tools, such as Jellyfish [135], DSK [165] and KMC [101]. These methods primarily focus on counting speed, with memory and queries often treated as second class citizens. Such a trade-off is understandable if counting is expected to be the primary bottleneck and k-mers are expected to be accessed only once. This is the case for exploratory analyses where counting tables provide useful insights about the composition of genomic datasets. However, the lack of support for fast single-point queries and large output sizes make most counting tools unsuitable for random comparisons of k-mers extended with their frequencies. Practical k-mer set representations extending k-mers with their weights have been proposed [133, 155], but these works focus on the idea of separating genomic and count information into independent data structures for maximal flexibility [132, 98].

In particular, an interesting question is whether taking into account the distribution of counts can lead to smaller overall space. Works like [36] or [44] suggest that k-mer spectra are heavy-tail distributed for k large enough. In other words, the majority of k-mers are mostly unique or appear few times, with a very small fraction repeating many times. For this reason, fixed-width counters potentially waste a lot of space since most of the significant bits are truly needed by only a handful of values. While classic sketching techniques for counts, like CountSketch or Count-Min [43, 33] work best on such kind of distributions, their estimations are affected by errors so large that they are unsuitable for bioinformatics applications. The problem of developing a sketching technique for k-mer count tables, taking advantage of the count distribution while, at the same time, achieving small errors is the main topic of Part II.

Sketching is not the only option for building maps between k-mers and their count values. Another approach involves the use of Minimal Perfect Hash Functions (MPHFs) [122] with the actual counts stored in an external array. Each query is thus split into two separate steps: using the MPHF to retrieve an index and accessing the array at the given position. While the external array can be compressed to save space, this is seldom done in practice in order to favor speed over memory. Furthermore, MPHFs take non-negligible space themselves, with a theoretical minimum of 1.44 bits/key and implementations taking

up to ≈ 3 bits/key in practice [122, 157]. An alternative method to MPHFs is provided by Compressed Static Functions (CSFs). Counts are bundled together with the mapping information avoiding the external array altogether. Most importantly, CSF size depends on the *empirical zero-order entropy* of the stored count vector. Since entropy of genomic k-mer counts is expected to be low, thanks to its power-law distribution, CSFs have the potential of being more memory efficient than MPHFs. In Part III we show how we adapted the only practical implementation of CSFs available today [67] to succinctly store k-mer count tables. The proposed methods in Part III can be viewed as more involved alternatives to our sketch-based solution of Part II.

Finally, in Part IV we shift our attention from efficient k-mer count table representations to the efficient computation of k-mer set differences, when sequences are highly similar. As noted earlier, MinHash has become the default sketching technique for comparing unweighted sets of k-mers. Nevertheless, when sets are of high similarity, sketch size s needs to be chosen large enough for MinHash to be able to sample differences. To this end, Part IV presents a method able to estimate Jaccard similarity for highly similar sets in small space. Our advancements are based on the combination of syncmers [52] and *Invertible Bloom Look-up Tables* (IBLTs), a sketching technique first developed for set reconciliation in distributed settings [55, 69].

To sum up, the contribution of this work is the development of new methods for the problems of: i) representing count data from k-mer count tables efficiently, while also providing support for single count queries, and ii) computing (symmetric) differences between very similar sets in small space. Solutions to the first point can be viewed as natural extensions to existing k-mer set representations, allowing to extend k-mers with their frequencies. We present two complementary approaches to this problem: the first is a new weighted sketch able to achieve competitive space with small query errors. Our second approach is based on a combination of minimizers and CSFs. We provide three alternative implementations with different trade-offs. All our methods take advantage of the characteristic power-law distribution of k-mers, which makes them more efficient than fixed-width counters, while retaining the ability of retrieving single k-mer counts.

Solving our second challenge gives us a memory-efficient sketching technique based on IBLTs to estimate Jaccard similarity, able to beat MinHash in both memory and precision on highly similar sequences (such as viral or bacterial strains).

Structure of this thesis

Part I starts with the introduction of key biological concepts followed by an overview of the methods that constitute the basis upon which we built our own results.
In particular, we open **Chapter 1** with an overview of the main sequencing techniques used to transform the genetic information contained in DNA into digital-friendly representations. This introductory chapter concludes with a remainder about assembly, for which full sequences can only be obtained by reconstruction from smaller fragments.
Chapter 3 introduces the concept of pairwise sequence comparisons and classic approaches to this problem.
Chapter 4 follows with an overview of alignment-free algorithms and related challenges (Section 4.2).
We present the idea of sketching in **Chapter 5** together with the sketching techniques used in this thesis in Section 5.1.
We continue with **Chapter 6** in which we present the problem of succinctly represent exact k-mer counts and relevant solutions. Note that current implementations of exact k-mer count tables provide the inputs of our methods in Part II and III (e.g. we build our sketches from KMC databases).

Part II presents Set-Min sketch, our sketching technique for highly skewed power-law distributions.
First, **Chapter 7** introduces the problem and related solutions: CountSketch and Count-Min sketch with and without conservative updates in Section 5.1.1, Section 5.1.2 and Section 5.1.3, respectively. A summary of the contributions of this part is given in Section 7.2.
Chapter 8 is about Set-Min sketch. We start by listing the main insights behind our method in Section 8.1. After the introduction of power-law distributions and the idea of using collisions between equal counters to reduce space, we follow with the internal structure of Set-Min sketch in Section 8.2. We show how tighter error bounds can be achieved by replacing counters with sets in a Count-Min matrix (Section 8.2.1.1). A heuristic algorithm for dimensioning Set-Min sketches is given in 8.2.2. Section 8.3 is about a simplified version of Set-Min sketch whose main purpose is to demonstrate that full set representations are indeed necessary to achieve the desired error bounds. We conclude by presenting an experimental comparison of our methods against well-known solutions based on MPHFs and k-mer counting tools.
A general discussion about Set-Min together with some final consideration make up **Chapter 9**.

Part III is about *locom*, our exact framework for efficient k-mer count representation. Multiple algorithms are presented, with CSFs as their maximum common divisor. Locom's data structures are more involved and computationally expensive to build than simple Set-Min sketches. The main advantage of our CSFs-based solutions over sketching are the more succinct and exact count representations they provide. In some instances, maps produced by locom are even smaller than the empirical zero-order entropy of the uncompressed counts. As before, this part is divided into three chapters.
Chapter 10 introduces the problem, previous solutions and a summary of our contributions (Sections 10.1, 10.2 and 10.3).
Chapter 11 starts once again with the insights behind our methods. The expected high correlation of neighboring k-mer counts allows hashing similar values to the same bucket by using minimizers (Section 11.1.1 and 11.1.2). We combine Bloom Filters (Section 5.1.5) with a third-party implementation of CSFs (Section 6.2.2) to produce *Bloom-Enhanced Compressed Static Functions* (BCSFs) in Section 11.2.1. Adding a LSH-based bucketing to BCSFs gives rise to two efficient count representations: AMB (Section 11.2.3) and FIL (Section 11.2.4). Both algorithms can work in the exact or approximate case demonstrating the flexibility of our framework. Comparisons and benchmarks are presented in Section 11.3.
Chapter 12 discuss the limitations and possible further applications of our methods.

Part IV deviates from the previous subject by addressing the problems of i) estimating Jaccard similarity of very similar sequences and ii) computing a superset of k-mer sets differences in small space.
Chapter 13 explains the interest of our approach (Section 13.1) and the summary of our contributions (Section 13.2).
Chapter 14 starts by introducing the relevant building blocks of our method in Section 14.1. We achieve the goal of efficiently estimating Jaccard similarities as a by-product of set reconciliation techniques. More in detail, we use IBLTs (Section 5.1.6) to efficiently estimate set differences. In order to do so, in Section 14.2.1 we extend IBLT difference to work with sketches, without the need to keep one of the original sets. To dampen the effects of mutations on the number of k-mers ending up in the symmetric difference (one single mutation can lead to the introduction of up to k different k-mers), we sample

k-mers before IBLT construction (Section 14.2.3). In Section 14.3.1, we experimentally show that, unlike minimizers, syncmers are unbiased estimators of Jaccard similarity suggesting a practical alternative to the shortcomings pointed out in [12]. Further, we apply the full potential of IBLTs by computing exact supersets of the whole k-mer sets in Section 14.2.5. Section 14.3 reports our results.
Chapter 15 wraps everything up by discussing and summarizing our findings and presents possible future developments.

Finally, in **Chapter 16** we close the thesis with conclusions and perspectives.

Part I

State of the art

Chapter 1

DNA Sequencing and assembly

Reading the succession of bases in DNA molecules is the fundamental task providing raw material to all bioinformatics pipelines. Despite 50 years of technological advancements, reading whole genomes remains a hard process, with most sequencing technologies unable to process very long biomolecules. Therefore, input DNA sequences have first to be split into multiple, shorter fragments. Furthermore, since sequencers are not perfect and reading DNA molecules is a noisy process subject to errors, sequences are first copied multiple times before fragmentation. Sequencing output is thus a collection of multiple, redundant strings, usually called "reads". The original sequence is then reconstructed from its set of reads. This process is called (de novo) "assembly" if no prior knowledge about the original sequence is known. If, on the other hand, an already assembled sequence similar to the sequenced one is available, it can be used as a guide by "aligning" reads to it. In both cases, redundancy is used for error correction. In this chapter, we provide an overview of the technologies and methods used in these two steps.

1.1 Classic sequencing approaches

The oldest surviving sequencing method still in use today is Sanger sequencing [174] which is heavily inspired by the cell replication process. Multiple imperfect copies of a DNA sequence are obtained by introducing some nucleotides with *dideoxyribose* sugars (instead of the normal deoxyribose normally found in DNA) during Polymerase Chain Reaction (PCR) [141, 172] prompting DNA polymerases to stop extension. Sorting the incomplete copies by their molecular weight through gel electrophoresis and reading their terminating character gives the original starting sequence. Sanger sequencing completely replaced a previous method: chemical sequencing (due to Maxam [136]) which relied on radioactive reagents making it less practical. Due to its high accuracy, Sanger sequencing is still actively being used in project targeting relatively short genomes or for validation of results produced by newer technologies.

1.2 Second-Generation sequencing

Second-Generation sequencing technologies (also known as Next-Generation sequencing (NGS)) started to appear around 2006. Reading fragments is done in a massively parallel way allowing for much higher throughput than the previous generation. The main advantage of Second-Generation sequencing is that it is considerably cheaper than Sanger's. For example, completion of the Human Genome Project using NGS would cost $1K against the $2.7 billion required by Sanger sequencing. Despite shorter reads (< 400bps) and slightly higher error rates compared to Sanger sequencing, NGS sequencers remain the

most convenient solution for a wide range of applications that require good trade-offs between accuracy and cost-effectiveness. Common Next-Generation technologies are listed below [25].

Illumina. Illumina sequencers were first introduced in 2006 [124], and quickly became the incumbent technology dominating its generation [70]. Read lengths range from 100 to 300 bp with optional support for paired-end output. Error rates are very small ($< 0.01\%$ of bases are wrongly called [176, 177]), with substitutions being the most common type of errors, especially at the end of reads.

SOLiD. Also introduced in 2006 [124] its distinguishing feature is how sequences are viewed in input. Instead of reading each nucleotide separately, SOLiD sequencers detect transitions between bases. Therefore, reads are returned as sequences of colors instead of letters [152]. This encoding allows distinguishing different events (sequencing errors, substitutions and indels) by looking at colors alone [24]. Error rates are also very small (< 0.001) but special analysis software is needed to work with the uncommon encoding. Average (paired-end) read length is 100 bp.

Ion Torrent. The most recent method of the list (2010 [124]), with lengths of about 400bp. The most common type of errors are indels which appear at a rate of about 0.03 with substitutions an order of magnitude less likely [22].

Roche/454. Sequencers based on the concept of "pyrosequencing" [168] were first introduced in 2005 [124]. The company sponsoring them shut down in 2013 [83]. Read length were around 1000 bp with errors rates of about 0.01, mostly indels.

1.3 Third generation sequencing

Short reads produced by NGS are often an obstacle in the detection of large rearrangements in parts of the genomes. Third-Generation sequencers complement short, highly accurate reads with longer ones (> 10000bp) and, as such, they are commonly referred to as Long-Read sequencing. The downside is that bases are affected by higher errors. Long reads are now best suited to the detection of large structural variations (long deletion/insertions or duplications). However, further improvements in read accuracy are expected to make them completely replace the previous generation.

There are currently two major competing long-read technologies:

Pacific Bioscience. Also known as PacBio (`www.pacb.com`), typical read lengths are 20000 bp with errors < 0.1 [202]. The underlying technology called Single-molecule real-time (SMRT) sequencing couples a single polymerase protein with a zero-mode waveguide unit [111]. Four fluorescent dyes are attached to nucleotides (one different dye for each of the 4 bases). When a new nucleotide is added by the polymerase, the fluorescent tag is removed from its nucleotide and diffuses through the zero-mode wavelength guide. The optical signal is captured by the sequencer for generating the sequence. High throughputs are the result of multiple polymerases working in parallel.

Oxford Nanopore. Sequencers based on nanopores use a different idea than SMRT. As the name suggests, sequences are passed though molecular electro-active "pores" of a membrane. Different bases produce different perturbations in the pore's current. This signal is used to read the sequence. Typical error rates range between 5% [53] to 10% [46]. Read lengths surpassing 1 Mbp have been reported with more common values in

the order of 100 Kbp. The most remarkable feature of Nanopore sequencers is their size with the entry level sequencers MinION and Flongle slightly larger than a USB dongle (https://nanoporetech.com/). Other aspects that make Oxford Nanopore sequencers interesting are their ReadUntil mode of operation [104, 1, 9] and their rapidly evolving performances.

Chapter 2

Assembly

In the previous section, we saw how sequencers produce in output datasets of reads by reading fragmented sequences. In order to obtain the original sequence of bases, reads need to be reassembled back into one contiguous representation.

The reconstruction process greatly depends on the error rate and type of the reads (short or long), and if an already assembled reference genome is available. With no prior reference genome, assembly must be performed "de novo". Almost all algorithms in this category use some sort of graph representation. The basic idea is to represent reads as nodes and to add edges between likely successive reads. The original DNA sequence thus appears as one of the paths in the graph.

Two main types of graphs exist [166]:

- de Bruijn graphs [217, 192, 118, 153, 68, 38, 34, 8, 53] are primarily used for short, low-error reads. Each read is decomposed into small overlapping fixed-sized substrings called k-mers which are used as nodes. Edges between two nodes (k-mers) represent overlaps of $k-1$ bases.

- Overlap graphs [170, 102, 113] link reads together directly, based on their prefix/suffix overlaps.

If a previous assembly of a sequence already exists, then the problem becomes much simpler as the order of most reads can be inferred from the guide. This particular instance of assembly takes the more appropriate name of "alignment".

Several families of aligners exist in practice:

- Dynamic programming [194, 145, 71] is the exact, baseline method which is generally too computationally inefficient for datasets larger than some thousand bases.

- Seed and extend [206, 151, 4, 114] makes use of k-mer indexes as a heuristic to quickly find highly similar regions to extend. Reads are placed along the sequences by finding the most similar ordered sequence of seeds that match in both. Dynamic programming is used locally to produce the final base-to-base alignment. Very fast in practice since it uses hash tables for storing seed. Recent advancements aimed at reducing index size though sampling allowed reviving seed and extend for long noisy reads [113].

- Sequence indexing by suffix trees/arrays [204, 200, 130], BWT [28] or FM-Indexes [61] such as [48, 47, 81, 106, 105, 112, 117] are best suited for short exact reads. Indexes provide a way to find all sequence positions that highly resemble a read. As before, an exact dynamic programming procedure is needed to finalize local alignments.

All methods described so far use some definition of sequence similarity to define edges or to place reads at the most accurate position for later refinement. However, similarity is not limited to assembly but is used in a plethora of other methods such as mappers [87, 88, 91], phylogenetic reconstruction or taxonomic placement to deduce organism hierarchies [30, 207], and others.

Chapter 3

Pairwise sequence similarity

Sequence similarity has been an instrumental bioinformatics tool for tracing the evolutionary history of sequences since it is a powerful way to prune for function homology or to reconstruct phylogenetic histories. New species arise by mutation of existing ones though an evolutionary process made possible by incremental modifications of DNA. Genes needed for survival, for example, are expected to change very slowly over time given that any misplaced mutation is fatal. We limit ourselves to *pairwise* sequence comparisons, given that most of the techniques mentioned in Section 4 and Section 5 only support this mode of operation. Moreover, various methods build multiple sequence alignments (MSAs) by combining multiple pairwise comparisons [123, 80, 146, 51] highlighting the importance of this reduction.

3.1 Alignment-based methods

Alignment-based methods are deeply rooted into the problem of edit distance computation. Given two sequences S_1 and S_2, under this framework similarity is expressed in terms of the number of basic operations needed to transform S_1 into S_2. Here with "basic operations" we refer to unary substitutions and indels. Scores can be assigned to each type of operations in order to weight them differently. For example, uniform schemes do not take into account that large indels are usually the result of one single event, in which case similarity might be under-estimated. In practice, it makes more sense to split the weight of indels into two distinct parts: one for the introduction of the indel and the for extending it. Scores for DNA and other biological sequences are most often defined by ad-hoc matrices, reporting, for each mutation a biologically-relevant value. Several schemes have been designed over the years based on various models, like BLOSUM62 [79] and PAM120 [3] for protein sequences.

Comparing two sequences can be done *globally* or *locally*. For global alignments, sequences are considered from end-to-end and their similarity is computed taking into account all bases. On the contrary, local alignments try to find the best matching of substrings, that is, some bases of S_1 and S_2 might not be taken into account in the similarity score. Matches can thus be ranked and reported from the most likely to the least probable.

Exact algorithms for computing similarity or align sequences are based on dynamic programming [145, 194, 71], but they are considered too computationally demanding even for moderately long sequences. Nevertheless, exact algorithms, or modified versions thereof, remain the last core step in various tools where they align homologous regions found by faster heuristics.

3.2 Full hash-based and succinct solutions

Heuristics to speed-up sequence comparison were first introduced in the programs FASTP [206] and FASTA [151] in 1985 and 1988, respectively. Both software were eventually replaced by BLAST [4] with their only remaining significant legacy being the ubiquitous Fasta format still in use today. Thanks to its higher sensitivity, better performance and rigorous statistical characterization of its output BLAST quickly overthrew FASTA as the de-facto standard in sequence similarity search. The main idea behind FASTA and BLAST is quite simple: index all k-mers (hashed or not) of a sequence in order to quickly prune for similar regions during alignment. Exact algorithms are only used in areas sharing a large fraction of k-mers (seeds). Such heuristic avoids running full algorithms (such as Smith-Waterman) over the whole sequences, instead limiting their use to localized, and smaller areas. BLAST is probably one of the most influential tools in bioinformatics (and in general), with almost 100K citations as the time of writing, still in use today for performing quick database similarity searches. Its limitations started to show with the introduction of Second-Generation sequencing. The huge throughputs of newer sequencers and the need to align more complex genomes, simply proved to be too much for BLAST to handle with reasonable performances.

This is why, almost immediately after the introduction of Next-Generation sequencing new aligners quickly started to appear. The basic idea of building an index to quickly prune the search space remains unchanged from the previous generation, with performance improvements primarily achieved by replacing k-mer indexing with full-text, succinct alternatives. Full-text indexing also allows for seed lengths not limited to k. Techniques such as: suffix arrays [130], suffix trees [204] and BWT-based indexing [28] (FM-Index [61]) have proved to be quite effective in terms of space. On the other hand, text-based mappers are less resilient to mismatches than their hash-based counter-parts, with every error leading to substantial slow-downs. Yet, the high quality of reads produced by Second-Generation sequencing makes these performance hits unlikely in practice.

3.3 Sampling for sequence alignments

Limitations of succinct text-based indexing appeared with the introduction of long noisy reads. Exact matches become hard to find in the presence of large errors. Even highly optimized solutions like BWA [112] were not able to give acceptable results. The answer was to combine classic hash-based (k-mer) indexing with sampling. Reducing k allows tuning for sensitivity since matches become more likely, even in the presence of errors. Sampling, on the other hand, allows reducing index size at the cost of negligible sensitivity losses.

The type of sampling can be important in practice. In principle, seeds should cover all parts of a sequence to avoid missing some potential homologies. Sampling k-mer uniformly at random by, for example, only keeping k-mers whose hash is 0, while easy to implement, provides no guarantee on the distribution of seeds along the sequence potentially leading to parts of the genomes that might not be covered enough resulting in unalignable reads.

Therefore, sampling techniques with known and provable coverage guarantees are preferred. Examples include minimizers [178, 167] and syncmers [52], the latter of which we use in Part IV. The first aligner implementing these concepts was minimap [113], quickly followed by its improved version minimap2 [114]. Note that, by default, both versions work as mappers, only providing base-to-base alignments as optional refinements of homologous regions found by using minimizer indexes.

Chapter 4

Alignment-free similarity estimation

Estimating sequence similarity from base-aligned sequences is best suited for applications that require high precision such as high-quality phylogeny reconstruction. In many other use cases however, exact retrieval of mutations is not a strict requirement and a simple estimation of similarity would be sufficient. This makes full alignments redundant and unnecessarily complex since they remain quite challenging and computationally intensive even with seeding heuristics. The idea of analyzing sequencing data without relying on actual alignments started to appear around the same time as Second-Generation sequencers. A classic example is the computation of sequence similarities by representing datasets as "bag-of-words", i.e. sets or multisets of k-mers. Simpler representations improve overall computational efficiency [18] allowing for comparisons at larger scales [15, 92, 193, 108]. Further, so-called alignment-free methods are resistant to shuffling and recombination events, since base order is completely irrelevant outside k-mers. Since then, alignment-free algorithms have been developed for the most disparate problems: phylogeny reconstruction [60, 65, 66, 108], metagenomic classification [207, 5], variant calling [180, 196, 49, 100], transcript quantification [23, 150] and a plethora of others. Here, we focus on methods computing sequence similarity, since it is one of the main ingredients of Part IV as well as the main motivation behind Parts II and III. Alignment-free methods computing similarities can be roughly classified into multiple categories depending on the type of information they use:

1. Set-based approaches represent sequences as simple sets of k-mers, without taking into account any other type of information [60, 108].

2. Frequency-aware solutions augment k-mers with their number of occurrences [160, 193, 127].

3. Methods based on information theory evaluate the information content of full-length sequences [203, 115].

4. Solutions based on the length of matching words (common [201], longest common [78] and minimal absent [158, 212] matches).

5. Solutions based on graphical representations of DNA [205, 164].

Different representations are usually associated to different measures of similarity which can be directly given in input to standard methods building phylogenetic-trees (see [223] for a review and benchmark). However, most of these measures were conceived without an underlying biologically-sound evolutionary model in mind [224]. This limitation becomes problematic when distances must be assigned to branches.

4.1 Jaccard similarity

One notable exception to the fact that most alignment-free measures lack a biological foundation is Jaccard similarity. Jaccard similarity (also known as Jaccard index) is defined for both weighted and unweighted sets [142, 129]. Unweighted Jaccard is the fraction of shared k-mers divided by their total number, and it is easily representable by set unions and intersections

$$J(x, y) = \frac{|A \cap B|}{|A \cup B|} \tag{4.1}$$

An estimator of the mutation rate between two sequences framed in terms of (unweighted) Jaccard similarity has been proposed by [60]. Full phylogenetic trees with accurate branch lengths can thus be inferred from sets of long-enough k-mers, and under the condition of uniformly distributed mutations. The same estimator was later reused in Mash [148] making Jaccard similarity one of the most successful alignment-free similarities in bioinformatics (more about this in Chapter 5).

On the other hand, weighted Jaccard is simply the extension of Jaccard to multi-sets. Given two vectors $x = (x_1, x_2, \ldots, x_n)$ and $x = (x_1, x_2, \ldots, x_n)$ with $x_i, y_i \geq 0, \forall i$ their weighted Jaccard similarity J^w is defined by:

$$J^w(x, y) = \frac{\sum_i min(x_i, y_i)}{\sum_i max(x_i, y_i)} \tag{4.2}$$

The utility of weighted comparisons for phylogenetic reconstruction has been questioned in [120]. While the rebuttal was targeting another popular frequency-based alignment-free technique (FFP [193]), it casts doubts on all other frequency-aware methods as well. Nevertheless, weighted Jaccard has been applied in other contexts, such as solving ambiguous alignments of very repetitive parts of the human genome [90, 89], or comparing the composition of metagenomes [14, 13].

4.2 Alignment-free algorithms at scale

The main drawback of k-mer-based alignment-free methods is the relatively high memory usage due to the costly and highly redundant nature of k-mers. The combination of rapidly declining sequencing costs (see Figure 4.1) and relative explosion of sequencing data coming from various initiatives, such as the Human Microbiome Project (HMP) [84], the Earth Microbiome Project [198] or the Global Ocean Survey [171] is putting current alignment-free pipelines under serious pressure. Analyzing data is quickly becoming the main bottleneck in many workflows. Additionally, available data today needs to be kept accessible and usable in the future so that new analyses can reuse it for validation/reproducibility purposes. Furthermore, count-aware alignment-free solutions have seen little to no developments on optimizing count storage, with most tools focusing on efficient representations of k-mers instead. Efficient k-mer count representations have only started to appear very recently [95, 154] in data structures designed to be static, memory-optimized alternatives of fully dynamic k-mer count tables. These monolithic solutions offer the same types of queries as fully dynamic alternatives, by storing both k-mers and counts together.

Another way to achieve space-efficient, static, general-purpose k-mer count tables for alignment-free algorithms is to add optimized count representations on top of already available solutions tailored at k-mer sets. Since both parts are completely independent to one another, multiple combinations of exact or approximate data structures become

possible depending on the desired trade-offs. In recent years, sketching data structures have gained particular interest in bioinformatics applications and, as such, we present them in the following section.

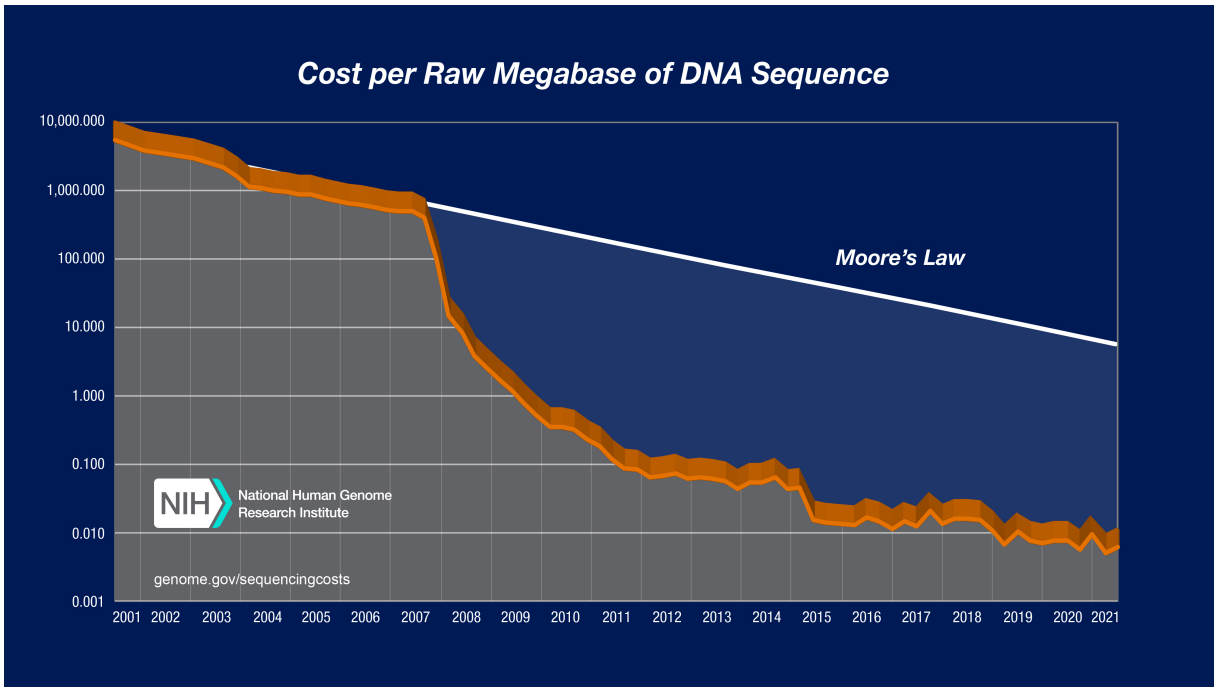


Figure 4.1: Sequencing cost against Moore's Law. Sequencing is becoming cheaper every passing year. Because of this, new data is produced at increasing pace straining current analysis methods.

Chapter 5

Sketching

Sketching refers to the general idea of applying some sort of dimensionality reduction technique on input datasets. Such ad-hoc reduced representations are called "sketches" and they can effectively replace the original inputs in all subsequent analyses. By reducing the dimensionality of data, often by several orders of magnitude, these techniques offer approximate answers to the specific applications they were built for. Computing estimations instead of exact results allow for much more favorable time and memory requirements overall. Original data can be deleted or stored elsewhere with additional sketches built as new inputs become available. Reduced memory size make sketching techniques an attractive alternative to exact alignment-free algorithms in big-data settings.

A vast majority of similarity-oriented sketching techniques available today are based on some sort of Locality Sensitive Hashing (LSH) scheme [110]. Put it simply, LSH tries to hash similar objects to similar hashes. Therefore, LSH schemes naturally cluster similar things together, under a given definition of similarity.

Here we provide a list of sketching techniques available at the time of writing this thesis.

1. **Sketches for similarity estimation** are great alternatives to alignment-free algorithms for pair-wise comparisons. Several techniques have been proposed:

   i **Set similarity.** MinHash sketching was first introduced in [26] as a method to detect duplicated web pages represented as sets. The original version of MinHash required multiple hash functions to work whereas the "bottom-s" variant uses only one. The "bottom-s" variant is at the basis of the seminal tool Mash [148], an implementation tailored to genomic datasets. Since then, Mash has been followed by many alternatives [27, 221, 216, 7, 56] and derived works [175, 103, 147, 99], including our own variant in Part IV which, unlike the competition, uses sketches initially conceived for set reconciliation [55, 69].

   ii **Multiset similarity.** Examples are [129, 85, 74, 116, 190, 208, 191, 37]. In particular, Histosketching [210, 211] is at the basis of the bioinformatics tool HULK [169].

   iii **Edit distance.** Edit distance sketches usually target database filtering, i.e. retrieving (w.h.p.) all sequences with edit distance within a user-defined threshold from the query. Classic solutions embed edit distance into simpler metrics (Hamming distance) [32, 218] with successful applications in bioinformatics [209]. Alternatives based on minimizers have also been proposed [134, 219].

2. **Sketches for count estimation** were first developed for estimating the multiplicity of objects coming from a stream (e.g. networking). Part II presents a modification of these methods specifically adapted for k-mer count tables.

i **Counting distinct elements.** The standard method for estimating the number of distinct elements in a dataset is HyperLogLog [62]. Recent works [216] have shown that sketches based on HyperLogLog can be used to estimate similarity and, as such, are tightly related to MinHash sketching.

ii **Count approximation.** The standard count-approximation techniques are CountSketch [33] and Count-Min sketch [42]. Both have been extensively studied [43, 41, 199, 197, 10] but surprisingly few bioinformatics applications use them in practice [220, 169]. This discrepancy can be explained by the large errors (up to a fraction of the size of the whole input) that potentially afflict count estimates. In k-mer based applications it is often of high importance to reliably distinguish low-frequency k-mers from more common ones, but having such large inaccuracies prevents that.

iii **Detection of heavy hitters or cold items.** Finding the most common (or rare) items in a data stream can be viewed as a sub-problem of count estimation. Sketches for approximating counts are often used as building-blocks of algorithms for this problem [33, 40].

3. **Graph sketching techniques** give approximate answers to a multitude of graph-related queries [137]: vertex connectivity [94, 72, 2], vertex cover [35, 6], triangle counting [31, 76] and set cover [31, 76].

5.1 Sketching techniques relevant to our work

We report here the main sketching techniques relevant to the works in Part II and IV. While Part II takes inspiration from count approximation sketches such as CountSketch and Count-Min (sections 5.1.1, 5.1.2, 5.1.3), Part IV use set reconciliation to challenge MinHash sketching in case of high similarity of the involved sets (sections 5.1.4, 5.1.5, 5.1.6).

5.1.1 CountSketch

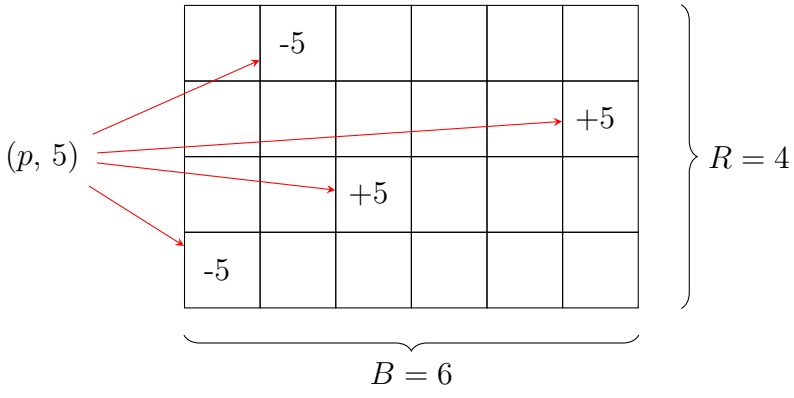


Figure 5.1: Example of inserting the pair $(p, 5)$ into an empty CountSketch. Empty cells represent counts equal to 0. Note how the sign of the inserted element in each cell depends on $s_i(\cdot)$

The CountSketch [33] is an approximated data structure that compactly stores an array $\boldsymbol{a}$ of counters. The sketch itself is an $R \times B$ matrix A of counters where each row i has associated two hash functions: $h_i(\cdot)$ and $s_i(\cdot)$. All $h_i(\cdot)$ and $s_i(\cdot)$ are supposed to be pairwise independent. Given an item p of frequency ℓ, doing $h_i(p)$ returns the bucket in

row i associated to p. On the other hand, $s_i(p)$ is a binary function returning values in $-1, 1$.

Updates are in the form of (p, ℓ) and are performed by doing $A(i, h_i(p)) = A(i, h_i(p)) + s_i(p) \times \ell$ for each row i (see Figure 5.1). Multiplication by the random sign $s_i(\cdot)$ ensures the unbiasedness of estimates. Collisions between different counters are a possibility. This introduces errors in the retrieved counts, which, as such, should more correctly be called "estimates". For this reason, queries are performed by computing $median_i(s_i(p) \times A(i, h_i(p)))$. Computing the median over R independent rows helps in reducing the effects of collisions involving very large counts. Averaging would not work as well because of its susceptibility to outliers of large magnitude. By choosing $B = \log(1/\delta)$ and $R = 1/\epsilon^2$ for two given parameters $0 < \delta < 1$ and $\epsilon > 0$, errors are guaranteed to be $< \epsilon \|\boldsymbol{a}\|_2$ with a probability of at least $1 - \delta$, where $\|\boldsymbol{a}\|_2$ is the L_2-norm of input vector $\boldsymbol{a}$ [33].

5.1.2 Count-Min sketch

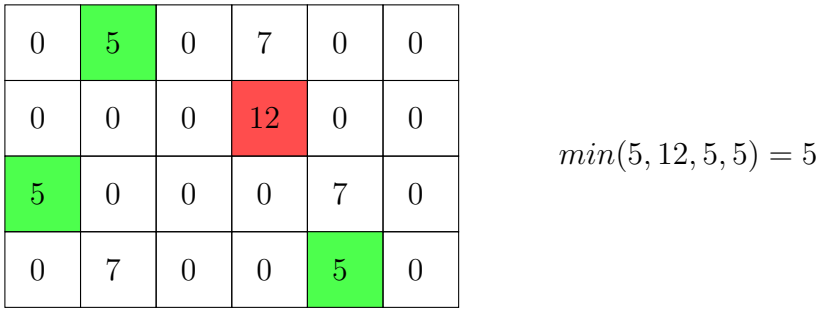


Figure 5.2: Example of query on a Count-Min sketch. Counters 5 and 7 corresponding to two different items were inserted in the sketch causing a collision in cell (2,4). In case of collisions in all cells of the sketch, taking the minimum minimizes error.

Similarly to Count-Sketch, Count-Min sketch [42] is a method able to represent an associative array $\boldsymbol{a}$ of counters in an approximated way. As before, Count-Min sketch is an $R \times B$ matrix A of counters with hash functions $h_i(\cdot)$ associated to each row. Hash functions $s_i(\cdot)$ are not required in this case, as we will see shortly. Updates are again in the form of (key, value) pairs (p, ℓ) for which Count-Min sketch perform $A(i, h_i(p)) = A(i, h_i(p)) + \ell$ for each row i. New counters are thus always inserted by addition and never by subtraction, making estimates returned by Count-Min sketches always biased by a positive quantity. The (approximate) current counter associated with a key p is retrieved as $\hat{\boldsymbol{a}}(p) = \min_i\{h_i(p)\}$. It is now easy to see from where Count-Min takes its name. Collisions between different counters lead to potential overestimation errors whose effects cannot be alleviated by multiplying each update with $s_i(\cdot)$. Taking the minimum of the R counters associated to an item p, instead of the median, is thus the best choice to minimize errors (see Figure 5.2). This simpler strategy also leads to a more straightforward analysis and error bounds. It is in fact known [42] that a Count-Min sketch built on a vector $\boldsymbol{a}$, with $R = \lceil \ln(\frac{1}{\delta}) \rceil$ and $B = \lceil \frac{e}{\varepsilon} \rceil$ for any given ε and δ, overestimates each returned count by at most $\varepsilon \|\boldsymbol{a}\|_1$, with probability at least $1 - \delta$, where $\|\boldsymbol{a}\|_1$ is the L_1-norm of $\boldsymbol{a}$.

If counts follow a Zipf distribution with parameter $a > 1$, B can be reduced to $O(\varepsilon^{-1/a})$ to guarantee the same bounds [42]. In Section 8.4 we show that these error bounds may not be acceptable for k-mer counting applications.

5.1.3 Count-Min sketch with conservative updates

Count-Sketch and Count-Min both support negative updates, i.e. they allow for $\ell < 0$ in updates (p, ℓ), provided that the cumulative count of each key remains positive at any given time. However, if updates are only positive, Count-Min can be modified for better accuracy entailing only a slightly different update procedure, first mentioned in [39] (therein attributed to [58]) as *conservative update*. Under this modification, updates for each row i are made according to $A(i, h_i(p)) = \max\{A(i, h_i(p)), \hat{\boldsymbol{a}}(p) + \ell\}$, where $\hat{\boldsymbol{a}}(p)$ is the current Count-Min estimate of $\boldsymbol{a}(p)$. It is easily seen that under this scheme, $\hat{\boldsymbol{a}}(p)$ can still only over-estimate $\boldsymbol{a}(p)$, but cannot be larger than $\hat{\boldsymbol{a}}(p)$ computed by the original Count-Min.

5.1.4 MinHash sketching

MinHash sketching was introduced in [26] as a method to estimate Jaccard similarity between two sets, applied to document comparisons. In bioinformatics, MinHash was first applied in Mash software [148] and then successfully used in a number of other tools. Assume we are given a universe U and an order on U defined via a hash function h. For a set $A \subset U$, the bottom-s MinHash sketch of A, denoted $\mathbb{S}(A)$, is the set of s minimal elements of A (or their hashes), where s is a user-defined parameter. The Jaccard similarity index between two sets A and B, $J(A, B) = |A \cap B|/|A \cup B|$, can then be estimated from the sketches of A and B, namely

$$\hat{J} = |\mathbb{S}(A \cap B) \cap \mathbb{S}(A) \cap \mathbb{S}(B)|/|\mathbb{S}(A \cup B)| \tag{5.1}$$

is an unbiased estimator of $J(A, B)$.

The Jaccard similarity between the k-mer sets of two datasets constitutes a biologically relevant measure of their similarity. In particular, if involved datasets are genomic sequences, this measure allows one to estimate the mutation rate between the sequences [60, 148].

5.1.5 Bloom filters

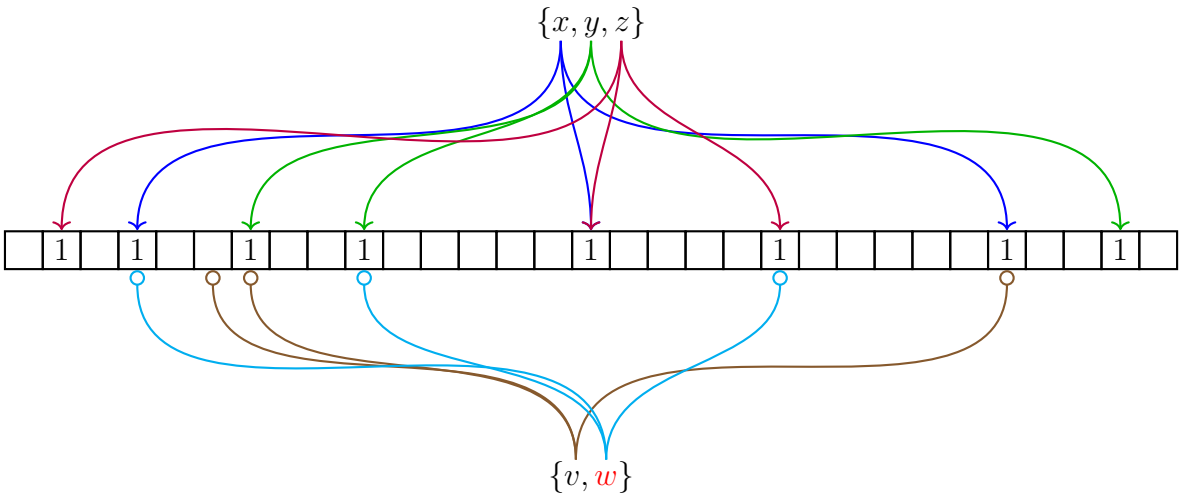


Figure 5.3: Using a Bloom filter with three hash functions to represent set x, y, z. Bits at positions given by the $r = 3$ independent hash functions are set to 1. At query, element v is correctly recognized as not present in the Bloom filter but w (in red) results in a false positive since all its bits are set to 1.

A Bloom filter (BF) is a well known data structure allowing for approximate membership queries [17]. Its key feature is the ability to favor space usage at the cost of a user-selectable false positive rate on each query. Given a set A of size $n = |A|$, a target false positive rate ε and a family of pairwise independent hash functions, a space-optimal Bloom filter B is a bit array of $m = -n\log(\varepsilon)/\ln(2) \approx -1.44n\log(\varepsilon)$ bits with $r = \ln(2)m/n \approx 0.693m/n$ hash functions associated to it. Inserting a new element p is as simple as setting to 1 the bits at positions given by the r hash functions when hashing p. Queries are performed by looking at the bits associated to p, if they are all set, then $p \in A$ while $p \notin A$ otherwise. False positives arise from the fact that elements $p \notin A$ might have all their bits set because of other insertions as shown in Figure 5.3. Applications where the retrieval of the actual elements stored in Bloom filters is important, have to rely on additional data structures for this purpose, with Bloom filters being used as temporary set representations [185]. For a set $F \subseteq U \setminus A$, we denote $FP_B(F)$ the set of false positives of F, of expected size $\varepsilon|F|$.

5.1.6 Invertible Bloom Lookup Tables

Invertible Bloom Lookup Tables (IBLT) [55, 69] are a generalization of Bloom filters for storing a set of elements (keys), drawn from a large universe, possibly associated with attribute values. In contrast to Bloom filters, in addition to insertions, IBLTs also support deletion of keys as well as listing. The latter operation succeeds with high probability (w.h.p.) depending on the number of stored keys relative to the size of the data structure. An important property is that this probability depends only on the number of keys stored at the moment of listing, and not across the entire lifespan of the data structure. Thus, at a given time, an IBLT can store a number of keys greatly exceeding the threshold for which it was built, returning to be fully functional whenever a sufficient number of deletions has taken place. Note also that IBLTs, in their basic version, don't support multiple insertions of the same key.

An IBLT is an array T of m buckets together with r hash functions $h_1, \ldots, h_r$ mapping a key universe U (in our case, k-mers or strings) to $[0..m-1]$ and an additional global hash function h_e on U. Each bucket $T[i], i \in [0..m-1]$, contains three fields: a counter $T[i].C$, a key field $T[i].P$ and a hash field $T[i].H$, where C counts the number of keys hashed to bucket i, P stores the XOR-sum of the keys (in binary representation) hashed to bucket i, and H contains the XOR-sum of hashes produced by h_e on keys.

Adding a key p to the IBLT is done as follows. For each $j \in \{1, \ldots, r\}$, we perform $T[h_j(p)].C = T[h_j(p)].C + 1$, $T[h_j(p)].P = T[h_j(p)].P \oplus p$, and $T[h_j(p)].H = T[h_j(p)].H \oplus h_e(p)$, where $\oplus$ stands for the XOR operation working with the binary representations of p and $h_e(p)$ (see Figure 5.4). Given that XOR is the inverse operation of itself, deletion of p is done similarly except that $T[h_j(p)].C = T[h_j(p)].C - 1$.

Listing the keys held in an IBLT is done through the process of peeling working recursively as follows. If, for some i we have $|T[i].C| = 1$, payload field $T[i].P$ is supposed to contain a single key p. Field H is not strictly necessary, it acts as a "checksum" to verify that p is indeed a valid key by checking if $h_e(T[h_j(p)].P) = T[h_j(p)].H$. This check is used to avoid the case when $|T[i].C| = 1$ whereas $T[i].P$ is not a valid key, which can result from extraneous deletions of keys not present in the data structure. In Section 14.2.2 we will elaborate on the role of this field in our framework. If the check holds, key p can be reported and deleted (peeled) from the IBLT. Updating hash sums and counters is then done in a way similar to insertion: $T[h_j(p)].H = T[h_j(p)].H \oplus h_e(p)$ and $T[h_j(p)].C = T[h_j(p)].C - 1$, $\forall j \in [1, \ldots, r]$. The procedure continues until all counters $T[i].C$ are equal to zero. The first part of Figure 5.5 shows how the removal of elements can make an unpeelable sketch to be peelable again. The second part illustrates the listing

procedure itself.

At each moment, an IBLT is associated to an r-hypergraph where nodes are buckets and edges correspond to stored keys with each edge including the buckets a key is hashed to. Listing the keys contained in an IBLT then relies on the peelability property of random hypergraphs [50, 139]. Assume our hash functions are fully random. Then it is known that for $r \geq 3$, a random r-hypergraph with m nodes and n edges is peelable w.h.p. iff $m \geq c_r n$ where c_r is a constant peelability threshold. The first values of c_r are $c_3 \approx 1.222, c_4 \approx 1.295, c_5 \approx 1.425, \cdots$ [69]. Thus, allocating

$$m = n(c_r + \varepsilon), \tag{5.2}$$

buckets, for $\varepsilon > 0$, for storing n keys guarantees successful peeling with high probability.

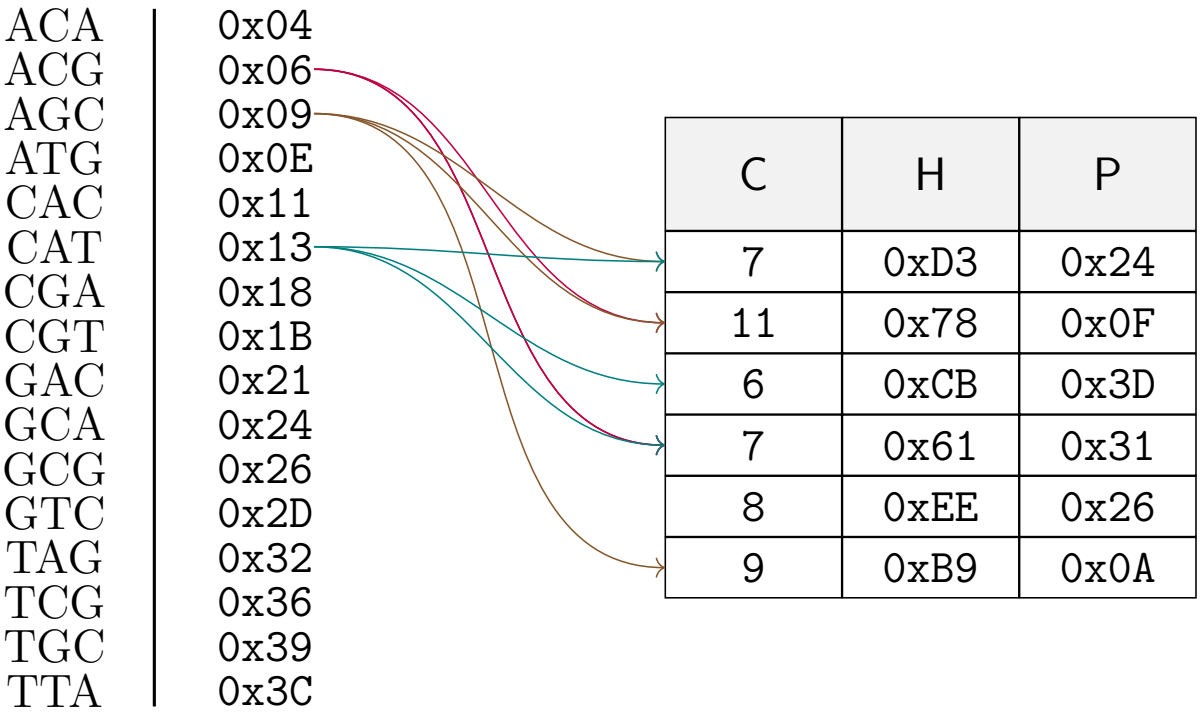


Figure 5.4: Example of insertions in an IBLT with $m = 6$ and $r = 3$. The binary representation of each k-mer is inserted into payload field P of $r = 3$ non-necessarily distinct buckets given by hash functions $h_j(\cdot), j = 1, 2, 3$. The resulting table is not peelable, since all buckets contain more than one element (all counts are different from 1 (or -1)). Note however that IBLTs support deletions and the sketch might return to be peelable again if enough inserted elements are removed.

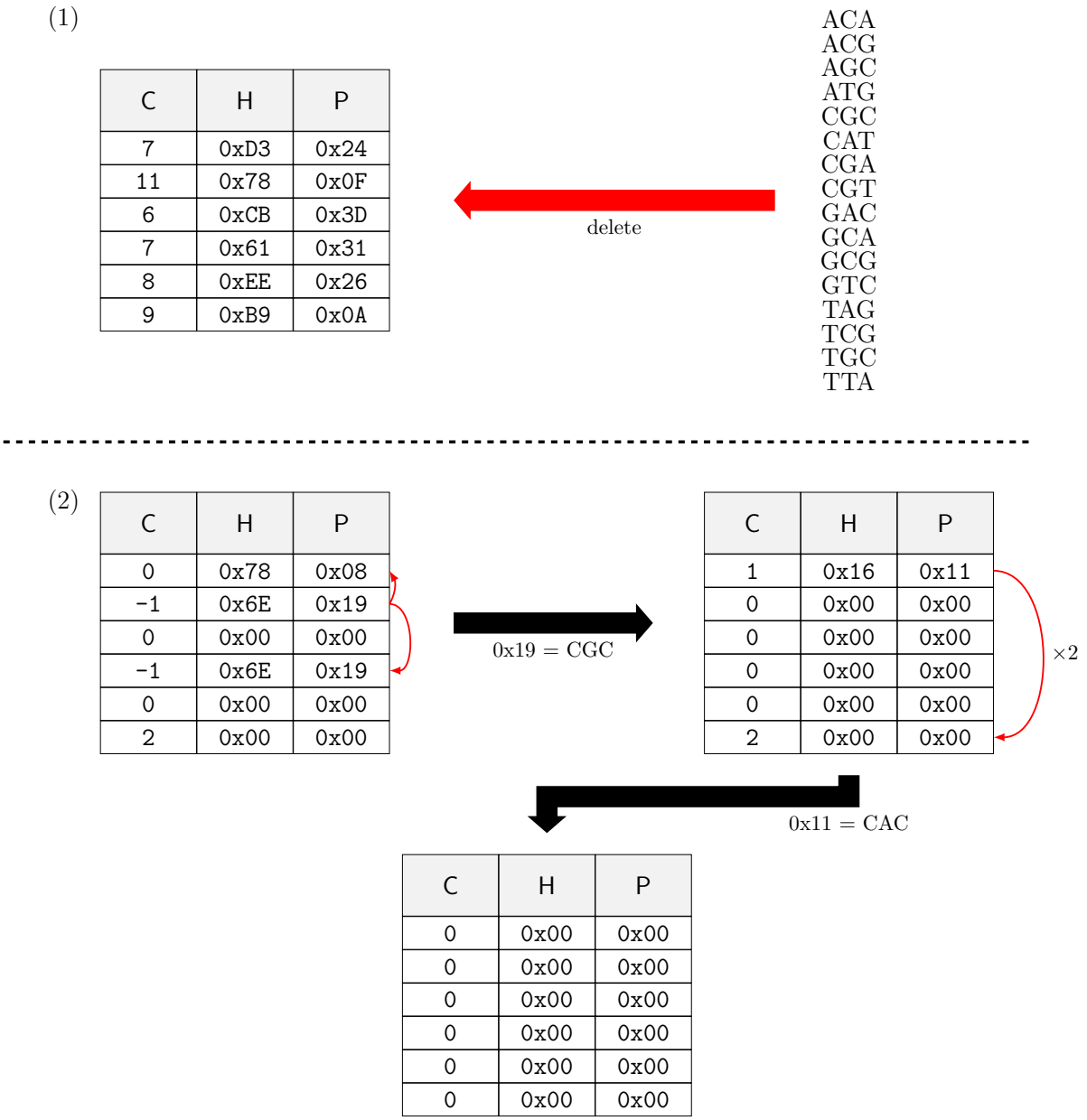


Figure 5.5: The reported example represents the difference between the k-mer set from Figure 5.4 and a modified version of itself, where k-mer CAC has been mutated into CGC (1). Once sketched, the difference (top left of (2)) has two buckets containing one element each ($C = -1$), and two buckets with two elements each (first and last rows). Despite having $C = 0$, the first row contains something (H and P fields are different from 0x00) while the last one is subject to a collision of two copies of the same element ($C = 2$ and $H = P =$ `0x00`). Peeling starts by first finding any eligible bucket whose counter is 1 or -1 and whose hash field H is equal to $h_e(P)$. This is the case for the bucket containing 0x19 (top left). Value 0x19 (corresponding to CGC) is then subtracted from all its buckets, leading to a new peelable bucket (first bucket of the second sketch in the top right corner of (2)). Peeling this new item (0x11 = CAC) empties the sketch and terminates the process. The retrieved symmetric difference is thus CGC, CAC with CAC coming from the set depicted in Figure 5.4 (counter $C = 1$) and CGC being its mutated version ($C = -1$).

Chapter 6

Exact representations of k-mer count tables

As it has been the case for sketches, we do a quick review of the main methods we use in our solution in Part III. Despite the fact that the structures of Part III do not store k-mers but only frequencies, we start this review by listing full static k-mer table representations first. This should provide a more general view helping to put the algorithms effectively used in this thesis into context.

6.1 Full static representations

This section is dedicated to k-mer count table representations that store both k-mers and counts, thus supporting both counter retrieval and extraneous k-mer identification. Exact presence/absence support is costly and not always required if queries are restricted over a known set of k-mers. Nevertheless, in applications where support for both types of queries is justified, exact dictionaries are generally the preferred option [154]

6.1.1 Hash Tables

Common hash table implementations found in standard libraries of most programming languages provide the easiest and quickest (in terms of development time) solution for storing (key, value) pairs. However, this leads to suboptimal space for both k-mers and counts, given that the former are highly redundant overlapping strings, while the latter follow skewed distributions. Solutions tailored to genomic datasets have been the subject of extensive research efforts leading to the many results of the following (sub)sections.

6.1.2 Quotient filters

Quotient filters try to reduce space by storing lists of sorted k-mers. k-mers are then split into prefix and suffix with contiguous stretches of equal prefixes implicitly represented as positions in an array (or just stored once using an exact set representation). Queries are performed by first searching for the prefix of a given k-mer. If it is found, then the search continues in the block of suffixes associated with the prefix. Counts are stored into additional fields (one for each suffix).

All modern k-mer counters use some sort of variation of this method [135, 101, 149] with DSK [165] the sole exception. Latest advancement also try to optimize space weights by dynamically allocating variable-length counters for count values [189].

6.1.3 Dictionaries based on super k-mers

Super k-mers are groups of contiguous k-mers sharing the same minimizer [119]. Bucketing super k-mers by minimizer allows for fast membership queries. Queries are performed by finding the set of super k-mers associated to the minimizer of a query k-mer p. Searching the queried k-mer as a substring of the super k-mers gives the answer. Mapping minimizers to their sets of super k-mers is achieved with the help of Minimal Perfect Hash Functions (see 6.2.1). Two implementations of this method are BLight [133] and the more recent ssHash [155] and its extension supporting k-mer counts [154].

6.1.4 Graph-based dictionaries

Another family of k-mer dictionaries is obtained by extending de Bruijn graphs with counts. Classic approaches rely on MPHFs to link additional information to k-mers in a graph [82], whereas recent solutions store k-mers and counts into two separate data structures optimized for their respective tasks [98]. A related approach is the structure proposed in [86] where delta-encoded weights are added on top of the BOSS data structure (a de Bruijn graph stored as a set of contigs) [19]. Note that, in general, FM-Indexes [61] naturally offer a *count* operation, if a given k-mer is found, then its number of occurrences can also be retrieved.

6.2 Count-only data structures

Count-only data structures do not explicitly store k-mers by limiting queries to a known set of k-mers. Further space optimizations of the weights becomes possible, at the cost of losing membership queries. This strategy is useful when k-mers are known in advance and counts need to be retrieved multiple times during execution. While not exactly equivalent to full k-mer count tables, they are strictly related concepts, often with readily-available drop-in replacements for a wide range of use cases. Methods in Part III belong to this family.

6.2.1 Minimal Perfect Hash Functions

Minimal Perfect Hash Functions (MPHFs) are bijective functions between keys of a set $S \subset U$ and integers in the range $[0, |S|-1]$. By using hash values as indexes for an external array, it is possible to associate any type of information to k-mers. Unlike the previous structures, keys in S are not explicitly represented, making answering membership queries impossible. On the other hand, not storing S leads to much greater space efficiency, with a theoretical lower bound of $\log e = 1.44$ bits/element [138]. However, this lower bound is achievable only in theory, for sets so big that they have no practical interest [75]

MPHFs can be categorized into four families [157]:

- **Hash and displace** MPHFs belonging to this family are FHD [63], CHD [11] and ptHash [157, 156]. They work by first splitting the input set into multiple buckets B_i of different (FHD) or homogeneous (CHD) size. Buckets are then serially processed in order to find integer values d_i, such that all elements in B_i can be inserted into a global binary array without collisions. FHD maps elements to their positions inside the binary array by doing $(h(x)+d_i \mod |S|)$ with buckets processed by decreasing size, where $h(\cdot)$ is a random hash function. On the other hand, both CHD and ptHash use two pairwise independent hash functions h_1 and h_2 to avoid failure if d_i reaches $|S|$. CHD computes positions by doing $(h_1(x) + z_0 h_2(x) + z_1 \mod |S|)$ with the displacement value d_i of bucket B_i the index of pair (z_0, z_1) in sequence

$(0,0),\cdots,(0,|S|-1),\cdots,(|S|-1,0),\cdots,(|S|,|S|)$. On the other hand, ptHash does $(h_1(x,s) \oplus h_2(d_i,s) \mod |S|)$ with s a user-defined seed. In case of successful construction, the binary array can be discarded and only the list of displacements is saved. Such vector is usually stored using succinct data structures, with ptHash achieving good time/space trade-offs thanks to the improved compressibility of its d_i values.

- **Linear systems** MPHFs use the relation between linear systems of $n = |S|$ equations and m variables to random r-hypergraphs [128]. If the ratio between n and m is above a certain threshold that depends on the parameter r of the hypergraph, then it is possible to assign a unique ordering to the n keys with high probability.

- **Cascading** First introduced in [144] this type of MPHFs are built by repeatedly resolving collisions when inserting keys into arrays. The algorithm starts by hashing all elements in S to a bit-vector v_0 of size $n_0 = |S|$. Bits corresponding to exactly one element are set to 1, unused bits and collisions are marked by 0 (collisions need to be temporarily remembered using additional bits). The n_1 elements colliding in v_0 are then inserted again into a new bit-vector v_1 and so on and so forth. On average, this procedure stops after 1.56 levels in the most succinct setting [144]. The final MPHF is the concatenation of each layer with added support for fast rank queries. Note that, during construction, it is thus necessary to save the sets of colliding keys or to re-run the construction algorithm over the entire stream of keys multiple times, generating a new array for each layer. The index of a key is retrieved by a query to a rank/select data structure counting the number of cells occupied before the wanted element. An implementation of this technique can be found in [122] where a parameter γ allows to increase the space of each bit-vector for faster construction times.

- **Recursive splitting** Suitable MPHFs can be found by brute force for small sets [57]. The above observation can be recursively applied in a divide and conquer approach to build MPHFs for larger sets. The downside of this approach are the slow queries due to the generation of a tree of splittings over multiple levels.

6.2.2 Compressed static functions

A static function (SF) is a representation of a function defined on a given subset S of a universe U such that an invocation of the function on any element from S yields the function value, while an invocation on an element from $U \setminus S$ produces an arbitrary output. The problem has been studied in several works (see references in [11, 67]) resulting in several solutions that allow function values to be retrieved without storing elements of S themselves. One natural solution comes through MPHFs: one can build a MPHF for S and then store function values in order in a separate array. This solution, however, incurs an overhead associated with the MPHF, known to be theoretically lower-bounded by about 1.44 bits per element of S.

This overhead is especially unfortunate when the distribution of values is very skewed, in which case the value array may be compressed into a much smaller space. Compressed Static Functions try to solve this problem by proposing a static function representation whose size depends on the *compressed* value array. The latter is usually estimated through the zero-order empirical entropy, defined by $H_0(f) = \sum_{\ell \in L} \frac{|f^{-1}(\ell)|}{|K|} \log(\frac{|K|}{|f^{-1}(\ell)|})$, where L is the set of all values (i.e. $L = \{f(t) \,|\, t \in K\}$) and $f^{-1}(\ell) = \{t \,|\, f(t) = \ell\}$ is the set of k-mers with value ℓ. $|K| \cdot H_0(f)$ can be viewed as a lower bound on the size of compressed value array, in absence of additional assumptions. Thus, the goal of CSFs is

to approach the bound of $H_0(f)$ bits per element as closely as possible, in representing a static function f. We refer the reader to [11, 67] and references therein for an overview of different algorithmic solutions for SFs and CSFs.

[11] proposed a solution for CSF taking an asymptotically optimal $nH_0(f)+o(nH_0(f))$ space (n size of the underlying value set), however the solution is rather complex and probably not suitable for practical implementation. As of today, to our knowledge, the only practical implementation of a CSF is GV3CompressedFunction [67], found in the Java package Sux4J (`https://sux.di.unimi.it/`). Although entropy-sensitive, the method of [67], has an intrinsic limitation of using at least 1 bit per element, due to involved coding schemes. This is a serious limitation when dealing with very skewed distributions of values, where one value occurs predominantly often and the empirical entropy can be much smaller than 1. This is precisely the case for count distributions in whole genomes (see Section 8.1.1).

Part II

Sketching count information

Chapter 7

Context and motivation

7.1 Problem statement

Counting every substring of length k in genomic sequences is a rather common task in many bioinformatics pipelines. Representing sequences as sets of k-mers is the idea behind many alignment-free algorithms with k-mers being used for efficient sequence similarity estimation and/or to quickly find seeds during the initial steps of sequence alignment. However, augmenting k-mers with their counts allow for a much greater range of possibilities. For example, k-mer count statistics can be used to filter too common k-mers and only use the rarer ones as seeds for greater specificity. Another use is genome size estimation from sequencing reads or, similarly, sequencing coverage estimation. Read trimming also makes use of k-mer frequencies in order to ignore uninformative parts of sequenced reads [125]. The same information can also be used to correct reads if its k-mer frequencies seem unlikely. Other applications range from detecting sequence duplication to alignment-free variant calling [163, 100].

In recent years, many k-mer counting algorithms have been proposed, such as Jellyfish [135], DSK [165] or KMC [101]. All these tools output a map associating k-mers to their counts. Such a map can require a fairly big amount of disk space, especially for large values of k because it stores both genomic (i.e. the k-mers) and count data together. For example, the binary file produced by KMC when counting a human genome with $k = 32$ weights around 28 GB. Almost all tools, including KMC, employ efficient representations of k-mers, such as quotienting, in order to reduce their memory footprint but, even with such optimizations, memory efficiency remains an important issue.

Many applications, however, only deal with k-mers that come from partially assembled reads or succinct representations of k-mer sets, such as colored de Bruijn graphs [82, 140], or spectrum-preserving string sets [162, 29], that is, only k-mers present in the original data are queried for their frequencies. For this reason, a way to reduce memory usage is to store counters only, together with a mapping linking them to their k-mers, without the need to reserve space for any genomic information. The idea of storing only counter information and not k-mers is also supported by the observation that the number of distinct k-mer counts in genomic data is relatively small. It is in fact known that k-mer counts in genomes obey a "heavy-tail" power-law distribution with a relatively large absolute value of the exponent [44]. For such distributions, the number of distinct k-mers makes a linear fraction of the data size, while the number of distinct counts is relatively small. For example, for the human genome and $k = 27$, there are about 2.5 billions distinct k-mers but about only $8,000$ distinct frequency values. In many practical cases, the majority of k-mers have very small counts: in the above example, 97% are unique, while almost 99% have a count of at most 5. Furthermore, frequent k-mers often tend to have identical frequencies as well, due to transposable elements: for example, k-mers

specific to Alu repeats in primate genomes will likely have the same count. It is thus possible to take advantage of these properties to reduce space by trying to share count values with multiple k-mers.

7.2 Contributions

The count approximating sketches described in Section 5.1 have seen little to no application in bioinformatics even though: i) k-mers usually follow highly skewed power-law distributions [44, 36], and ii) both CountSketch and Count-Min sketch are known to have an advantage under this particular setting [43]. In most cases they are only applied as temporary storage solutions while performing more complex operations [169, 220] with none used for permanent storage of counts. The answer to this apparent paradox resides in the streaming nature of both sketch constructions, where updates are allowed to arrive in the form of unary increments. While general, this constraint limits the possibility to take full advantage of the peculiar distribution of the weights in order to reduce errors.

Basically, the goal of this work is to provide an approximated technique for the long-term storage of counts for a known set of k-mers. To this end, we propose a new probabilistic data structure that we call Set-Min sketch, capable of representing k-mer count information in small space and with small errors. Set-Min sketch and its analysis follow the same path as the aforementioned sketches by saving counts into a matrix, and by avoiding collisions through multiple independent hash functions. The distinguishing property of our method compared to, e.g. Count-Min sketch, is that the matrix stores sets of counters instead of sums. Changing counter representation leads to better error guarantees, or in other words, to achieve the same error guarantees, Set-Min sketches need less space than previous solutions.

Memory-wise sets are more efficient than simple counters in two cases:

- The number of possible count values is small.
- Counts are distributed following a "heavy-tail" power-law distribution. The higher the skew, the smaller the space required by Set-Min.

The main insight is that the same count value in a given cell of the sketch can be shared by multiple k-mers. Adding a new k-mer p with its counter c is as simple as adding c to the r cells of the matrix associated to p. Retrieval, on the other hand, is performed by set intersections which has the potential to retrieve the exact values associated to k-mers. Thanks to these modifications, Set-Min guarantees that the expected *cumulative* error obtained when querying all k-mers of the count table T it represents can be bounded by a fraction of the total number of k-mers (including multiplicities) in T. This is in stark contrast to Count-Min where the same bound applies on *each* single query. Note, finally, that Set-Min sketch is a general data structure in that it can be used to efficiently represent a mapping of k-mers to any type of labels, provided that the number of possible labels is relatively small.

We introduce these considerations more formally in Section 8.1 together with the concept of skewed distributions. Section 8.2 introduces the data structure itself and how insertion and queries are performed, respectively. The theoretical analysis of our sketch can be found in Section 8.2 together with a practical algorithm for dimensioning the sketch according to the desired error bound. We present experimental results in Section 8.4 on a range of datasets illustrating the benefits of our approach. We provide comparisons against alternative solutions, namely Count-Min sketch (and its optimized version called Max-Min sketch) and, on the other hand, Minimal Perfect Hashing. An implementation of Set-Min can be found at `https://github.com/yhhshb/fress`.

Chapter 8

Set-Min sketch

Similarly to Count-Min sketch (Section 5.1.2), Set-Min itself does not store k-mers. Instead, it estimates frequencies by relying on a matrix of sets instead of numerical counters. The intuition behind this modification is presented in Section 8.1. We follow by formally presenting our method and its analysis in Section 8.2. Finally, Section 8.4 experimentally validates our results on a practical implementation of our ideas.

8.1 Key algorithmic ideas

Count-Sketch and Count-Min sketches use independent and distinct insertions at different cells to mitigate the effects of collisions between different counts (Section 5.1). Both can be applied in a streaming setting where count values for each item arrive as a series of partial updates. The main drawback of using sums to update the content of each bucket is that the more values hash to a cell, the larger the expected error will be, since it becomes impossible to retrieve the original unaggregated values. In applications where counts arrive already computed, and the interest of sketching is more towards space rather than streaming, it might be interesting to take advantage of the distribution of the input to boost compression. Both Count-Sketch and Count-Min are not prone to such improvements since input values are lost due to the sum operations. This chapter introduces the idea of a sketching method tailored to efficiently represent power-law distributed counts by taking advantage of collisions instead of being hindered by them.

8.1.1 Skewed distribution

A k-mer spectrum is a distribution of k-mer frequencies across all k-mers occurring in the data, showing how many k-mers support each frequency value. For large values of k, k-mer spectra follow a power-law distribution [44, 36] characterized by a linear-like dependence when represented in the log-log scale. That is, the k-mer frequency distribution fits a dependence $f(t) \approx c \cdot t^{-a}$, where a is usually greater than 2. An example of the spectrum for the human reference genome with k equal to 32 is given in Figure 8.1. According to this distribution, a very large fraction of k-mers has very low frequencies, while only a handful of k-mers are "unexpectedly" very repetitive. Large values of a imply that there are relatively few distinct frequency values with non-zero support, whose number is given by the $\frac{1}{a}$-power of the total number of k-mers. That means that, for large-enough k's, few distinct frequencies are expected to appear on the x-axis of a spectrum.

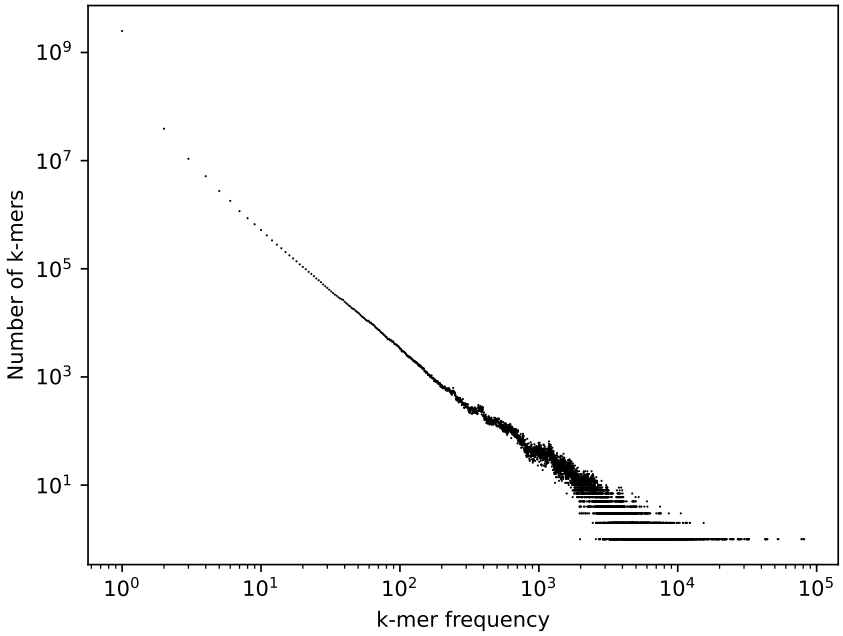


Figure 8.1: k-mer spectrum of the human genome for $k = 32$ in log-log scale. Note how the number of highly repetitive k-mers rapidly decreases as their repetitions increase. Most of the k-mers in a fully assembled genomes are thus unique, for large enough ks.

8.1.2 Using counter collisions to reduce space

In order to make use of the few distinct frequency values, we will limit our method to sketching already constructed count tables. As we saw earlier (Chapter 7), counting k-mers is a well known problem with optimized solutions readily available [101, 135, 165]. The only downside of limiting ourselves to full tables is that it prevents our sketch to update counters of already inserted k-mers. On the other hand, knowing exactly which counters have been inserted opens up the possibility of re-using the values already present in buckets for later insertions. This can be done quite easily by replacing each cell of a Count-Min matrix with a set of counters. We call the resulting data structure *Set-Min* sketch. Unlike Count-Min, Set-Min sketches are able to represent mappings of very skewed distributions (e.g. power-law distributions with large a) in less space, by sharing the same counters across multiple k-mers of the same buckets.

Estimates returned by Set-Min sketches are not affected by the sum of colliding k-mers's counts as in Count-Min sketches or Count-Sketches. Instead, each count value in a set is treated as a distinct object, and errors can only come from returning the wrong value instead of the correct one. This can happen with probability < 1 even when rows are made of $B \ll L_0$ buckets, where $L_0 = \|\boldsymbol{a}\|_0$ is the number of distinct k-mers in a counting table $\boldsymbol{a}$. This is not the case for neither Count-Min nor Count-Sketch whose estimates are always affected by some errors, if $B \ll L_0$. As we will see in Section 8.2.1.1, this intuition behind Set-Min sketches leads to better overall estimates than competing methods.

8.2 Set-Min data structure

We now more formally introduce the Set-Min sketch data structure. Assume we are given a set K of keys with associated values taken from a set L with $|L| \ll |K|$. In our case,

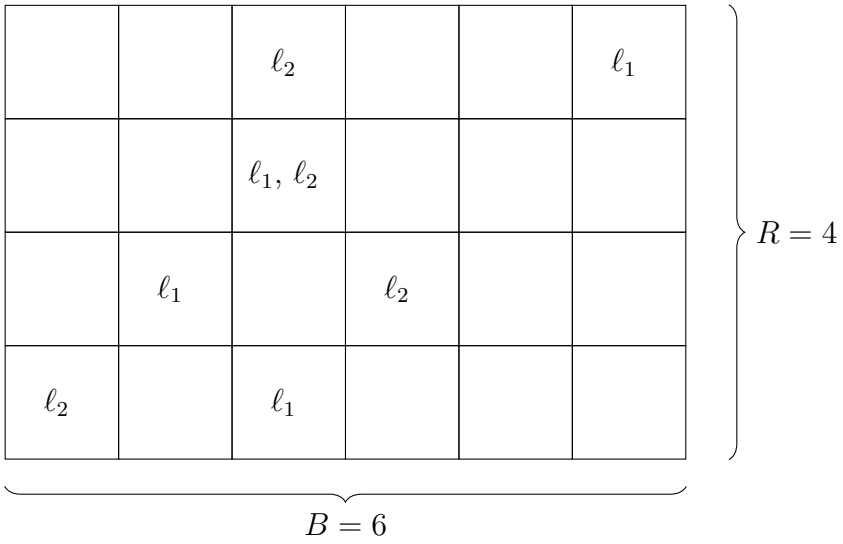


Figure 8.2: Example of a Set-Min sketch with $L = \{\ell_1, \ell_2\}$. Two pairs (e, ℓ_1) and (f, ℓ_2) with $e \neq f$ have been inserted into the sketch, with e, f hashed to the same bucket at line 2.

K is a set of k-mers and L the set of their frequencies, although our method will hold for any set of labels L, not necessarily numerical. We want to compactly implement the associative map of (key, value) pairs. A Set-Min sketch is an $R \times B$ matrix M where each bucket is treated as a set, initially empty. Similar to Count-Min sketch, rows in the matrix correspond to hash functions h_i, $0 \leq i \leq R-1$, that we assume pairwise independent.

At construction time, the key of each (key, value) pair (p, ℓ) is hashed by the hash functions to retrieve its buckets, and the value ℓ is inserted into each set. Formally, we update $M(i, h_i(p)) = M(i, h_i(p)) \cup \{\ell\}$ for each row i (see Figure 8.2).

Retrieval of a value associated with a key p, is performed by simply computing the intersection of the sets corresponding to p, that is $\cap_{0 \leq i \leq R-1} M(i, h_i(p))$. In general, three outcomes are possible:

1. The intersection is a singleton, in which case the exact value associated to p is returned.

2. The intersection contains more than one value, which corresponds to a collision. One value must be selected as the answer.

3. The intersection is empty, p is not present in the map.

However, since in this work we assume that only k-mers present in the dataset can be queried, empty intersections (case 3) becomes impossible. This in turns, allows for tighter sketch dimensions. Not having to worry about recognizing alien k-mers means that the most common label, i.e. the label ℓ_1 with the largest support, can be in every bucket without affecting errors. In other words, it becomes the "default" value returned by the sketch.

Instead of potentially having the same value physically stored into each set, we further optimize by ignoring it. ℓ_1 thus becomes the new case 3 and is retrieved implicitly: when the intersection is empty, ℓ_1 is returned. In practice, this optimization allows us to save space and will be further discussed later. Note that, with these modifications, an error may occur even if the resulting intersection is a singleton, but the right label is actually ℓ_1. In case of k-mers and for large k, ℓ_1 is usually equal to 1, which is the frequency of the largest fraction of k-mers. A graphical representation with colored labels of the above modifications can be seen in Figure 8.3.

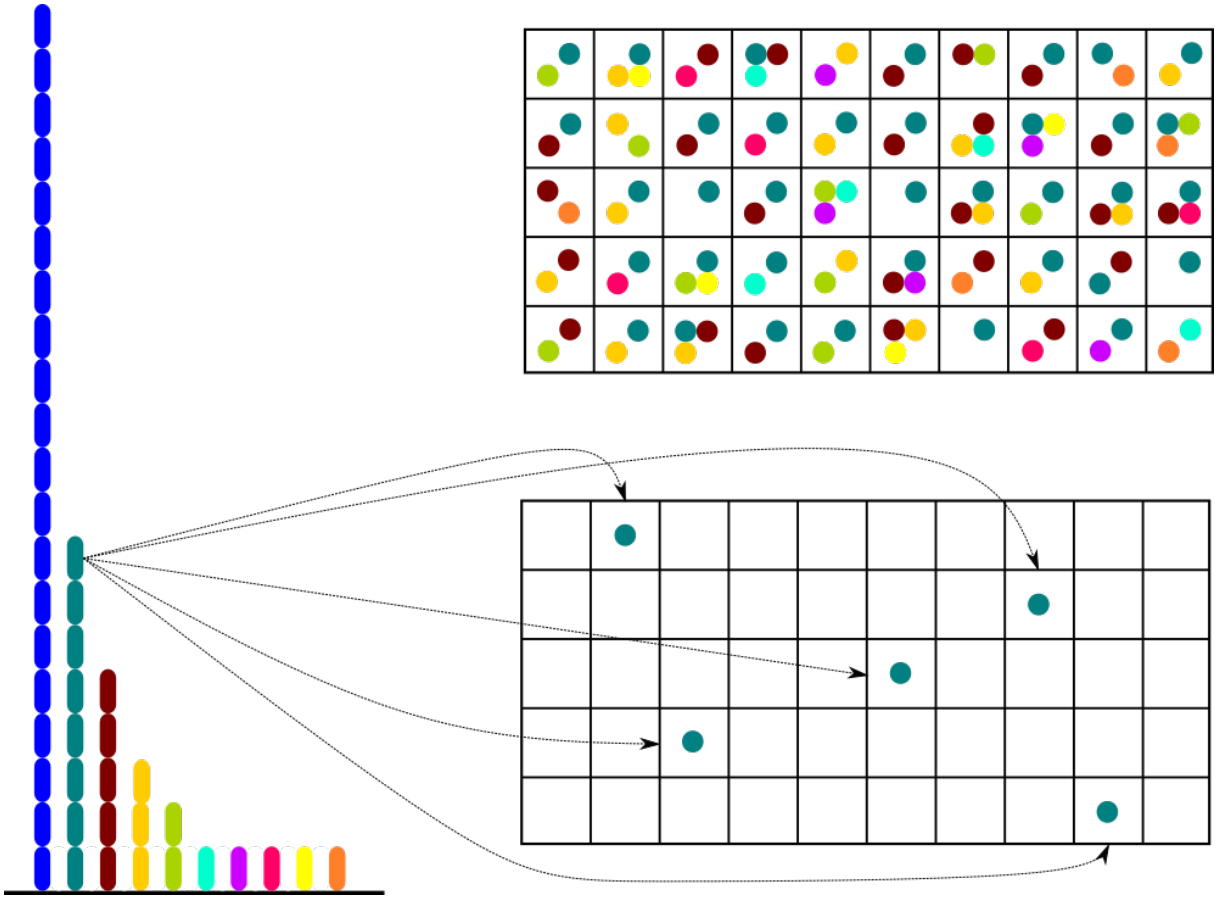


Figure 8.3: Set-Min sketch memory optimization. The most common label (in blue) in the histogram (left) is not actually inserted into the sketch (bottom right). Empty intersections of the final sketch (top right) are interpreted as the missing label.

8.2.1 Dealing with collisions

The only thing left to decide is what label to return in case of collisions. This choice is guided by the number of k-mers supporting each label of the intersection: the label with the smallest support is returned (see Figure 8.4). The rationale for this is that the label with the smallest support has the smallest probability to appear "by chance". On the other hand, labels with larger supports belong to more buckets in the sketch, and are therefore more likely to occur in the intersection by accident. Thus, the algorithm compares spectrum values for all labels in the intersection, and returns the label with the smallest value (ties are broken randomly). If the spectrum is monotonically decreasing (as it is usually the case for large k, see Figure 8.1), then the label returned is simply the largest one among those in the intersection.

8.2.1.1 Bounding the total error

We now show that with Set-Min sketch, we can bound the *total* absolute error over all k-mers of the dataset. Consider a sketch S built on a map assigning to each k-mer $p \in K$ a value (label) $\ell_p \in L$ which is the frequency of p in the dataset. We denote by c_ℓ the number of k-mers with frequency $\ell \in L$ (spectrum value).

Consider

$$D = \sum_{p \in K} |\hat{\ell}_p - \ell_p| \tag{8.1}$$

where $\hat{\ell}_p$ is the label of p returned by the sketch. Our goal is to dimension R and B such that $D \leq \varepsilon \|\boldsymbol{a}\|_1$, where $\|\boldsymbol{a}\|_1$ is the total number of k-mers in the dataset (roughly, the

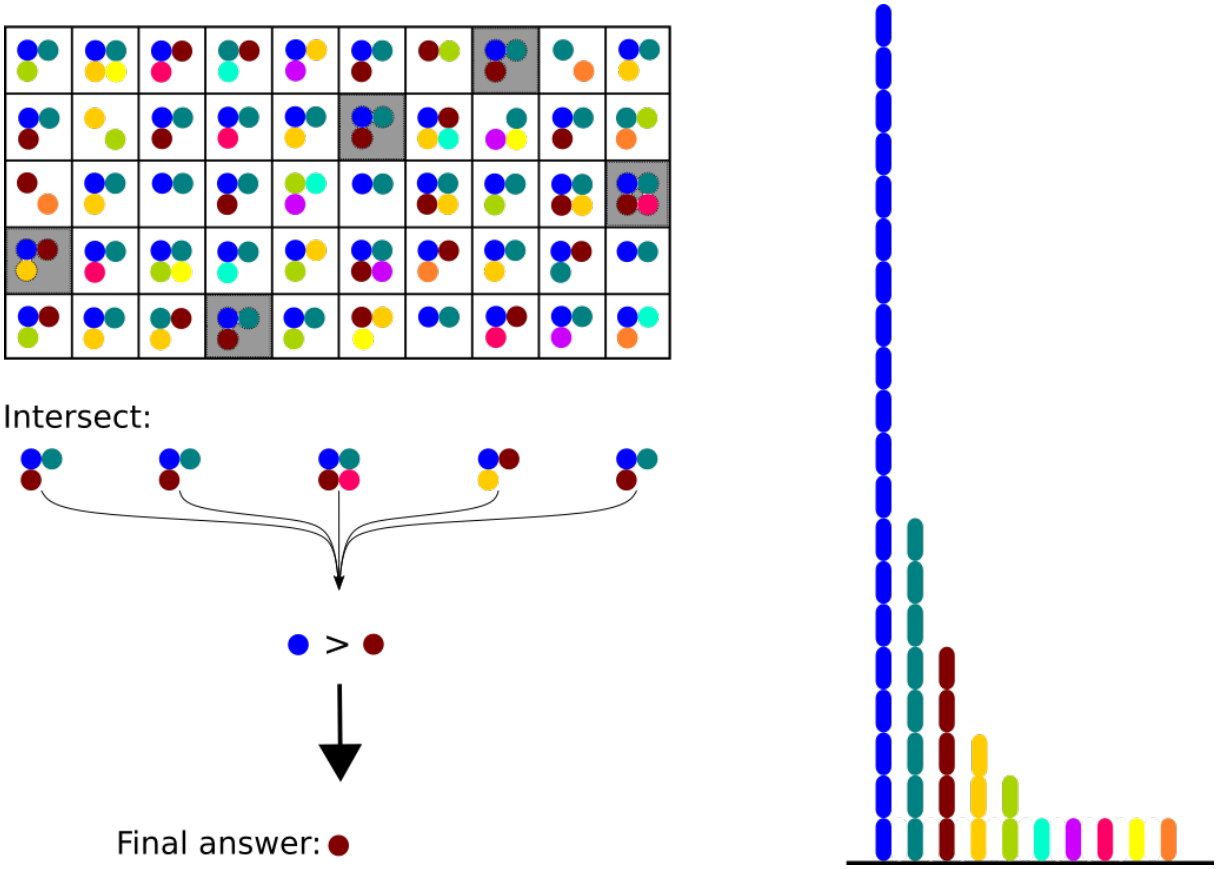


Figure 8.4: Example of collision resolution in case of multiple items occurring in the intersection. The brown label is returned because it is more rare compared to the blue one.

dataset size) and $0 < \epsilon \leq 1$.

Querying p returns an incorrect frequency $m \neq \ell_p$ iff m occurs in the intersection and $c_m < c_{\ell_p}$. The probability of this event is

$$\left(1 - \left(1 - \frac{1}{B}\right)^{c_m}\right)^R \approx \left(1 - e^{-\frac{c_m}{B}}\right)^R \tag{8.2}$$

and the expectation of the error when querying p is then

$$\sum_{\substack{c_m < c_{\ell_p} \\ m \in L}} |m - \ell_p| \left(1 - e^{-\frac{c_m}{B}}\right)^R. \tag{8.3}$$

Summing up over all k-mers, we obtain

$$\mathrm{E}[D] = \sum_{\ell \in L} c_\ell \sum_{\substack{c_m < c_\ell \\ m \in L}} |m - \ell| \left(1 - e^{-\frac{c_m}{B}}\right)^R. \tag{8.4}$$

The total number of k-mers is $\|\boldsymbol{a}\|_1 = \sum_{\ell \in L} \ell c_\ell$. Given $0 < \varepsilon \leq 1$, our goal is to choose B and R in order to ensure

$$\sum_{\ell \in L} c_\ell \sum_{\substack{c_m < c_\ell \\ m \in L}} |m - \ell| \left(1 - e^{-\frac{c_m}{B}}\right)^R < \varepsilon \sum_{\ell \in L} \ell c_\ell. \tag{8.5}$$

Assuming that k is sufficiently large, and the spectrum is monotonically decreasing,

i.e. $c_m < c_\ell$ iff $m > \ell$. (8.5) then rewrites to

$$\sum_{\ell \in L} c_\ell \sum_{\substack{m > \ell \\ m \in L}} (m - \ell) \left(1 - e^{-\frac{c_m}{B}}\right)^R < \varepsilon \sum_{\ell \geq 1} \ell c_\ell. \tag{8.6}$$

Assume now that the spectrum follows a power-law with large exponent, that is, $c_\ell = C \cdot \ell^{-a}$ for some $a > 2$. Note that under this assumption, the number of unique k-mers is $c_1 = C$, and the number of all k-mers is

$$\sum_{\ell \geq 1} \ell c_\ell = C \cdot \sum_{\ell \geq 1} \frac{1}{\ell^{a-1}} \leq C \cdot \frac{a-1}{a-2},$$

since $\zeta(s) = \sum_{i \geq 1} \frac{1}{i^s} \leq \frac{s}{s-1}$ for $s > 1$.

We then have the following result.

Theorem 1. *Given $0 < \varepsilon \leq 1$, if $B > C$ and R, B satisfy*

$$R \cdot \log \frac{B}{C} > \log \frac{1}{\varepsilon}, \tag{8.7}$$

then (8.6) holds.

Proof. Our goal is to estimate

$$\sum_{\ell \in L} c_\ell \sum_{\substack{m > \ell \\ m \in L}} (m - \ell) \left(1 - e^{-\frac{c_m}{B}}\right)^R, \tag{8.8}$$

where $c_\ell = C \cdot \ell^{-a}$. We assume $B > C$ and approximate $1 - e^{-\frac{c_m}{B}} \approx \frac{c_m}{B} = \frac{C}{B} m^{-a}$. We further lower-approximate (8.8) by replacing sums by integrals, thus obtaining

$$\int_1^\infty C \cdot \ell^{-a} \int_\ell^\infty (m - \ell) \left(\frac{C}{B} m^{-a}\right)^R dm \, d\ell. \tag{8.9}$$

Routine computation of the integral yields

$$C \left(\frac{C}{B}\right)^R \frac{1}{(aR-2)(aR-1)(aR+a+3)}. \tag{8.10}$$

The inequality of the Theorem becomes

$$\left(\frac{C}{B}\right)^R \frac{1}{(aR-2)(aR-1)(aR+a+3)} < \varepsilon \frac{a-1}{a-2}. \tag{8.11}$$

The Theorem follows. □

The theorem allows us to dimension the Set-Min sketch. For example, one can set $B = \alpha C$ for some constant $\alpha > 1$ and $R = \log_\alpha \frac{1}{\varepsilon}$.

8.2.2 Computing tighter sketch dimensions

Theorem 1 provides a way to dimension a Set-Min sketch, provided that the spectrum follows a power-law distribution with a sufficiently large parameter a. In order to validate these estimates experimentally, and, at the same time, obtain a tool for computing tighter values B and R for arbitrary spectra, we implemented a simple heuristic hill climbing algorithm to compute those values by directly solving equation 8.5. Algorithm 1, given below, starts with $R = 1$ and some initial value of B and then iteratively increments R

and recomputes (8.4) until equation (8.5) holds true. In the implementation, B is initially set to $1.44 \times c_{max}$, where c_{max} is the largest spectrum value. After such a value of R is found, the algorithm starts decrementing R while incrementing B to maintain the total space RB constant as long as (8.5) holds. The final R and B are thus the last which satisfied (8.5) in the decrementing loop. Note that, for both loops, there is a value of R for which the exit condition is satisfied. The rationale for this step is to have as small R as possible in order to reduce query time, while maintaining the total space constant.

Data: $\{c_\ell\}_{\ell \in L}$, $\|\boldsymbol{a}\|_1 = \sum_{\ell \in L} \ell c_\ell$, ε
Result: R and B
$R \leftarrow 1$;
$B \leftarrow 1.44 \times c_{max}$;
$T \leftarrow \varepsilon \|\boldsymbol{a}\|_1$;
$E \leftarrow \mathrm{E}(D)$ (computed by (8.4));
while $E > T$ **do**
  $R \leftarrow R + 1$;
  $E \leftarrow \mathrm{E}(D)$;
end
$M \leftarrow R \times B$;
while $E < T$ **do**
  $R \leftarrow R - 1$;
  $B \leftarrow \lceil \frac{M}{R} \rceil$;
  $E \leftarrow \mathrm{E}(D)$;
end
$R \leftarrow R + 1$;
$B \leftarrow \lceil \frac{M}{R} \rceil$;

Algorithm 1: Heuristic to compute R and B

8.3 Max-Min sketch

Now, consider a k-mer spectrum $H(\cdot)$ where $H(\ell_i)$ is the number of k-mers associated to label ℓ_i. If it follows a decreasing trend such that $H(c_{i+1}) \leq H(c_i) \forall i \in 1 \cdots |L|$, and we are in the static case (when values of keys are given once and never change afterwards) then Count-Min sketches can be modified to achieve better errors. Under this framework, the conservative update strategy seen in Section 5.1.3 can be further modified by defining updates as $A(i, h_i(p)) = \max\{A(i, h_i(p)), \ell\}$. For this reason, we call this variant of Count-Min sketch *Max-Min*.

In the general case, however, we use a variant when, instead of directly comparing count values, counts are ordered according to the support size. Whenever the k-mer spectrum is not strictly decreasing for increasing k-mer counts, the maximum count does not correspond to the one with the smallest support and a simple max operation would be incorrect. When this happens, updates are performed by keeping the label with the minimum number of k-mers in the k-mer spectrum, and a query returns the label with the smallest such number. This policy is akin to Set-Min's choice of returning the count with the minimum support in case of collisions and should be the default behavior of Max-Min when no previous knowledge about spectrum distribution is available.

In the static case, Max-Min sketch improves Count-Min without any computational overhead: it simply replaces addition by max in the update rule, somehow simulating the behavior of Set-Min sketch. As with the conservative update, one can check that estimates by Max-Min can only be over-estimates which, however, don't exceed estimates

by original Count-Min. Nevertheless, in Section 8.4 we will see how these improvements are not enough to beat our method.

8.4 Results

Comparisons are performed against Count-Min and MPHF implementations. In particular, we use `BBHash` [122] as our MPHF implementation of choice, with its supporting library as the only external dependency of our code, as Count-Min is implemented directly within `fress`. In addition to following the optimizations seen in Section 8.2, `fress` only stores indexes to a dictionary (array) of sets in each cell of the sketch, rather than full sets. This array of involved sets is stored in text format together with the spectrum of the count table in input. The current version of `fress` does not include any complex optimization, such as multi-threading or succinct representations of the final sketch matrix.

8.4.1 Data sets

We tested Set-Min on six different data sets of different size and complexity. Four of them are fully assembled genomes:

- SAI: Sakai strain of *Escherichia Coli* taken from [213] (NCBI accession number B000007),
- MNO: genome of *Drosophila melanogaster* from FlyBase[1],
- RAI: genome of *Gossypium Raimondii* [77] downloadable from AFproject[223],
- GRC: human reference genome assembly GRCh38[2].

The other two contain unassembled reads:

- USAI: Sakai strain at 5x coverage from again from the AFproject paper [223],
- SRR: low-coverage human data SRR622461 from the 1000 Genomes Project[3]

Table 8.1 summarizes the characteristics of each data set for each value of k in our analysis. Observe that, while the number of distinct k-mers is comparable to the total number of k-mers (data size), the number of distinct k-mer counts is small. This is in accordance with the power-law distribution discussed in Section 8.1.1.

8.4.2 Set-Min vs Count-Min sketch

Table 8.2 compares Set-Min sketch to Count-Min sketch. Dimensions R and B were computed using Algorithm 1 to ensure bound (8.5) to hold for $\varepsilon = 0.01$. Dimensions of Count-Min sketch were set to be the same. Value 1 is the most common count in all reported datasets, and it was not inserted into Count-Min sketch in order to make the comparison against Set-Min as fair as possible. Zero values are thus interpreted as the non-inserted count. For ease of comparison, column T reports the threshold $\varepsilon\|\boldsymbol{a}\|_1$ given to Algorithm 1. Columns E_s and E_c report the actual total sum of errors for Set-Min

[1] http://flybase.org

[2] ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCA/000/001/405/GCA_000001405.15_GRCh38/seqs_for_alignment_pipelines.ucsc_ids/GCA_000001405.15_GRCh38_no_alt_analysis_set.fna.gz

[3] ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR622/SRR622461/SRR622461_1.fastq.gz. Only the file SRR622461_1 is used in this study.

Table 8.1: Data sheet for the data sets used in our study. Columns T_k and D_k report the total number of k-mers and the number of distinct k-mers (in millions), respectively. D_c reports the number of distinct k-mer counts. C_k reports the number (in millions) of distinct k-mers with a count value different from the most common one (which is 1 in all reported cases).

Type	Name	T_k [M]	k	D_k [M]	D_c	C_k [M]
assembled	SAI	5.50	11	2.38	69	1.30
			15	5.23	42	0.16
			21	5.30	28	0.11
	MNO	143	15	101	883	15.51
			21	122	710	4.10
			27	124	605	4.00
			32	125	522	4.01
	RAI	727	15	251	1944	105.71
			21	546	1019	57.13
			27	604	699	45.21
			32	632	540	38.59
	GRC	2935	15	547	12718	370.16
			21	2327	10038	95.15
			27	2483	7946	73.99
			32	2567	6651	66.71
unassembled	USAI	25	11	3.06	239	2.76
			15	9.74	143	6.74
			21	10.0	97	6.52
	SRR	7500	15	676	22323	538.56
			21	3635	17211	1435.30
			27	3734	13157	1353.95
			32	3703	10643	1261.06

and Count-Min, respectively. In all reported cases $E_s < T$, as expected. The total error of Count-Min, E_c is, most of the time, one order of magnitude larger than E_s. For SRR with $k = 15$, it even exceeds the total number of k-mers $\|\boldsymbol{a}\|_1$ in the dataset.

The average error of Set-Min is, in most cases, very close to 1, which suggests that the overwhelming majority of collisions occur between successive counts such as 1 and 2 – the most abundant ones in the spectra considered here. The average error of Count-Min is bigger but of the same order of magnitude, except for small k and unassembled datasets. On the other hand, the fraction of k-mers producing an error is in striking contrast: in case of Set-Min, about only 1-3% of distinct k-mers produce an error, while for Count-Min, this fraction is much larger. This shows that Count-Min cannot be used when most of k-mer counts are expected to be retrieved precisely, for comparable sketch sizes.

8.4.3 Set-Min vs Max-Min sketch

Table 8.3 compares Set-Min with Max-Min – the optimized version of Count-Min discussed in Section 5.1.2. As expected from theoretical considerations, the performance in terms of average error and sum of errors is better for Max-Min than regular Count-Min, but worse than the one of Set-Min. The same behavior is observed for the number of k-mers having an erroneously estimated frequency. Therefore, Max-Min falls in-between Set-Min and

Count-Min, providing a simple and inexpensive practical method to enhance the latter, without reaching the accuracy of the former.

Altogether, Table 8.3 shows that the performance of Max-Min is closer to Count-Min than to Set-Min. This is because, by keeping the maximum element in each bucket, we are reducing each set of Set-Min to a single element opening the possibility of increased collisions by potentially sharing a given maximum element between unrelated k-mers. The intersection operation performed by Set-Min during query is thus strictly necessary, to guarantee the desired error bounds.

8.4.4 Set-Min sketch vs KMC output

Not surprisingly, Set-Min achieves better memory consumption than KMC in all our tests (columns M_{kmc} and M_s of Table 8.4). Values of R and B do not change from Table 8.2. Compression rate is variable: from a small factor to two orders of magnitude. The best compression is achieved for larger values of k and assembled genomes. The former is primarily explained by the decreasing number of distinct counts, due to the power-law behavior. As for the difference between assembled genomes and sequencing data, we will discuss it in more details in Section 8.4.6.

8.4.5 Set-Min sketch vs MPHFs

Table 8.4 also reports the space usage for `BBHash` to obtain the best memory-optimized hash functions. Column M_{bbhash} is the space (in bytes) required by the hash function only, while M_{bball} is the space required by the hash function plus the external array of frequencies.

As in the previous case, Set-Min sketch is more memory-efficient when k is large, taking about an order of magnitude less memory than a MPHF. For small values of k, `BBHash` takes slightly less space and, being exact, may therefore be the preferable choice. However, one should keep in mind that MPHF does not support updates, while a Set-Min sketch is updatable to a certain extent with new (k-mer,count) pairs, and also mergeable with another possibly redundant map.

The behavior of the unassembled datasets is of particular interest. Even for large k's, MPHF appears to be a better choice for this type of data. The causes of this phenomenon and possible solutions are discussed in Section 8.4.6.

8.4.6 Unassembled datasets

As seen in Table 8.4, for the unassembled datasets, Set-Min sketch does not seem to have an advantage in memory usage, even for large k's. We found that this is due to low-count k-mers, specifically to k-mers whose count does not exceed the sequencing coverage. It is known that for Illumina sequencing, sequencing errors produce a linear growth of the number of new distinct k-mers (for large k) depending on the coverage (see e.g. Figure 2(b) of [173]). Frequencies of these "erroneous" k-mers do not have the same statistical behavior as *bona fide* k-mers, in particular first spectrum values do not decay at the same rate as the rest of the spectrum. Figure 8.5 shows the spectrum of the unassembled SRR datasets. One can observe a slower decay behavior for a few first spectrum values. In this situation, additional rows are needed just to make the sketch able to distinguish, with required precision, between small frequency values. Note that in practice, distinguishing between small frequencies is often irrelevant. For example, many read assemblers simply discard low-frequency reads as a way to de-noise the data. In the case of Set-Min, it is possible to collapse together the first m columns of the spectrum

by assigning to all k-mers in this subset the same frequency. This would considerably reduce the sketch size. Formally, error guarantee (8.5) would not hold anymore, but most of newly introduced errors would be small (typically, equal to 1) and would occur for low counts only.

To check the above, we constructed a Set-Min sketch for SRR with dimensions $(R, B) = (4, 3310557)$, merging together the first five columns of the sorted spectrum and assigning count 5 to all merged k-mers. While the final sum of errors was well above the theoretical limit (10^9 against $7 \cdot 10^6$), the maximum and average error were respectively 55 and 2.8. In many applications this error level could be acceptable.

8.4.7 Time measurements

Construction time is reported in Figure 8.6. Set-Min sketches are generally faster to build than memory-optimized BBHash except for smaller values of k. However, similarly to Table 8.4, Set-Min sketch is at a disadvantage when the count value distribution is less skewed, such in the case of short k or for unassembled reads. For highly skewed data, Set-Min can be built faster than BBHash MPHFs, but is still more computationally demanding than Count-Min or Max-Min because of the additional operations required to create and update the sets of labels. Set-Min average query time performance is 50% slower than Count-Min (and by extension of Max-Min) and comparable to those of BBHash, when data is very skewed. Following the previous trend, Set-Min sketches appear to be the slowest method for small values of k and for unassembled reads.

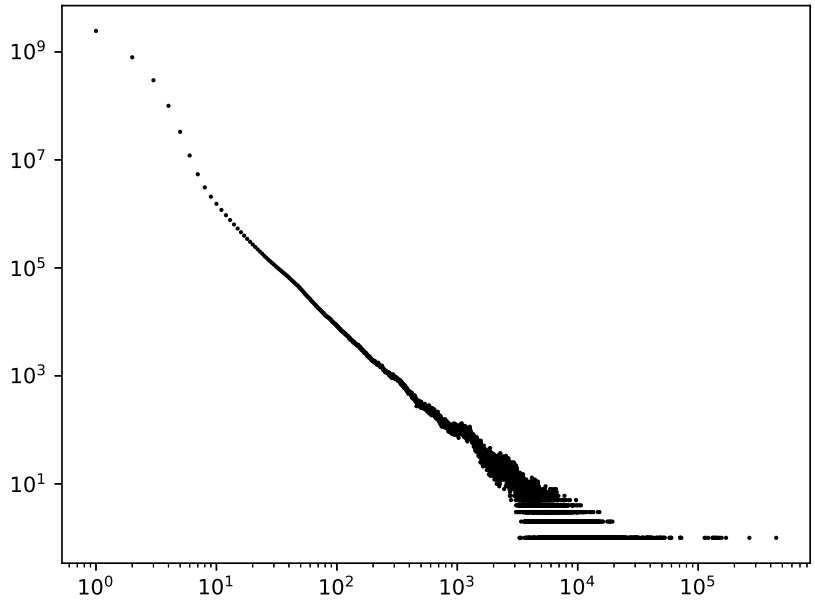


Figure 8.5: Spectrum in log-log scale of SRR unassembled data sets for $k = 32$. Note how, the number of k-mers appearing 2 times is not that smaller than the number of unique k-mers. In such a case, Set-Min sketch needs a lot of space just to distinguish count values of these two groups.

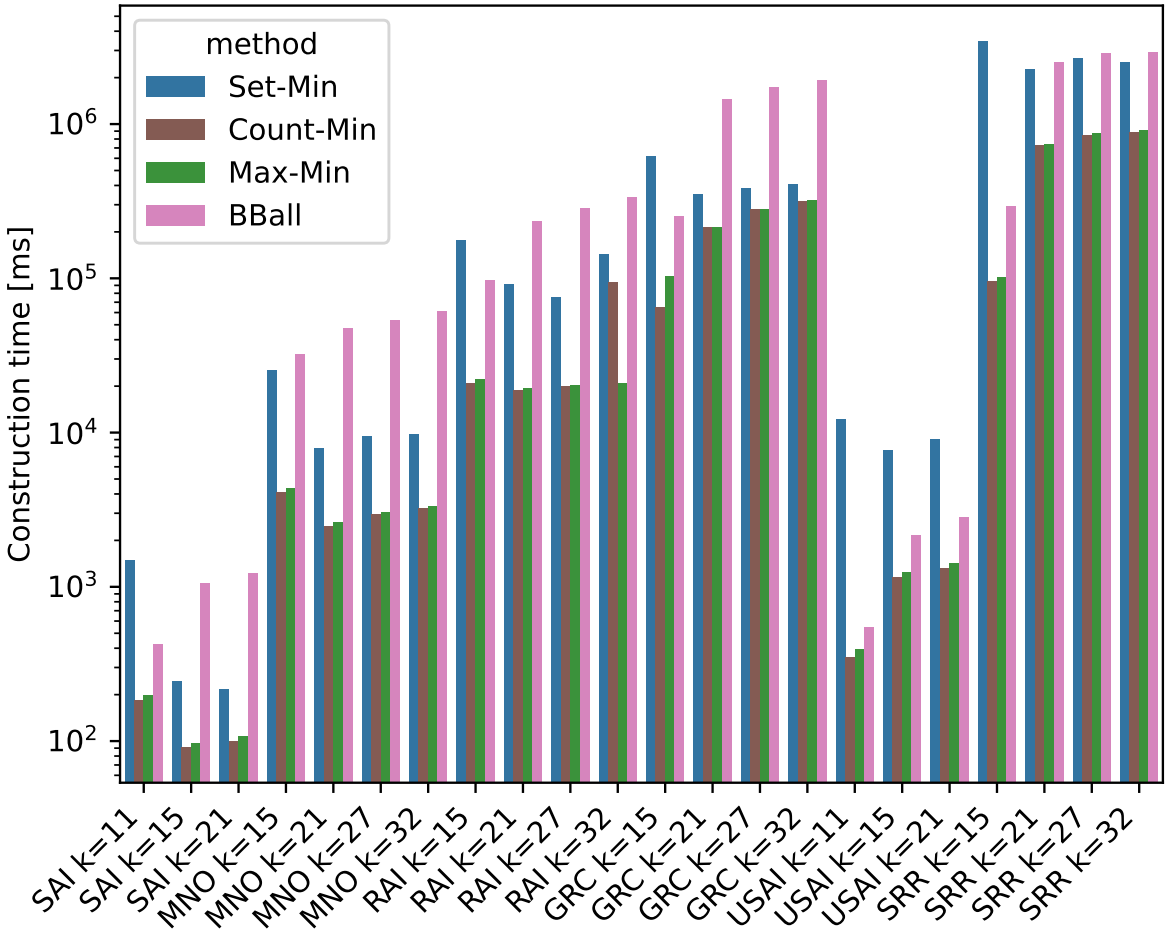


Figure 8.6: Construction time of Set-Min sketches compared to Count-Min, Max-Min and BBHash (with external array). Time is reported in milliseconds on a logarithm scale. Set-Min sketches tend to be slower than Count-Min or Max-Min sketches of comparable size due to the extra operations needed to manage sets. Compared to MPHFs, Set-Min sketches are generally faster when data is very skewed, which is not the case for unassembled datasets or small values of k.

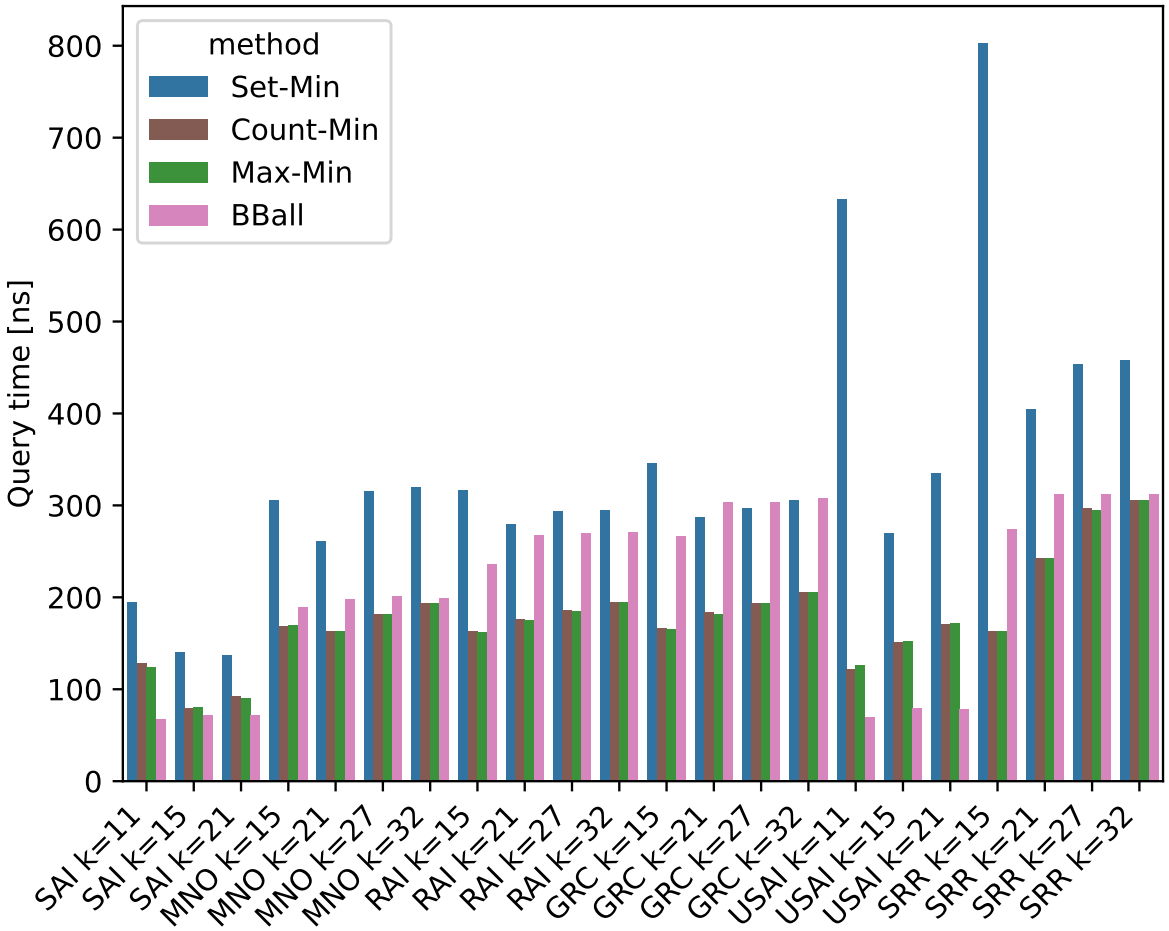


Figure 8.7: Average query time of Set-Min, Count-Min and BBHash. Similar to Figure 8.6 Set-Min sketches are generally slower than their Count-Min or Max-Min counter-parts. As before, unassembled datasets and small values of k prove to be the most difficult situations. On the other hand, for very skewed distributions Set-Min sketches perform as well as BBHash MPHFs.

Table 8.2: Set-Min compared to Count-Min. T is the reference upper bound on the sum of errors equal to $\varepsilon\|\boldsymbol{a}\|_1$ (right-hand side of (8.5)). E_s and E_c are the sum of errors for Set-Min and Count-Min respectively. N_s and N_c are the percentages (rounded to integers) of distinct k-mers producing an error, for Set-Min and Count-Min, respectively. A_s and A_c are respective average errors, with average taken over the number of distinct k-mers resulting in an error in the respective sketch.

Name	k	R	B	T	E_s	E_c	N_s	N_c	A_s	A_c
SAI	11	4	$1.04 \cdot 10^6$	$5.50 \cdot 10^4$	$5.00 \cdot 10^4$	$1.39 \cdot 10^6$	1.8	26	1.15	2.24
SAI	15	5	$2.13 \cdot 10^5$	$5.50 \cdot 10^4$	$4.66 \cdot 10^4$	$2.38 \cdot 10^5$	0.9	4	1.01	1.1
SAI	21	5	$1.20 \cdot 10^5$	$5.50 \cdot 10^4$	$5.29 \cdot 10^4$	$4.57 \cdot 10^5$	0.9	7	1.05	1.18
USAI	11	5	$4.58 \cdot 10^5$	$2.57 \cdot 10^5$	$1.99 \cdot 10^5$	$7.44 \cdot 10^7$	2.6	99	2.46	24.6
USAI	15	4	$4.79 \cdot 10^6$	$2.49 \cdot 10^5$	$2.45 \cdot 10^5$	$8.01 \cdot 10^6$	1.9	33	1.31	2.53
USAI	21	5	$3.99 \cdot 10^6$	$2.38 \cdot 10^5$	$2.00 \cdot 10^5$	$7.91 \cdot 10^6$	1.6	34	1.21	2.35
MNO	15	5	$1.79 \cdot 10^7$	$1.43 \cdot 10^6$	$1.39 \cdot 10^6$	$8.35 \cdot 10^6$	1.4	7	1	1.25
MNO	21	4	$4.69 \cdot 10^6$	$1.43 \cdot 10^6$	$1.41 \cdot 10^6$	$2.26 \cdot 10^7$	1.1	12	1.03	1.61
MNO	27	5	$3.72 \cdot 10^6$	$1.43 \cdot 10^6$	$1.11 \cdot 10^6$	$2.28 \cdot 10^7$	0.9	12	1.01	1.48
MNO	32	5	$3.81 \cdot 10^6$	$1.42 \cdot 10^6$	$1.11 \cdot 10^6$	$2.10 \cdot 10^7$	0.9	12	1.01	1.44
RAI	15	4	$8.36 \cdot 10^7$	$7.27 \cdot 10^6$	$5.65 \cdot 10^6$	$1.50 \cdot 10^8$	2.1	27	1.08	2.25
RAI	21	4	$8.47 \cdot 10^7$	$7.27 \cdot 10^6$	$5.73 \cdot 10^6$	$4.09 \cdot 10^7$	1	6	1.01	1.3
RAI	27	4	$7.14 \cdot 10^7$	$7.27 \cdot 10^6$	$6.49 \cdot 10^6$	$3.56 \cdot 10^7$	1.1	5	1.01	1.22
RAI	32	4	$6.38 \cdot 10^7$	$7.27 \cdot 10^6$	$6.87 \cdot 10^6$	$3.15 \cdot 10^7$	1.1	4	1.01	1.17
GRC	15	3	$2.26 \cdot 10^8$	$2.93 \cdot 10^7$	$2.78 \cdot 10^7$	$1.36 \cdot 10^9$	2.9	52	1.78	4.74
GRC	21	4	$1.42 \cdot 10^8$	$2.93 \cdot 10^7$	$2.60 \cdot 10^7$	$1.65 \cdot 10^8$	1.1	6	1.01	1.23
GRC	27	4	$1.07 \cdot 10^8$	$2.93 \cdot 10^7$	$2.82 \cdot 10^7$	$1.91 \cdot 10^8$	1.1	6	1.01	1.25
GRC	32	4	$9.84 \cdot 10^7$	$2.93 \cdot 10^7$	$2.92 \cdot 10^7$	$1.85 \cdot 10^8$	1.1	6	1.01	1.22
SRR	15	4	$1.17 \cdot 10^8$	$7.90 \cdot 10^7$	$4.93 \cdot 10^7$	$1.34 \cdot 10^{10}$	3.3	96	2.2	20.68
SRR	21	3	$2.39 \cdot 10^9$	$7.35 \cdot 10^7$	$7.09 \cdot 10^7$	$5.63 \cdot 10^8$	1.8	9	1.11	1.68
SRR	27	4	$1.78 \cdot 10^9$	$6.80 \cdot 10^7$	$4.92 \cdot 10^7$	$4.58 \cdot 10^8$	1.3	8	1.04	1.53
SRR	32	4	$1.72 \cdot 10^9$	$6.34 \cdot 10^7$	$4.95 \cdot 10^7$	$4.01 \cdot 10^8$	1.3	7	1.03	1.48

Table 8.3: Set-Min compared to Max-Min sketch. Columns E_c, N_c, A_c are replaced by E_m, N_m, A_m with the same meaning as their Table 8.2 counterparts.

Name	k	T	E_s	E_m	N_s	N_m	A_s	A_m
SAI	11	$5.50 \cdot 10^4$	$5.00 \cdot 10^4$	$4.15 \cdot 10^5$	1.8	13.5	1.15	1.29
SAI	15	$5.50 \cdot 10^4$	$4.66 \cdot 10^4$	$2.13 \cdot 10^5$	0.9	4	1.01	1.01
SAI	21	$5.50 \cdot 10^4$	$5.29 \cdot 10^4$	$4.02 \cdot 10^5$	0.9	7.2	1.05	1.06
USAI	11	$2.57 \cdot 10^5$	$1.99 \cdot 10^5$	$1.24 \cdot 10^7$	2.6	63.5	2.46	6.4
USAI	15	$2.49 \cdot 10^5$	$2.45 \cdot 10^5$	$1.72 \cdot 10^6$	1.9	13.3	1.31	1.33
USAI	21	$2.38 \cdot 10^5$	$2.00 \cdot 10^5$	$1.76 \cdot 10^6$	1.6	14.2	1.21	1.24
MNO	15	$1.43 \cdot 10^6$	$1.39 \cdot 10^6$	$5.74 \cdot 10^6$	1.4	5.6	1	1.02
MNO	21	$1.43 \cdot 10^6$	$1.41 \cdot 10^6$	$1.78 \cdot 10^7$	1.1	11.2	1.03	1.31
MNO	27	$1.43 \cdot 10^6$	$1.11 \cdot 10^6$	$1.78 \cdot 10^7$	0.9	12.1	1.01	1.2
MNO	32	$1.42 \cdot 10^6$	$1.11 \cdot 10^6$	$1.65 \cdot 10^7$	0.9	11.3	1.01	1.17
RAI	15	$7.27 \cdot 10^6$	$5.65 \cdot 10^6$	$5.81 \cdot 10^7$	2.1	16.8	1.08	1.38
RAI	21	$7.27 \cdot 10^6$	$5.73 \cdot 10^6$	$3.05 \cdot 10^7$	1	5.2	1.01	1.07
RAI	27	$7.27 \cdot 10^6$	$6.49 \cdot 10^6$	$2.84 \cdot 10^7$	1.1	4.5	1.01	1.05
RAI	32	$7.27 \cdot 10^6$	$6.87 \cdot 10^6$	$2.61 \cdot 10^7$	1.1	4	1.01	1.03
GRC	15	$2.93 \cdot 10^7$	$2.78 \cdot 10^7$	$4.07 \cdot 10^8$	2.9	27.8	1.78	2.67
GRC	21	$2.93 \cdot 10^7$	$2.60 \cdot 10^7$	$1.38 \cdot 10^8$	1.1	5.5	1.01	1.08
GRC	27	$2.93 \cdot 10^7$	$2.82 \cdot 10^7$	$1.62 \cdot 10^8$	1.1	6	1.01	1.09
GRC	32	$2.93 \cdot 10^7$	$2.92 \cdot 10^7$	$1.58 \cdot 10^8$	1.1	5.7	1.01	1.08
SRR	15	$7.90 \cdot 10^7$	$4.93 \cdot 10^7$	$3.31 \cdot 10^9$	3.3	61.8	2.2	7.92
SRR	21	$7.35 \cdot 10^7$	$7.09 \cdot 10^7$	$2.42 \cdot 10^8$	1.8	5.9	1.11	1.13
SRR	27	$6.80 \cdot 10^7$	$4.92 \cdot 10^7$	$2.05 \cdot 10^8$	1.3	5.2	1.04	1.06
SRR	32	$6.34 \cdot 10^7$	$4.95 \cdot 10^7$	$1.89 \cdot 10^8$	1.3	4.9	1.03	1.04

Table 8.4: Set-Min ($\epsilon = 0.01$) compared to KMC and BBHash (run with $\gamma = 1$). All memory is reported in bytes. Column M_{kmc}, M_s, M_{bball} are the memory taken by a fully functional map between k-mers and their frequencies when applying KMC, Set-Min sketch and `BBHash`, respectively. M_{bbhash} is the memory of the hash function produced by `BBHash` without the external array of frequencies.

Name	k	M_{kmc}	M_s	M_{bbhash}	M_{bball}
SAI	11	$1.21 \cdot 10^7$	$5.75 \cdot 10^6$	$9.13 \cdot 10^5$	$3.00 \cdot 10^6$
SAI	15	$3.80 \cdot 10^7$	$1.20 \cdot 10^6$	$2.06 \cdot 10^6$	$5.99 \cdot 10^6$
SAI	21	$4.77 \cdot 10^7$	$6.77 \cdot 10^5$	$2.03 \cdot 10^6$	$6.00 \cdot 10^6$
USAI	11	$1.54 \cdot 10^7$	$1.49 \cdot 10^7$	$1.17 \cdot 10^6$	$4.61 \cdot 10^6$
USAI	15	$6.95 \cdot 10^7$	$2.87 \cdot 10^7$	$3.72 \cdot 10^6$	$1.35 \cdot 10^7$
USAI	21	$8.53 \cdot 10^7$	$3.00 \cdot 10^7$	$3.91 \cdot 10^6$	$1.39 \cdot 10^7$
MNO	15	$7.08 \cdot 10^8$	$1.68 \cdot 10^8$	$3.87 \cdot 10^7$	$2.15 \cdot 10^8$
MNO	21	$9.80 \cdot 10^8$	$3.55 \cdot 10^7$	$4.66 \cdot 10^7$	$2.45 \cdot 10^8$
MNO	27	$1.24 \cdot 10^9$	$3.75 \cdot 10^7$	$4.73 \cdot 10^7$	$2.48 \cdot 10^8$
MNO	32	$1.37 \cdot 10^9$	$3.84 \cdot 10^7$	$4.90 \cdot 10^7$	$2.36 \cdot 10^8$
RAI	15	$1.76 \cdot 10^9$	$7.54 \cdot 10^8$	$9.59 \cdot 10^7$	$5.98 \cdot 10^8$
RAI	21	$4.37 \cdot 10^9$	$6.78 \cdot 10^8$	$2.16 \cdot 10^8$	$1.24 \cdot 10^9$
RAI	27	$6.04 \cdot 10^9$	$5.36 \cdot 10^8$	$2.37 \cdot 10^8$	$1.29 \cdot 10^9$
RAI	32	$6.96 \cdot 10^9$	$4.79 \cdot 10^8$	$2.70 \cdot 10^8$	$1.38 \cdot 10^9$
GRC	15	$3.83 \cdot 10^9$	$1.70 \cdot 10^9$	$2.09 \cdot 10^8$	$1.65 \cdot 10^9$
GRC	21	$1.86 \cdot 10^{10}$	$1.28 \cdot 10^9$	$9.32 \cdot 10^8$	$6.46 \cdot 10^9$
GRC	27	$2.48 \cdot 10^{10}$	$9.66 \cdot 10^8$	$9.86 \cdot 10^8$	$6.57 \cdot 10^9$
GRC	32	$2.82 \cdot 10^{10}$	$8.38 \cdot 10^8$	$1.08 \cdot 10^9$	$6.54 \cdot 10^9$
SRR	15	$4.73 \cdot 10^9$	$2.00 \cdot 10^9$	$2.59 \cdot 10^8$	$2.12 \cdot 10^9$
SRR	21	$2.91 \cdot 10^{10}$	$1.61 \cdot 10^{10}$	$1.48 \cdot 10^9$	$1.10 \cdot 10^{10}$
SRR	27	$3.73 \cdot 10^{10}$	$1.60 \cdot 10^{10}$	$1.48 \cdot 10^9$	$1.08 \cdot 10^{10}$
SRR	32	$4.07 \cdot 10^{10}$	$1.46 \cdot 10^{10}$	$1.57 \cdot 10^9$	$1.04 \cdot 10^{10}$

Chapter 9

Discussion

We presented Set-Min sketch – a novel sketching method inspired by the Count-Min sketch, whose primary use is to associate keys to labels without explicitly storing the former. We demonstrated its advantages for storing k-mer counts information when the distribution of labels (k-mer counts) follows a power-law distribution. Under this assumption, we proposed simple bounds for a Set-Min sketch that guarantee the total error sum to be within an ε fraction of the total number of k-mers in the dataset. We showed that the probabilistic compression provided by Set-Min sketches allows for better memory usage compared to the raw output of the popular KMC k-mer counting tool when applied to labels following a skewed distribution, at the price of a very modest error rate. Space savings are especially remarkable in case of whole-genome data and large values of k, where they can reach two orders of magnitude reductions in memory usage. Set-Min has been shown to be more space efficient than the MPHF-based solution for large values of k. For smaller k's, however, MPHFs provide an implementation with comparable memory consumption.

Set-Min sketches are easy to implement and fast to build thanks to the same matrix layout of Count-Min sketches. The only true complex operations that need extra attention are insertions into each cell of the sketch. Luckily, overall space can be further reduced by implementing one of the many optimizations introduced earlier: replacement of cell sets with indices to involved sets or ignoring the most common count value if the set of possible k-mers is known in advance. Count sketches provide a good trade-off between algorithmic complexity, space, construction/query times and errors, with Set-Min optimized for medium-long term storage of counts. Finally, Set-Min sketch achieves better point-query errors than both Count-Min and Max-Min sketches of comparable dimensions, thanks to the distribution-aware dimensioning performed on the k-mer spectrum.

Another application of Set-Min sketches, not explored here, is to act as a temporary representation while building more complex structures based on counters. Consider, for example, the exact computation of weighted pairwise distances between all pairs of genomes in a given set. Examples of such distances are the Bray-Curtis similarity measure, see e.g. [13], or Weighted Jaccard similarity estimation [85]. The most naive algorithm is to first process each sequence independently, generating one count tables for each, and then compare tables pair-wisely to produce the desired output. Instead of storing whole tables, one can store multiple Set-Min sketches together with a presence-absence data structure, such as a Bloom filter. By doing so, the weighted comparison computation is reduced to a single pass through the presence-absence data structure with the counters of a given k-mer retrieved on-demand from the Set-Min sketches of the datasets in which the k-mer is found. Yet another possible application is sharing counter information between different computational units in a distributed setting, where a server oversees multiple less powerful machines. All nodes have access to the same genomic representation (say,

a set of contigs), but only the server can efficiently perform k-mer counting, while the smaller machines have the task of processing incoming data based on the counts. In this case, the server could send a Set-Min sketch to all its subordinates.

One further feature of Set-Min sketch is its mergeability from redundant maps. A large map can be split into m sub-maps without the restriction of having disjoint sets of keys. Even if some maps have redundant information, i.e. share common (key, value) pairs, the Set-Min sketch built by cell-wise union of the m sketches will be equivalent to the sketch built from the whole original map. Count-Min sketches do not have this property, but instead they are mergeable when constituent maps should be "added up". In case of redundancy, Count-Min will simply count each repeated item multiple times. In this respect, Set-Min and Count-Min sketches may have complementary uses.

As introduced in Section 8.1, in this work we assumed that only k-mers present in the dataset can be queried. This assumption allowed us to discard the largest value of the spectrum corresponding to unique k-mers, thereby saving space. Set-Min sketches can seamlessly work without this assumption, but the space required for storing k-mer counters may not be competitive to other solutions. An alternative could be to build an additional data structure, such as a Bloom filter, representing presence-absence information for the set of k-mers having the largest count. This allows the discrimination between k-mers absent in the dataset from those present but non-represented in the sketch.

Another scenario occurs when working with multiple datasets of very high similarity, such as large collections of bacterial strains or collections of RNA-seq data [195, 215]. In this case, it might be beneficial to build a Bloom filter for the k-mers present in the union of the datasets, and maintain multiple Set-Min sketches to represent k-mer counts in each dataset.

Note that Set-Min sketches can also be helpful for long-term storage and transmission of the k-mer composition of a dataset augmented with count information. The k-mers of the dataset can be reassembled into simplitigs [29] with a Set-Min sketch storing the (approximated) frequencies. The full count table can be restored from the simplitigs and the sketch.

Part III

Space-efficient representation of genomic k-mer count tables

Chapter 10

Context and motivation

10.1 Problem statement

The previously introduced Set-Min sketching technique allows for efficient storage of very skewed distributions of k-mer counts. However, in some applications even small errors might be undesirable. For example, it could be useful to separate unique k-mers from the others in order to find unambiguous seeds or correct reads [121]. Finding unique k-mers exactly with Set-Min sketches is not possible, since errors primarily come from mixing together the most common frequencies which, in practice, are usually the lowest ones [44]. More generally, information about k-mer counts is increasingly used in other applications too [193, 163, 100, 132, 95, 96, 143], which can benefit from (exact) space-efficient solutions. Furthermore, assigning to each k-mer its frequency in a dataset is just one particular application of mapping k-mers to numerical values. To this end, mapping k-mers to indices of an array is an effective solution which makes possible any type of annotation.

Minimal Perfect Hash Functions (MPHFs for short) implement such an approach [144, 214, 57] and are already extensively applied as building blocks in bioinformatics solutions [122, 215]. Having two independent data structures allows for more aggressive space optimizations. For example, the original sequence dataset can be used as the primary source of k-mers while a random-access data structure will then allow retrieving their counts efficiently. A MPHF bijectively maps each item from a set S to an index in the range $[0, |S| - 1]$. Any additional information can then be stored in an array indexed by the values returned by the MPHF. The idea works well as long as the array stores a number of distinct elements close to $|S|$, but it can be suboptimal for count values non-uniformly distributed, which is the case for k-mer counts. Because of this, the multiset of k-mer counts will typically have a fairly low empirical zero-order entropy, and it could be effectively compressed to save further space. However, standard compression algorithms do not maintain random access to the compressed contents limiting this solution to be a long-term storage option only. Furthermore, MPHFs themselves encompass a non-negligible space overhead which must be added to the space of the array storing the counters. In case of BBHash [122] this surplus is around 3 bits/key, whereas the theoretical minimum is 1.44.

An alternative to MPHFs are the so-called Static Functions [11, 67] which encode values together with their hash function into a single structure. In particular, Compressed Static Functions (CSFs), as their name suggests, are Static Functions whose space approaches the number of bits defined by the empirical entropy of the stored values. This feature makes them particularly useful for representing different k-mer annotations, such as counts or presence information across sequences of a given sample [132, 95, 96, 143]. Both MPHFs and CSFs do not deal with k-mers themselves (as it was the case for sketch-

ing), and require appropriate additional structures in order to answer membership queries. This is not restrictive, as having different, specialized methods allows for greater flexibility, otherwise unattainable in a monolithic setting.

Further memory reductions are still possible despite the advantage of CSFs over MPHFs for skewed input count distributions if k-mer similarity is taken into account. Recent works [132] have suggested that similar k-mers often share similar count values. In fully assembled genomes, for example, repetitive k-mers are often the result of long duplications. All k-mers involved in such events are thus more likely to have increased copy numbers which end-up clustered together. Hence, for k big enough, Locality-Sensitive Hashing based on k-mer similarity is also able to cluster together similar counts.

The following chapters focus on the development of novel techniques based on Compressed Static Functions as better representations of k-mer counts. Compared to Set-Min sketch, our new methods are targeted to EXACT count storage with optional extension to the approximate case. Like before, we do not explicitly take into account k-mers, limiting our structures to be collision-free only on the set they were built for. Our contributions can be summarized as the following:

1. An extended CSF implementation achieving smaller memory than current solutions for very skewed distributions (Section 11.2.1).

2. The first algorithm combining item 1 with minimizer-based bucketing of weights (Section 11.2.3).

3. An alternative version of item 2.

We report our analysis and results in Section 11.3.

10.2 Related work

Despite ongoing efforts to reduce overall space usage [149, 189] of classic k-mer counters [135, 165, 101, 181] they remain tailored toward dynamic representations of count tables and are ill-suited as long-term storage options due to the computational overhead needed to support updates. Further optimizations become possible if count tables are considered to be static [133]. Under this hypothesis, fully queryable static count tables can be equivalently seen as a combination of three different components:

1. A presence/absence data structure telling whether k-mers are in the table or not.

2. A map data structure mapping each k-mer to its frequency.

3. An efficient representation for count values.

Such framework is the foundation of algorithms representing k-mer counts across multiple datasets such as [132, 98]. Presence/absence information can be retrieved using variations of de Bruijn graphs, but other methods are possible, e.g. by using simplitigs (spectrum-preserving string sets) [29, 162] or similar methods [161, 179] combined with indexing for quickly locating the positions of k-mers. If exact queries are not needed, Approximate Membership Queries (AMQ) data structures can be used instead [17, 59].

Techniques for representing counts coming from multiple datasets fall short when applied to single k-mer sets, since it is an application they were not designed for. We hereby focus on developing efficient count representations under the aforementioned framework for single datasets only. Similarly to what we did for Set-Min sketch we limit ourselves to retrieve counts of k-mers that are known to be present in the dataset, leaving presence/absence queries to other data structures.

10.3 Contributions

The following chapters present new data structures based on Compressed Static Functions for storing genomic k-mer count tables using the smallest possible space. Section 11.2.1 demonstrates how to extend the only practical implementation of CSFs [67] with Bloom filters (see Section 5.1) in order to break its intrinsic lower-bound of 1 bit per element allowing to efficiently represent multisets of counts of very low entropy. The resulting Bloom-enhanced CSFs (BCSFs) are then used as building blocks in Chapter 11 in combination with minimizer bucketing to obtain two additional exact algorithms: AMB and FIL. Both algorithms take advantage of the fact that similar k-mers tend to have identical (or similar) counts [132] by bucketing together count values of k-mers with the same minimizer. A similar idea is used by some k-mer counting algorithms [165, 101, 109] with the difference that in our case buckets contain counts rather than the k-mers themselves. By choosing a representative value for each bucket, we obtain a "bucket table" that we encode using Bloom-enhanced CSF.

Results for both AMB and FIL are reported in Section 11.3 demonstrating their utility for both low and high entropy datasets. Space and time comparisons clearly show how the proposed methods are able to represent count tables efficiently, while still being able to query the resulting data structures. In particular, for large enough k (and large enough minimizers lengths), we are able to compress count values in *less space than their empirical entropy* while retaining fast query times. To the best of our knowledge, this is the first implementation proposing such a compact representation. In addition to these exact representations, the natural extension of our algorithm AMB to the approximate case is presented in Section 11.2.6 with additional space savings obtained by allowing a predefined absolute error over queries.

We collectively call our implementations "Locom" which can be found at `https://github.com/yhhshb/locom`.

Chapter 11

Locom: minimizers meet Compressed Static Functions

Unlike Set-Min sketches (Chapter II), Locom is able to provide exact representations, and it is capable of dealing with counts not necessarily power-law distributed. The wider applicability range stems from the smart combination of minimizer bucketing, Bloom Filters and Compressed Static Functions producing, in some cases, compressed representations smaller than the empirical entropy of the counts they are storing. The intuition behind our methods is presented in Section 11.1. We follow by presenting two different methods for compressing counts, AMB in Section 11.2.3 and FIL in Section 11.2.4. Finally, Section 11.3 presents our results.

11.1 Key algorithmic ideas

For ease of explanation, throughout the rest of this work dedicated to Locom, we will consider k-mer count tables as associative arrays f, mapping a set of k-mers K, considered static, to their counts, i.e. number of occurrences in a given dataset. As before, $\|f\|_1$ stands for the L1-norm of f, that is $\sum_{q \in K} f(q)$.

11.1.1 Correlation of neighboring k-mer counts

Decomposing a sequence into its constituent k-mers is a required step in many bioinformatics analyses. For appropriately long k's, k-mers can be used for sequence similarity computation replacing classical string-based approaches. Moreover, by storing k-mers into fast data structures such as hash tables, algorithms that work on bags of words are usually faster and easier to understand and to implement. Sometimes, however, k-mers alone are not sufficient to capture the complexity of the original sequences. One example of such situation are genomes with high numbers of duplications, which cannot be represented by simple sets. In this case, counting information nicely complements k-mers with additional structural information about the original sequences. For example, k-mers with very high copy numbers come, most of the time, from very low-complexity zones spread throughout genomes, and are often removed from indexes by alignment algorithms. Thus, k-mer counting can provide invaluable information about sequence structure.

As we will empirically demonstrate with our results, this relation between genomic content and k-mer counts also weakly applies in the opposite direction. That is, similar k-mers are likely to have similar counts. Possible sources of similarity are:

- Skewed distributions. All counts are similar because one value is more likely than others (see Figure 11.1). For very high skews, k-mer similarity becomes less important, since very different k-mers start to have the same count values.

- Repeats. Repeated blocks increase the counter of all their k-mers together as depicted in Figure 11.2.
- Mutations. Changing one base likely generates k unique neighboring k-mers (see Figure 11.3).

Figure 11.1: High correlation between neighboring k-mer counts can be due to their high skewed distribution without any particular relation between k-mers. In the reported case $k = 7$ with all k-mers unique.

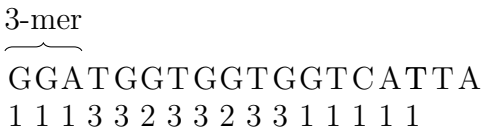


Figure 11.2: Example of count correlation due to repetitions. Substring TGG repeats 3 times generating a block of relatively high counts very similar to one another. Non-consecutive duplications have the same effect.

Figure 11.3: Example showing the effect of a single mutation when coverage is > 1. The first sequence is the original one, of coverage 20. One of its 20 copies contain a single mutated letter (highlighted in red) which generates a block of 4 additional unique 4-mers in the mutated sequence and a block of 4 4-mers of frequency 19 from the original sequence. Since the affected count values in both sequences are somehow linked to some particular k-mers, LSH techniques looking at k-mer composition are a viable option to bucket counts together. However, collisions between different count values are possible, and should be dealt with.

The above examples only refer to counting fully assembled genomes, but the property holds true for other types of datasets too. For example, in [132] counts obtained from transcriptomes are compressed by replacing blocks of similar values by their average, without detrimental effects on downstream analysis.

11.1.2 Minimizers as a context-aware bucketing technique of k-mers

Minimizers are a popular technique used in different applications involving k-mer analysis. The use of minimizers for biosequence analysis goes back to [167], whereas a similar concept, named *winnowing*, was earlier applied in [178] to document search. However,

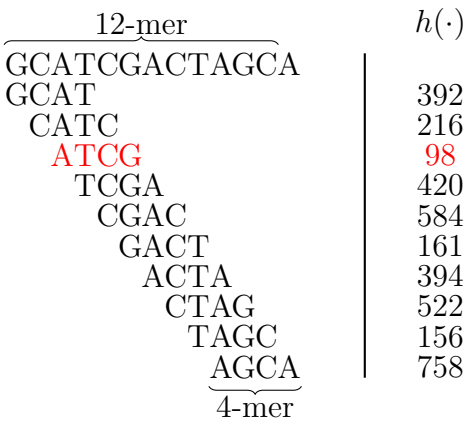


Figure 11.4: Example of minimizers used as k-mer fingerprints. The same minimizer is likely to be shared by multiple neighboring k-mers. In the example above $k = 12$ and $m = 4$. Column $h(\cdot)$ indicate the hash value of each substring of length 4. The minimizer of two successive 12-mers is highlighted in red.

both papers define minimizers over windows of w consecutive k-mers with a minimizer being the minimum k-mer of a window. This original definition of minimizers have been extensively applied to various data-intensive sequence analysis problems in bioinformatics, such as metagenomics (KRAKEN [207]) or minimizing cache misses in k-mer counting (KMC [101]), or mapping and assembling long single-molecule reads [113, 114]. Space reduction comes from the fact that successive windows often share the same minimum with minimizer indexes subsets of the k-mer sets they originate from. Recently, there has been a series of works on both theoretical and practical aspects of designing efficient minimizers for aligning sequences, see e.g. [222, 54].

Here, instead, we define minimizers following the definition given in [165, 101]. Given a k-mer q of length k, its minimizer of length m, with $m \leq k$, is the smallest substring of q of length m w.r.t. some order defined on m-mers. Standard practice is to use a non-cryptographic hash function to hash m-mers and take the one with the minimum hash value as minimizer. Note that the lexicographic ordering has been shown to have poor statistical properties [167]. The choice of hash function is not important as long as it has good statistical guarantees (randomness and uniformity). The guiding idea is that a minimizer can be considered as a "footprint" (hash value) of a corresponding k-mer so that similar (e.g. neighboring in the genome) k-mers are likely to have the same minimizer, see Figure 11.4. In this case, minimizers can be seen as a specific instance of locality-sensitive hashing, in particular of MinHash sketching [26].

The same algorithm to find "windowed" minimizers can be reused, without modifications, to bucket k-mers, by simply setting window size equal to k and minimizer size equal to m, whereas the original minimizer definition targeted sampling.

11.2 Adapting Compressed Static Functions to k-mer count tables

11.2.1 Bloom-enhanced Compressed Static Functions

As mentioned earlier (Section 6.2.2), the Compressed Static Functions (CSF) of [67] do not properly deal with datasets generated by low-entropy distributions (when entropy is smaller than 1). This case occurs when datasets have a dominant value representing a large fraction (say, more than a half) of all values. This is typically the case with genomic

k-mer count data, and whole-genome data in particular, where a very large fraction of k-mers occur just once. For example, in *E.Coli* genomes (≈5.5Mbp), about 97% of all distinct 15-mers occur once, whereas only the remaining 3% occur twice or more. For such datasets, the method of [67] does not approximate well the empirical entropy, as it cannot achieve less than 1 bit per key. Our technique to circumvent this deficiency and compress close to the empirical entropy is to augment CSFs of [67] with Bloom filters (Section 5.1.5). We start by building a Bloom filter for all k-mers whose value is not the dominant one, and then we construct a CSF on all positives (i.e. true and false positives) of this filter. At query time, we first check the query k-mer against the Bloom filter and, if the answer is positive, recover its value from the CSF.

Formally, let K_0 be the k-mers with the most common frequency. Let $|K_0| = \alpha|K|$. Assume that our Bloom filter implementation takes $C_{BF} \log \frac{1}{\varepsilon}$ bits per key and our CSF implementation takes C_{CSF} bits per key. For the purpose of explanation, we will specify both C_{BF} and C_{CSF} at the end of this section.

We store keys $K \setminus K_0$ in a Bloom filter B and build a CSF for $(K \setminus K_0) \cup FP_B(K_0)$. The total space is

$$C_{BF}(1-\alpha)|K| \log \frac{1}{\varepsilon} + C_{CSF}|K|((1-\alpha) + \varepsilon\alpha). \tag{11.1}$$

The Bloom filter enables space-saving only if α is sufficiently large. To decide if we need a Bloom filter, we have to verify if the inequality

$$C_{BF}(1-\alpha)|K| \log \frac{1}{\varepsilon} + C_{CSF}|K|((1-\alpha) + \varepsilon\alpha) < C_{CSF}|K|. \tag{11.2}$$

holds for some $\varepsilon < 1$. Note again that C_{CSF} on the left and right sides are not exactly the same in reality, however assuming them the same is not reductive because of specificities of the CSF implementation we use. We will elaborate further on this later on. Then (11.2) rewrites to

$$\frac{C_{BF}}{C_{CSF}} \frac{1-\alpha}{\alpha} \log \frac{1}{\varepsilon} + \varepsilon < 1. \tag{11.3}$$

Using simple calculus, we obtain that if $\frac{C_{BF}}{C_{CSF}} \frac{1-\alpha}{\alpha} > \ln 2$ (that is, $\frac{C_{BF}}{C_{CSF}} \frac{1-\alpha}{\alpha} \log e > 1$), then (11.3) never holds for $0 < \varepsilon < 1$. The left-hand side of (11.3) reaches its minimum for

$$\varepsilon_0 = \frac{C_{BF}}{C_{CSF}} \frac{1-\alpha}{\alpha} \log e, \tag{11.4}$$

and this minimum is smaller than 1 if $\varepsilon_0 < 1$. We conclude that in order to decide if a Bloom filter enables space-saving, we have to check the value ε_0. If $\varepsilon_0 \geq 1$, we do not need a Bloom filter, otherwise we need one with $\varepsilon = \varepsilon_0$. This shows that a Bloom filter is needed whenever

$$\alpha > \frac{C_{BF} \log e}{C_{CSF} + C_{BF} \log e} \tag{11.5}$$

For $C_{BF} = C_{CSF}$, this gives $\alpha > 0.59$.

In order to apply equation (11.4), we need estimates of C_{BF} and C_{CSF}, that is, estimates of the number of bits per element taken by our implementations of Bloom filter and CSF. For C_{BF}, we have $C_{BF} = 1.44$ corresponding to the theoretical coefficient of Bloom filters. On the other hand, we experimentally estimated C_{CSF} associated with the implementation we use as a function of the empirical entropy H_0, giving:

$$C_{CSF} = \begin{cases} 0.22H_0^2 + 0.18H_0 + 1.16, & \text{if } H_0 < 2 \\ 1.1H_0 + 0.2, & \text{otherwise.} \end{cases} \tag{11.6}$$

In the following we use the term *Bloom-enhanced Compressed Static Function*, BCSF for short, to speak about CSF possibly augmented by a prior Bloom filter, as described in this section. Algorithm 2 summarizes the computation of the BCSF data structure while Figure 11.5 graphically depicts its main steps.

11.2.2 Minimizer bucketing

A key idea to reduce the computational burden of counting k-mers, is to use minimizers to bucket k-mers and split the counting process across multiple tables (cf e.g. [101]). Here we use the same principle to bucket count values instead of k-mers themselves. Let $M_m(K) = \{\mu_m(q) \,|\, q \in K\}$ be the set of minimizers of all k-mers of K of a given length $m < k$ ($\mu_m(q)$ is the function returning the minimizer of k-mer q). We map the input set K onto the (smaller) set $M_m(K)$. To each minimizer $s \in M_m(K)$, corresponds the bucket $\{f(q) \,|\, q \in K, \mu_m(q) = s\}$. We call a minimizer and the corresponding bucket *ambiguous* if this set contains more than one value. The guiding idea is to replace f by a mapping g of $M_m(K)$ to $\mathbf{N}$. Querying value $f(q)$ for a k-mer $q \in K$ will reduce to first querying $g(\mu_m(q))$ and then possibly "correcting" the retrieved value. In other words, for each bucket, we replace its set of counts with one representative value, and we split the query into two operations: retrieving the representative from the buckets and correcting to reconstruct the original value. The rationale is that k-mers having the same minimizer tend to have the same count allowing multiple values to be dealt with by a single bucket. We consider two implementations which differ on how the representatives are chosen and how corrections are applied.

11.2.3 Lazy collision resolution: AMB

The *first implementation* is named AMB (from AMBiguity). For non-ambiguous minimizers u, AMB defines $g(u)$ to be the unique value of the bucket. For ambiguous minimizers v, we set $g(v) = 0$, where 0 is viewed as a special value marking ambiguous buckets (k-mers with count 0 are not present in the input). This has the disadvantage of providing no information about the values of ambiguous buckets, and also of making g less compressible (because of an additional value). On the other hand, this has the advantage of distinguishing between ambiguous and non-ambiguous buckets and allows the query to immediately return the answer for k-mers hashing to non-ambiguous buckets. As a consequence, unambiguous k-mers are not propagated further, and if $g(\mu_m(q)) \neq 0$ it can be immediately returned as $f(q)$. We then have to store mapping f restricted only to k-mers from ambiguous buckets, which we denote $\tilde{f}$. Both mappings g and $\tilde{f}$ are stored using BCSFs. A graphical representation of the above procedure is reported in Figure 11.6.

11.2.4 Correcting the effects of collisions: FIL

The *second implementation* is named FIL (from FILtration) and is shown in Algorithm 4. Here, $g(s)$ is defined to be the majority value among all values of its bucket, ties resolved arbitrarily. In particular, if s is a non-ambiguous minimizer then $g(s)$ is set to the unique value of the bucket. In practice, computing the majority value may incur a computational overhead as this requires storing bucket values until all values are known. An option to cope with this, not explored further in this work, is to use the "approximate majority" computed by the online Boyer-Moore majority algorithm [20]. We then store a "correcting mapping" $h : K \to \mathbf{N}$ defined by $h(q) = f(q) - g(\mu_m(q))$. That is, we construct another counting table h where each k-mer is associated to the correction factor $h(q)$, which,

added to the representative $g(s)$ results in the original count c. Both mappings g and h are stored using BCSFs. The rationale for this scheme is that, due to the properties of minimizers, $h(q)$ is supposed to be often 0, which makes h well compressible using BCSF. Note that because of the majority rule, 0 will always be the majority value of h (see Figure 11.7). Therefore, the Bloom filter of the BCSF storing h (if any) will hold k-mers q with $f(q) \neq g(\mu_m(q))$ (i.e. $h(q) \neq 0$). Then the BCSF will store h restricted to k-mers with $h(q) \neq 0$ together with a subset of k-mers (false positives of the Bloom filter) for which $h(q) = 0$.

11.2.5 Cascading

An intermediate layer corresponding to a minimizer length $m < k$, introduced in Section 11.2.2, can be viewed as a "filter" providing values for some k-mers and "propagating" the other k-mers to the next layer. Therefore, both implementations can be cascaded into more than one layer. This construction is reminiscent of the BBHash algorithm [122] or to cascading Bloom filters from [173].

For $m_1 < m_2 < ...m_\ell \leq k$, each layer i is then input some map f_{i-1} defined on a subset of k-mers $K_{i-1} \subseteq K$ ($f_0 = f$, $K_0 = K$) and outputs another map f_i defined on a smaller subset $K_i \subseteq K_{i-1}$. Each layer stores a bucket table for minimizers $M_{m_i}(K) = \{\mu_{m_i}(q) \,|\, q \in K_{i-1}\}$. The specific definition of f_i and K_i depends on the implementation.

The multi-layer scheme is particularly intuitive for the AMB implementation, where each layer stores a unique value for non-ambiguous minimizers and a special value 0 otherwise. In this case, K_i consists of those k-mers of K_{i-1} hashed to ambiguous buckets, and f_i is simply a restriction of f to those k-mers. Algorithm 3 shows a pseudocode of multi-level AMB extended to the approximate case (see Section 11.2.6 below), while Figure 11.8 depicting a graphical representation of the algorithm used for query. The multi-layer version of the FIL scheme is shown in Algorithm 5.

11.2.6 Extension to approximate counts

In addition to cascading, AMB can also be easily extended to work as an approximation algorithm. Consider, to this end, the layered bucketing procedure described in 11.2.5. In the exact case, a bucket is marked as colliding whenever it contains two or more distinct count values. In the approximate case, a collision is defined if a bucket contains a pair of counts, c_i, c_j such that $|c_i - c_j| > \delta$ with δ a pre-defined maximum absolute error. With this modification, the algorithm guarantees to output a value within the absolute error δ from the true count.

We chose $g(s)$ to be the minimum value in a bucket if the bucket is unambiguous. The rationale of using minimum is the decreasing behavior of k-mer spectra which implies that smaller counts are more frequent and therefore more likely to constitute the majority. In order to detect collisions, it is then sufficient to only remember the maximum $max(s)$ and minimum $min(s)$ values seen by each bucket and check if $max(s) - min(s) > \delta$. If that is the case, then the bucket is marked as colliding, otherwise $min(s)$ is chosen as representative (see Algorithm 3).

11.3 Results

11.3.1 Datasets

Three datasets were used in this study:

Data: A count table T
Result: A BCSF for T
Compute R, the spectrum of T;
Let $K_0 \subseteq K$ be the set k-mers with the most common frequency in R;
Compute $\alpha = |K_0|/|K|$;
Compute ε by using equation 11.4;
if $\varepsilon < 1$ **then**
    $C = K \setminus K_0$;
    Initialize a Bloom Filter B of $\lceil |C| \log(e) \log_2(\frac{1}{\varepsilon}) \rceil$ bits;
    Insert C into B;
    Compute $E = FP_B(K_0)$;
    $S = C \cup E$;
else
    $S = K$
end
Construct CSF for S;

Algorithm 2: BCSF construction

Data: Input count table T, $M = m_1 < m_2 < ...m_\ell \leq k$, δ
Result: One BCSF for each layer
$i = 0$;
$T_i = T$;
foreach minimizer length m in M **do**
    let L be a map from minimizers to pairs of values;
    foreach key-value pair (q, c) in T_i **do**
        let z be the minimizer of q;
        if z is a key in L **then**
            let $(r_{min}, r_{max}) = L[z]$;
            $L[z] = (min(r_{min}, c), max(r_{max}, c))$;
        else
            $L[z] = (c, c)$;
        end
    end
    let B be a map from minimizers to integer values;
    foreach minimizer z in L **do**
        let $(r_{min}, r_{max}) = L[z]$;
        if $r_{max} - r_{min} > \delta$ **then** $B[z] = 0$;
        else $B[z] = r_{min}$;
    end
    Compress B by using BCSF;
    Initialize T_{i+1};
    foreach key-value pair (q, c) in T_i **do**
        let z be the minimizer of q;
        if $B[z] == 0$ **then**
            $T_{i+1}[q] = c$;
        end
    end
    $i = i + 1$;
end

Algorithm 3: AMB multi-layer construction algorithm. Exact AMB can be obtained by setting $\delta = 0$.

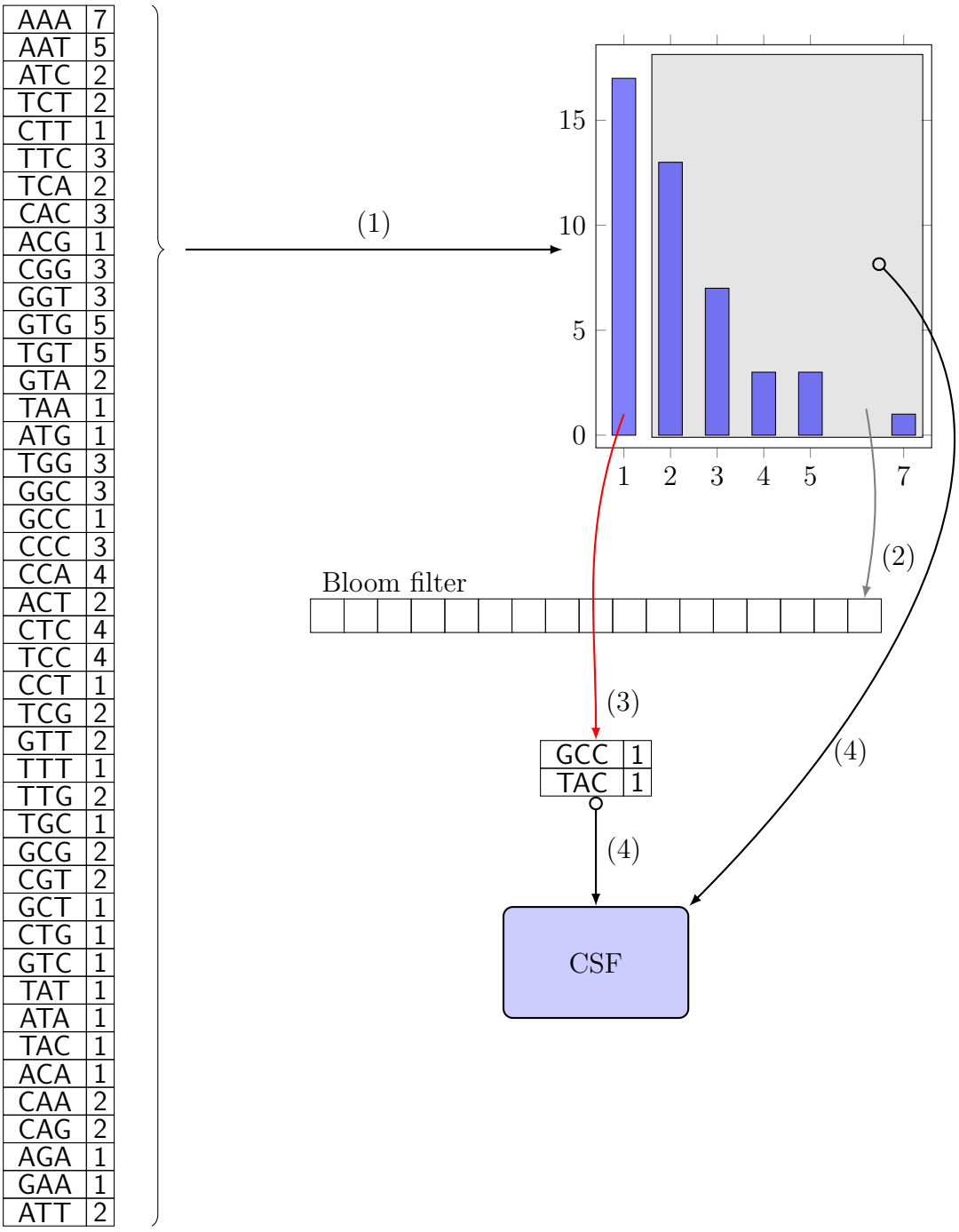


Figure 11.5: Graphical representation of Algorithm 2. Construction starts with histogram computation (1) in order to divide the given table into two sets K_0 and $K_1 = K \setminus K_0$ (the latter highlighted as a gray area). A Bloom filter is then built over K_1 (2) and false positives from K_0 are extracted (3) Finally, a CSF storing counters for both K_1 and the set of false positives is built (4) The final BCSF is the combination of the Bloom filter (if any) and the CSF.

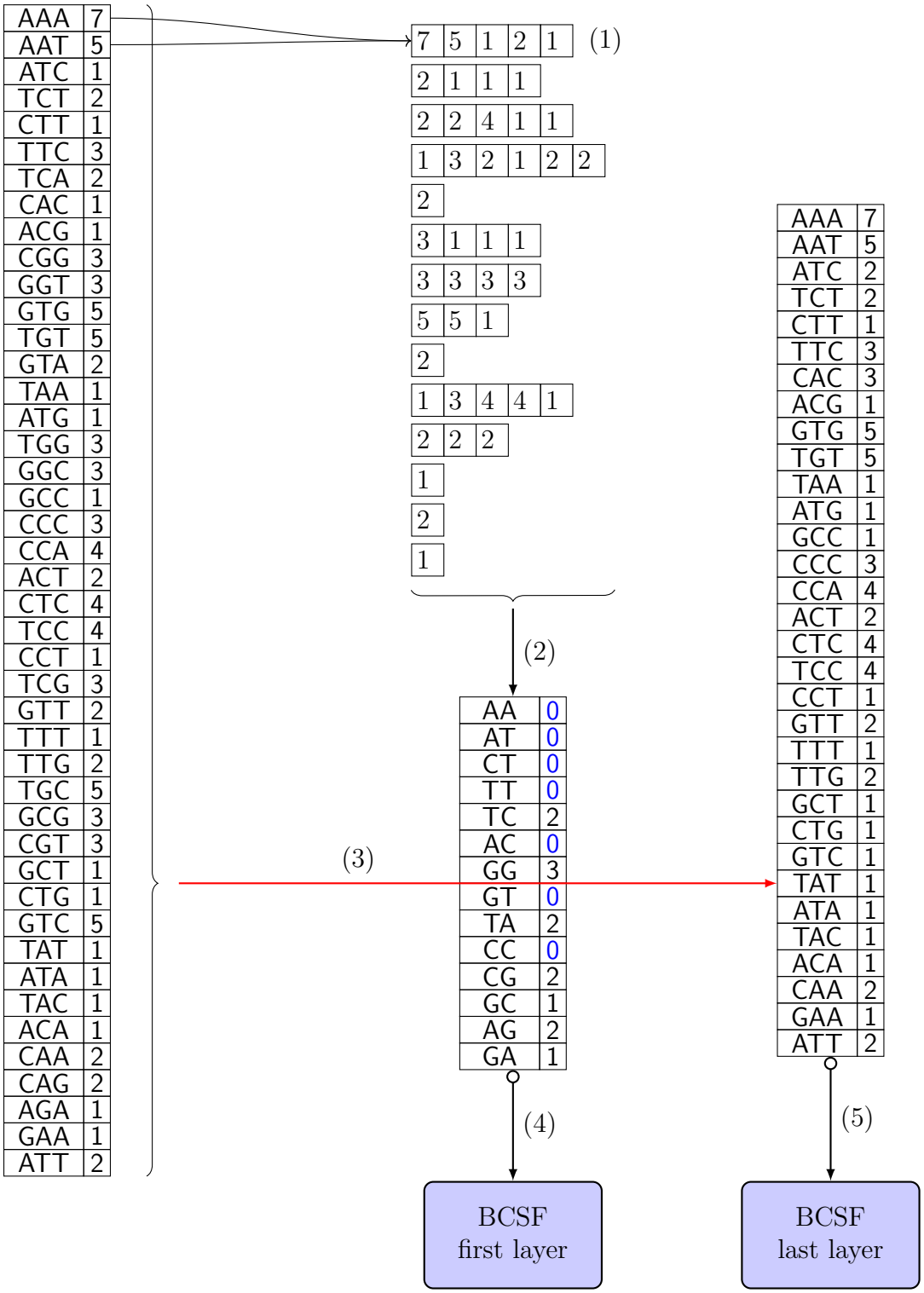


Figure 11.6: Depiction of AMB's construction algorithm for a table containing 3-mers, minimizer length $m = 2$ and $\delta = 0$. Counts are first bucketed using minimizers (1) Non-ambiguous buckets are reduced to their single value while ambiguous buckets are marked with the special value 0 (2) k-mers inside ambiguous buckets are then filtered (3) Both the array of representatives and the filtered table are then stored using BCSFs (steps (4) and (5)). Extension to the multi-layered case can be easily achieved by re-applying the same procedure on the filtered table with a bigger $m' > m$ as long as $m' \leq k$.

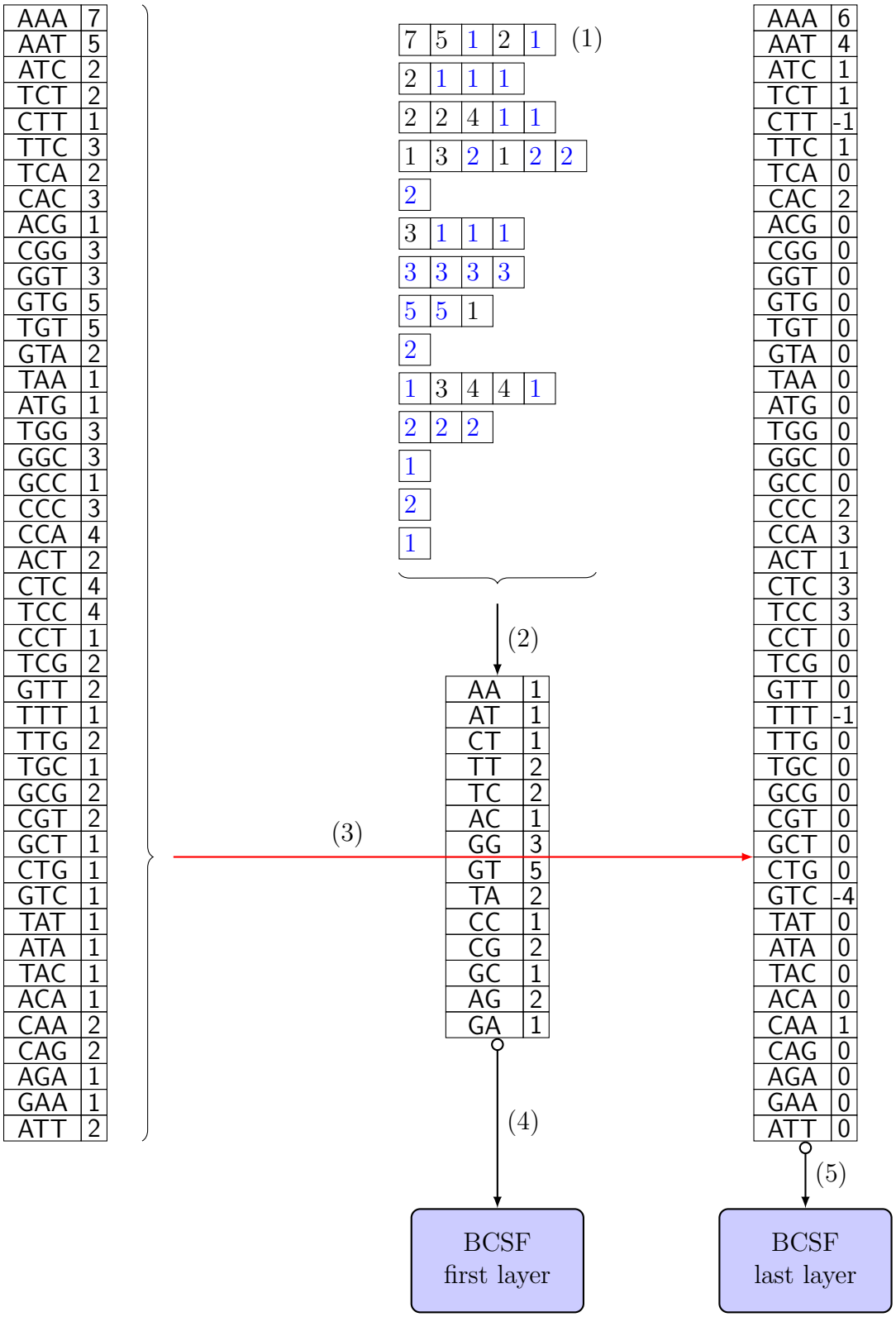


Figure 11.7: Similarly to AMB the construction of FIL data structures starts by bucketing counts (1) However, this time representatives are chosen by majority rule (2) so that the special value 0 is not required. The output table is obtained from the one in input by performing differences between counters and their representatives (3) instead of propagating ambiguous k-mers. The array of representatives and the new values are stored using BCSFs (4) (5) Note that the updated count table contains a large number of 0s, and it is thus more compressible than the original one.

Data: Input count table T, a minimizer length m_0
Result: FIL compressed structure
let L be a map from minimizers to multisets of values;
foreach key-value pair (q, c) in T **do**
    let z be the minimizer of q;
    insert c into $L[z]$;
end
let B be a map from minimizers to integer values;
foreach minimizer z in L **do**
    let b be the multiset at $L[z]$;
    let r be the representative value of b chosen by majority rule;
    $B[z] = r$;
end
Compress B by using BCSF;
Create output table O;
foreach key-value pair (q, c) in T **do**
    let z be the minimizer of q;
    $O[q] = c - B[z]$;
end
Compress O by using BCSF;

Algorithm 4: FIL construction algorithm.

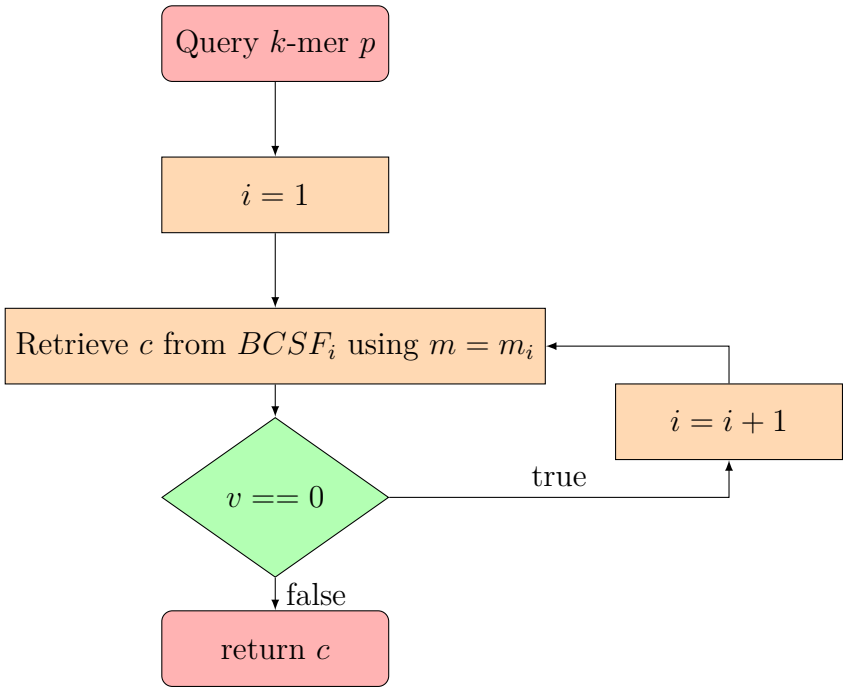


Figure 11.8: Multi-layered AMB queries. Thanks to the special value 0 each query stops as soon as a suitable value is found. Increasing minimizer lengths are used to gradually solve collisions in successive layers.

Data: Input count table T, $M = m_1 < m_2 < ...m_\ell \leq k$
Result: One BCSFs + Bloom filter for each layer
$i = 0$;
$T_i = T$;
foreach minimizer length m in M **do**
    Let L be a map from minimizers to multisets of values;
    Let $n = 0$;
    foreach key-value pair (q, c) in T_i **do**
        Let z be the minimizer of q;
        Insert c into $L[z]$;
        $n = n + 1$;
    end
    Let B be a map from minimizers to integer values;
    foreach minimizer z in L **do**
        Let b be the multiset at $L[z]$;
        Let r be the representative value of b chosen by majority rule;
        $B[z] = r$;
    end
    Compress B by using BCSF;
    Initialize T_{i+1};
    Let $p_q = 0$;
    foreach key-value pair (q, c) in T_i **do**
        Let z be the minimizer of q;
        if $B[z] \neq c$ **then**
            $T_{i+1}[q] = c - B[z]$;
            $p_q = p_q + 1$;
        end
    end
    Compute $\alpha = (n - p_q)/n$;
    Let $\epsilon = (1 - \alpha)/\alpha$;
    Initialize an empty Bloom Filter F of size $1.44 \log_2(1/\epsilon)$;
    Insert all elements of T_{i+1} into F;
    foreach key-value pair (q, c) in T_i **do**
        let z be the minimizer of q;
        if $B[z]$ == c and q is in F **then**
            $T_{i+1}[q] = c - B[z]$; `//Add false positive of` F `to` T_{i+1}`,`
            $c - B[z] = 0$ `by definition`
        end
    end
    $i = i + 1$;
end

Algorithm 5: FIL multi-layer construction algorithm.

1. The collection of fully assembled *Escherichia Coli* genomes from [213], from now on referred to as "df".

2. *Escherichia Coli* Sakai strain (NCBI accession number B000007) from the previous collection [213] but from now on referred to as "Sakai" to highlight its stand-alone usage.

3. Full reference genome of *Caenorhabditis Elegans*, strain *Bristol N2* downloaded from RefSeq (accession number GCF_000002985.6). We will refer to this dataset as "Elegans".

4. "SRR10211353" run of Illumina reads (10x coverage, *Escherichia Coli*) downloaded from NCBI SRA (accession number SAMN12880992).

Unless stated otherwise, FIL and AMB were run on all possible combinations of two and three minimizer lengths for $k \in [13, 15, 18, 21]$ with only the best combinations reported using the following naming convention:

- CSF: baseline CSF implementation from Sux4J [67].
- BCSF: extended CSF with Bloom filter from Section 11.2.1. It may get reduced to a simple CSF if the Bloom filter is not useful.
- AMB m_1 k: our *first implementation*, selecting each representative by minimum and marking colliding buckets with a special value.
- AMB m_1 m_2 k: same as before but with an additional layer.
- FIL m_1 k: our *second implementation*, saving into each bucket a majority-selected representative and saving corrections into its second layer.
- FIL m_1 m_2 k: same as before but with an additional layer.

11.3.2 Implementation

All construction code is written in python, except for the CSF part which is handled by a simple Java program using Sux4J [67]. A utility written in C using the code provided by Sux4J for reading and querying its CSFs provides time measurements. We use xxHash [1] to define an ordering over minimizers.

11.3.3 Compression of skewed data

Figure 11.9 reports memory usage when compressing the Sakai dataset. Simple CSF use more than 1 bit/k-mer, while Bloom-enhanced CSF (BCSF) is considerably more efficient, reaching space closer to the entropy. For relatively small k's ($k = 13$) AMB and FIL give almost the same results as BCSF, that is, minimizer-based bucketing is not helpful. For larger k's, however, both AMB and FIL lead to significant space reductions, eventually breaking the entropy barrier for larger values of k ($k = 18, 21$). This demonstrates that for larger k's, minimizers provide an effective way of factoring the space of k-mers in such a way that k-mers with equal counts tend to have the same minimizer.

More in detail, for larger k, the overwhelming majority of buckets are unambiguous (e.g. more than 99% of them, for $k = 18, m = 13$). As a consequence, AMB is able to "filter out" a very large number of k-mers with few buckets. Only a small set of k-mers,

[1] https://github.com/Cyan4973/xxHash

corresponding to ambiguous buckets, are propagated to the next layer. This, combined with the prevalence of one value due to the skewness of the count distribution, and the fact of using minimizers with increasing lengths, leads to highly compressible bucket tables. Altogether, this enables breaking the empirical entropy lower bound.

The situation is similar for FIL: its first layer is even better compressible than the one of AMB, due to the absence of the additional special value which makes the table of AMB slightly less compressible. On the other hand, the BCSF of the second layer table of FIL turns out to take more space than that of AMB. This is because its Bloom filter operates on the large set of all k-mers, which implies a very small value of ε to keep the set of false positives under control, and as a consequence, a relatively large Bloom filter. Overall, FIL turns out to yield a slightly larger space than AMB.

For small k's, none of our methods beats the empirical entropy, with minimizers unable to provide an efficient mean to factor the space of k-mers according to count values. On the contrary, we observe that in this case applying a BCSF to the input table provides the most efficient solution.

Since longer k-mers lead to more skewed data, and by extension, to smaller entropies, both AMB and FIL better compress whole genome count tables for increasing ks. The Elegans dataset (around 100 Mbp) tests this assumption. We randomly chose $m_1 = 18$ and $m_2 = 19$ for both three-layer AMB and FIL (ignoring m_2 for the two layered versions). Figure 11.10 demonstrates that our algorithms are not limited to bacterial genomes. Instead, they are applicable in the general case as long as count tables are computed on fully assembled data and k is large enough. Note that, under such a regime, larger values of k only reduce the entropy of the data, leading to more succinct representations whereas simple CSF could not go below 1.2 bits/k-mer.

11.3.4 Compression of higher entropy data

With very skewed data, collisions of k-mer counts may happen between unrelated k-mers simply because one counter value strongly dominates the spectrum. In order to demonstrate the utility of minimizers in a more general setting other than whole genome count tables, we applied our methods to less skewed distributions. To this end, we compressed the k-mer count tables when using dataset SRR10211353 whose results are presented in Figure 11.12. As opposed to fully assembled genomes, entropy in this case remains well above 1 even for larger values of k. Nonetheless, both AMB and FIL are able to produce representations more compact than both simple CSFs and BCSFs for all $k > 13$, beating the entropy lower bound.

Further proof of the ability of minimizer-based bucketing to boost compression of k-mer count tables can be found in Figure 11.11. Here, we compressed the table produced by counting the number of occurrences for each k-mer among the 29 E.Coli genomes of dataset df (note that df is a mnemonic for "document frequency"). Note that entropy does not decrease as rapidly as before with increasing k, despite counts bounded in the range $[1, 29]$.

The use of minimizers for larger k's, proves to be beneficial again, with AMB and FIL requiring much less space than the empirical entropy of the data. Again, when $k = 13$, both AMB and FIL do not have an advantage over a simpler (B)CSF. For even smaller k-mers (B)CSF remains the best option (see Figure 11.13). The seemingly erroneous exceptions (BCSF taking more space than simple CSF) are explained by the approximation carried out by formula (11.2) (assumption of equal values of C_{CSF} in both sides).

11.3.5 Approximate counts

In many applications, it is acceptable to tolerate a small absolute error in retrieved counts. Figure 11.15 reports space usage when using the approximate version of AMB ($\delta > 0$, see section 11.2.6) on the Sakai dataset. Results for the exact algorithm ($\delta = 0$) are reported in Figure 11.14 for comparison.

In order to show how the approximate algorithm achieves better compression ratios, k was chosen from $[10, 11, 12, 13]$, a range of values which is particularly difficult for AMB (or FIL) with $\delta = 0$. Trying all possible minimizer combinations compatible with such ks, the best results are obtained for very short minimizer lengths (between 1 and 5). Building minimizer layers for such small values of m does not lead to better compression than simple (B)CSFs, with Figure 11.14 showing no tangible differences between (B)CSFs and AMB (or FIL). For these reasons, minimizer lengths in Figure 11.15 are equal to $k-1$ (and $k-2$) for every choice of k (e.g. if $k = 10$, layers will be $8, 9, 10$ for three-layer AMB). Using the same small lengths of the exact case would not allow meaningful bucketing of counts values.

An interesting observation about the approximate case is that AMB with three layers is substantially better than AMB with two layers only for $k = 12$ and $k = 13$. For $k = 10$ and $k = 11$ both versions give almost the same results.

11.3.6 Query speed

Figure 11.16 shows query time averaged over all distinct k-mers, in ns/k-mer. Simple CSFs, not surprisingly, are the fastest method, with BCSF having a negligible effect on the average query speed. On the other hand, bucketing has a tangible effect on performance, with speed negatively affected by additional layers. For short k-mers, both FIL and AMB are slower than the simple CSF by a factor equal to their number of layers.

The situation is different for larger k's where AMB is only marginally slower than a bare-bones CSF. This is because most queries are solved without accessing all layers every time, thanks to unambiguous buckets. Two-layered FIL, on the other hand, gives almost constant average query times across all test, since all queries have to access both of its layers to reconstruct the exact count value. We did not perform tests for FIL with 3 layers because it will always be slower than the two layered version.

11.3.7 Technical observations

In all reported cases, good minimizer lengths for the first layer (m_0) follow the rule: $m_0 > m_s = (\log_4|G| + 2)$ with $|G|$, the size in base pairs of the genome. Smaller m_0, are no longer capable of partitioning k-mers in a meaningful way. Furthermore, space tends to first monotonically decrease to a minimum for increasing minimizer lengths, to increase again once the optimal value is passed. It is therefore possible to find the minimum by sequentially trying all possible minimizers greater than m_s and stop as soon as the compressed size starts to increase again.

If it is not possible to choose $m_0 > m_s = (\log_4|G| + 2)$ because, e.g. k is already too small, approximation might be a viable option even for relatively small δ. The only caveat to pay attention to in this case is to check if a minimizer layer would be useful or not. If yes, δ can be incremented without further adjustments compared to exact case. If not, minimizer lengths for the bucketing layers should be chosen as big as possible to allow meaningful bucketing of count values.

Our results also show how multiple layers have a marginal effect on final compression sizes. In case of AMB, using three layers is always helpful, compared to the two-layer case. Best results are usually achieved for combinations including the best minimizer length

obtained for the two-layer case. On the other hand, FIL with three layers seems to be advantageous only for low entropy data, performing worse than its two-layer counterpart when compressing document frequency tables and for small k's.

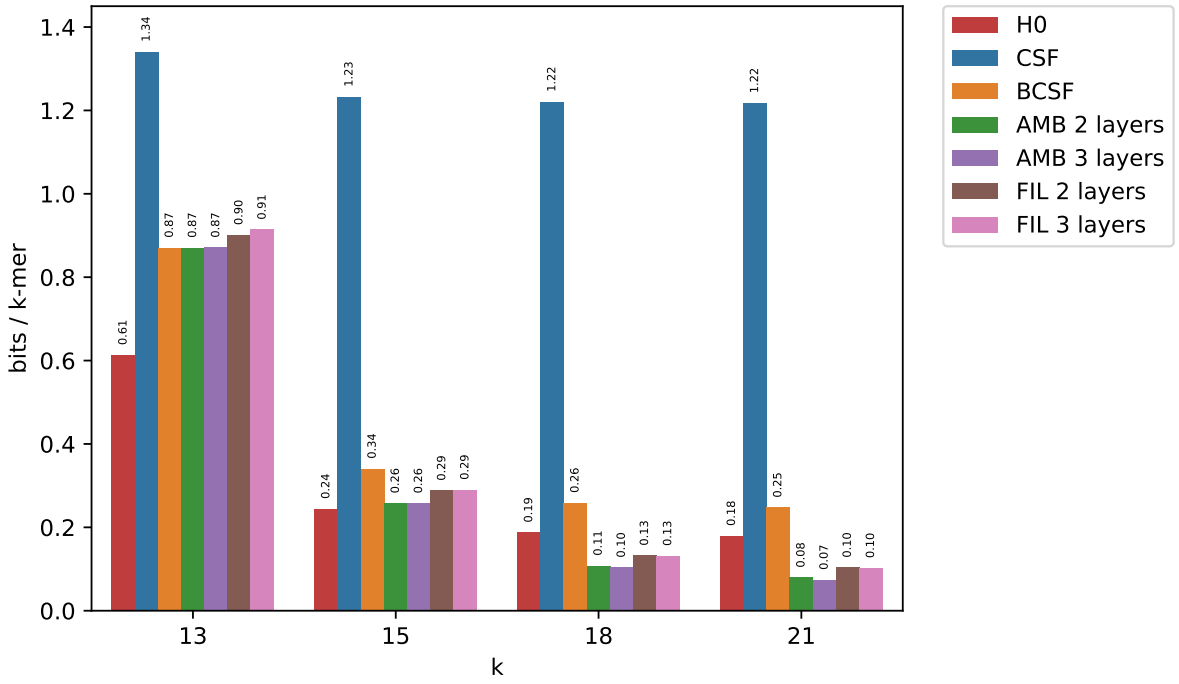


Figure 11.9: Results for the Sakai dataset for big values of k. For presentation purposes, H_0 is represented as an additional red column in each subgroup.

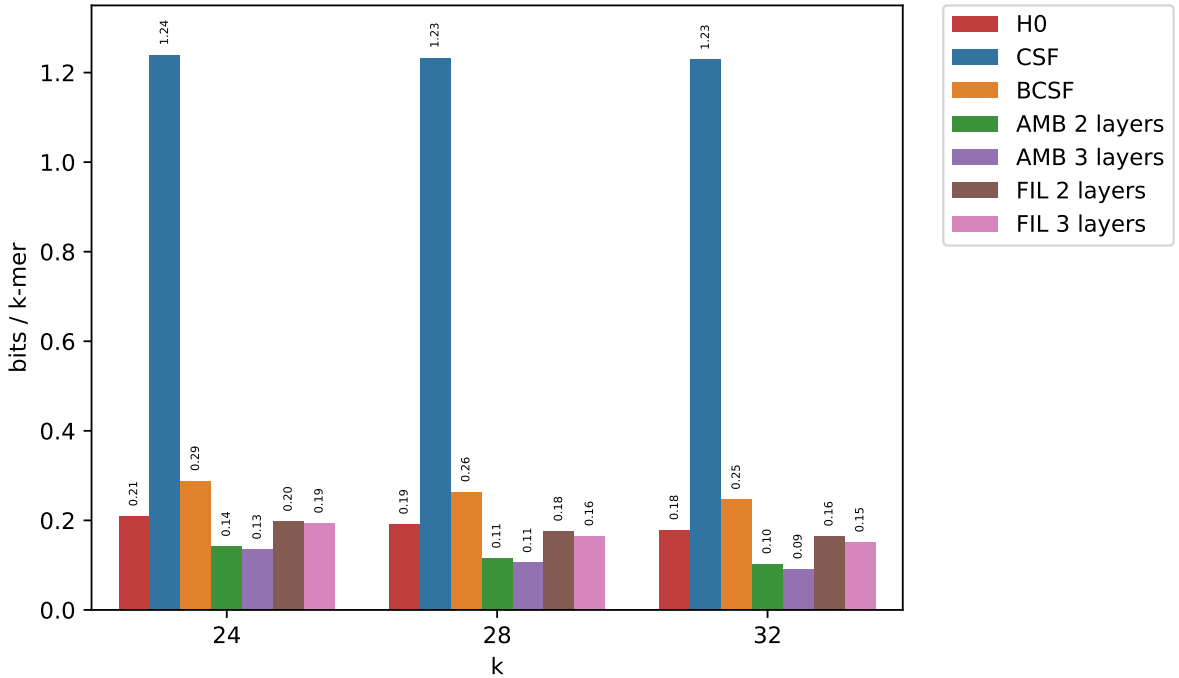


Figure 11.10: Results when compressing the reference genome of *C.Elegans*

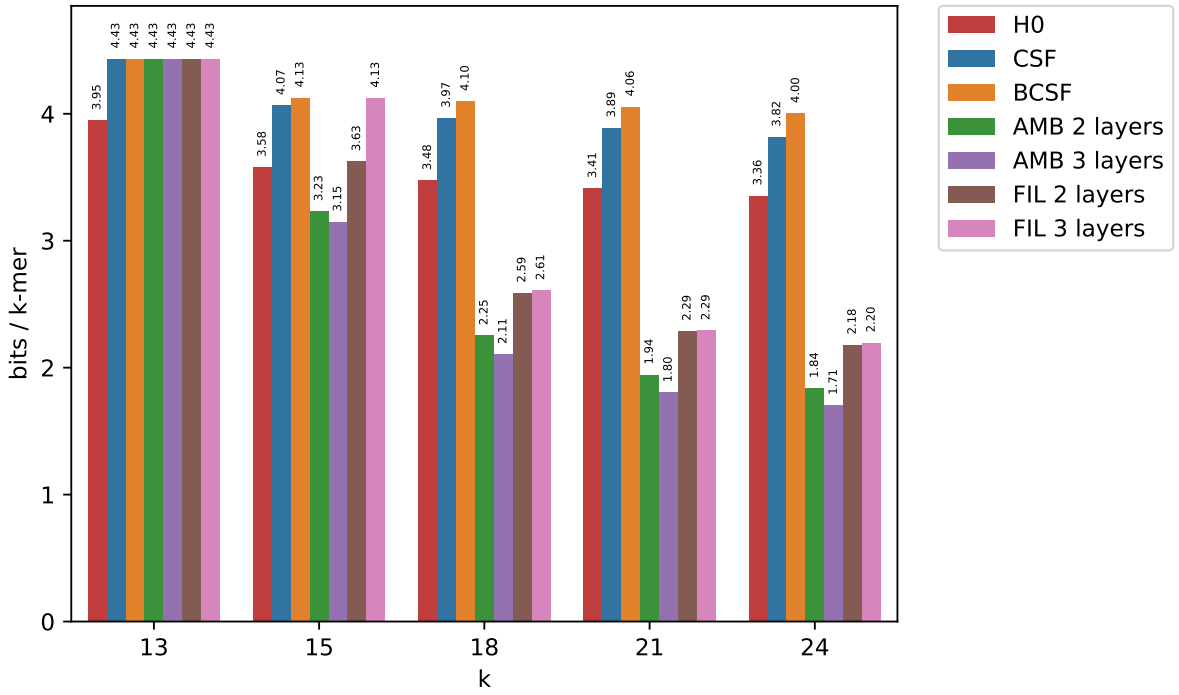


Figure 11.11: Compressed space usage for the high entropy df dataset.

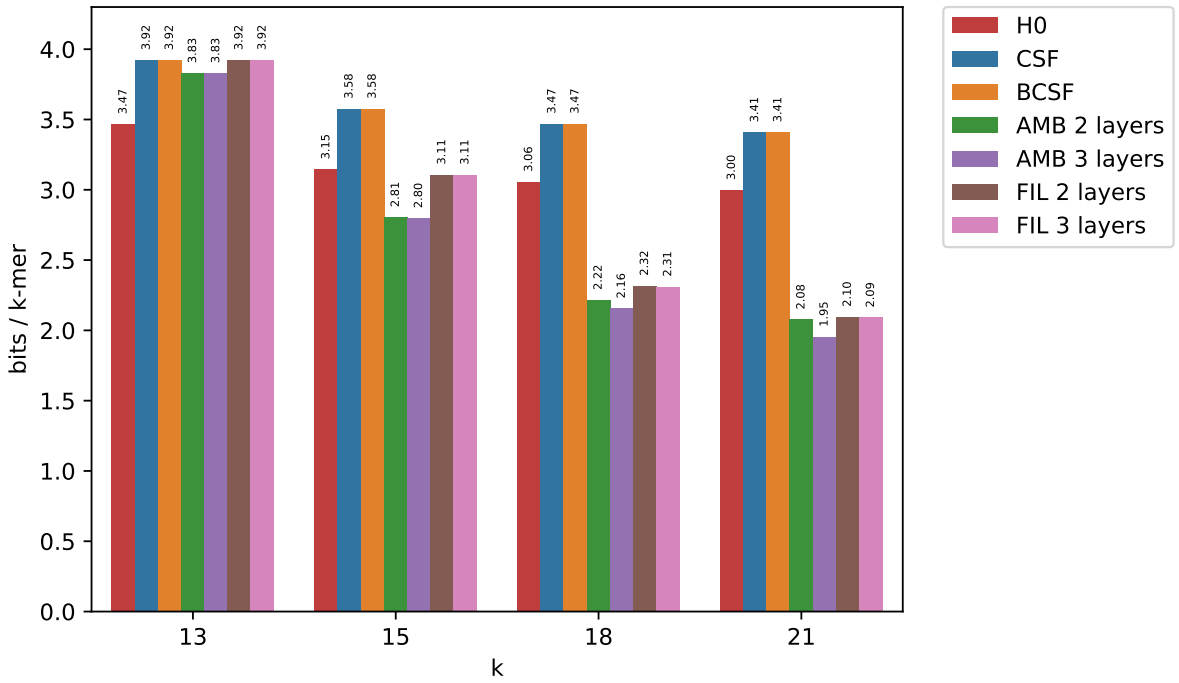


Figure 11.12: Compressed space usage for the high entropy SRR dataset.

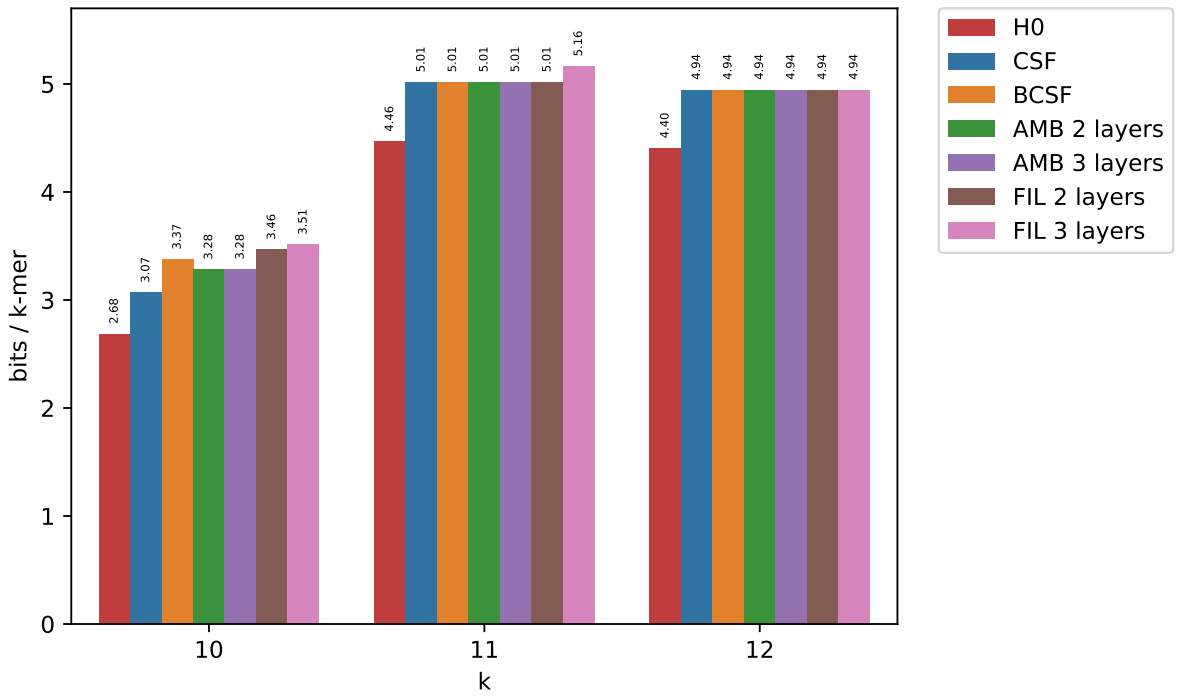


Figure 11.13: Compressed space usage for the low entropy df dataset.

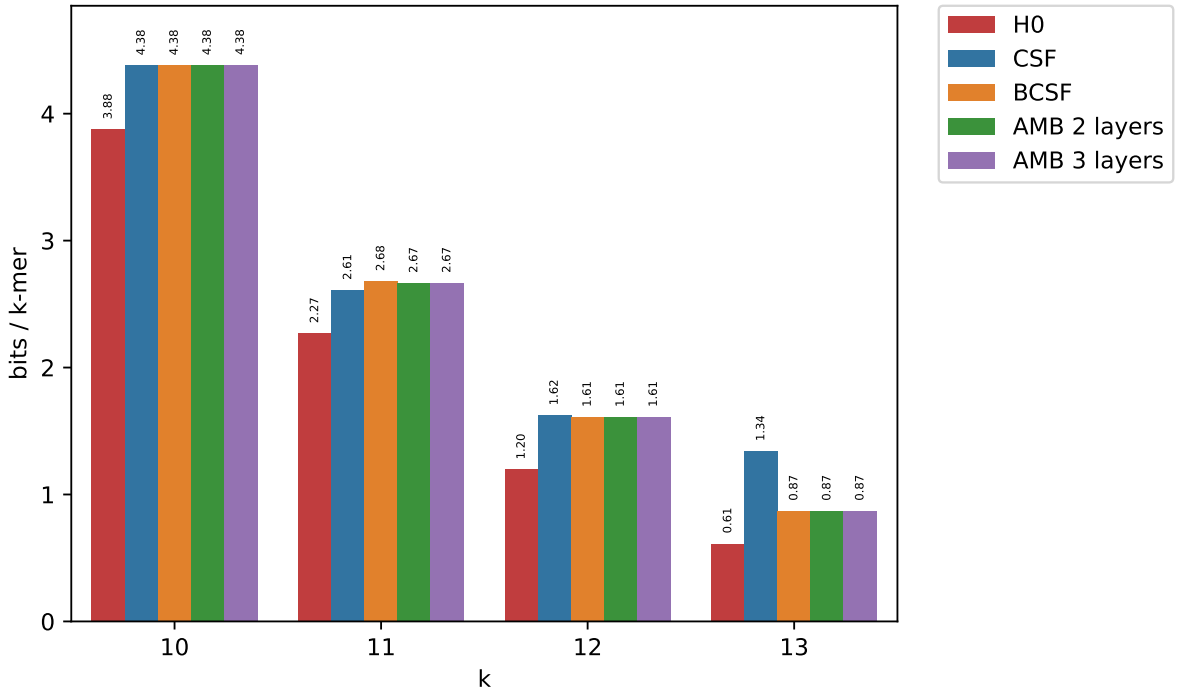


Figure 11.14: Space usage for the Sakai dataset with small k when using AMB (FIL is slightly worse and was omitted). Minimizer lengths vary between 1 and 5 indicating that the best option is to use a simple (B)CSF.

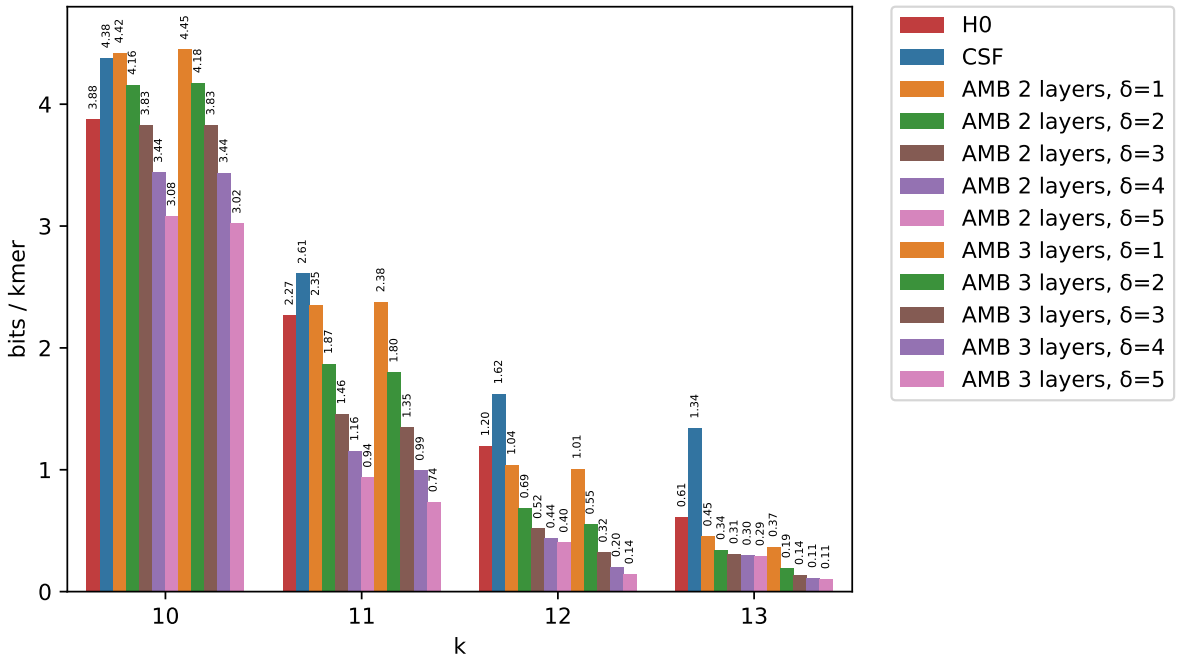


Figure 11.15: Space usage when using the approximated version of AMB. Entropy (red columns) and CSF (blue columns) are reported for comparison. Unlike Figure 11.14, AMB is able to break the empirical entropy lower bound when small errors are acceptable.

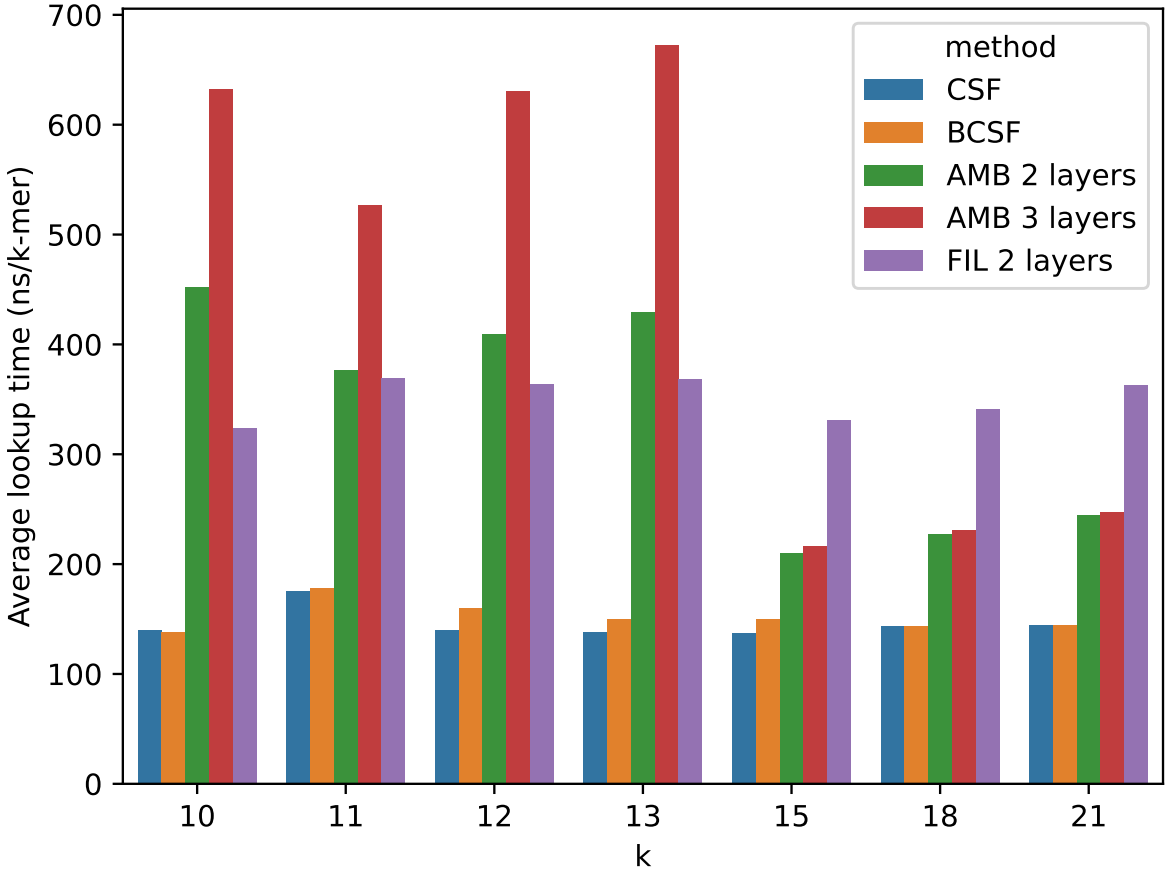


Figure 11.16: Average query time for AMB with 2 and 3 layers and FIL with 2 layers.

Chapter 12

Discussion

We introduced three data structures to represent compressed k-mer count tables.

BCSFs combine Compressed Static Functions, as implemented in Sux4J software [67], with Bloom filters thus allowing for much better compression of skewed distributions with empirical entropy smaller than 1. To the best of our knowledge, this was the first time CSFs were used in a bioinformatics application. We provide a method to optimally dimension BCSFs.

AMB and **FIL** pair BCSFs with a bucketing procedure where count values are mapped into buckets according to k-mer's minimizers. This locality-sensitive hashing scheme allows to efficiently factor the space of counts, which leads to breaking the empirical entropy lower bound for large enough k's. Both algorithms use slightly different strategies in decomposing the input table across minimizer layers:

- AMB tries to reduce space by representing counts of multiple k-mers at once. Colliding buckets containing multiple distinct values are marked with a special value and all involved k-mers are propagated to the next layer. Since non-colliding k-mers do not propagate further, each successive layer is expected to contain fewer k-mers than its predecessor.

- FIL, on the other hand, is not limited by collisions in buckets since it is always able to assign representatives to buckets by majority rule. The input table is then altered to map each k-mer to the difference between its value and its representative. Count values are then retrieved by adding representatives to these correction factors. Memory reduction comes from the fact that, by construction, 0 is the expected most common value in each layer, making them more compressible using BCSFs.

Our last contribution is an extension of AMB to the **approximate** case, gaining more space at the expense of a small and user-definable absolute error on the retrieved counts.

We validated our algorithms on four different types of count tables, two fully assembled genomes (*E.Coli* and *C.Elegans*) of different sizes, one dataset of *E.Coli* reads at 10x coverage and one document frequency table of 29 different *E.Coli* genomes, for different k-mer lengths. AMB and FIL have a clear advantage when minimizers are long enough to bucket k-mers in a meaningful way, for both skewed and high entropy data. When it is not possible to define a long-enough minimizer length, the advantage of using intermediate minimizer layers vanishes, and simple CSF and its BCSF provide a better solution.

In all our experiments, the data structures produced by AMB are smaller than the ones given by FIL. This is due to the bigger BCSF needed by FIL in order to store the propagated k-mers after the bucketing procedure. Since there is no way for FIL to distinguish between colliding and non-colliding k-mers, output tables contain all k-mers in input. Despite the better compressibility of the set of corrections, the combined size of the two BCSFs of FIL ends-up bigger than its AMB equivalent. Potential regimes

where FIL performs better than AMB are an interesting open question left for future developments.

At query time, CSF and BCSF are the fastest methods requiring about 100ns on average for a single query. For a fixed number of layers, AMB is faster than FIL in all situations when minimizers are useful. FIL becomes faster than AMB only for those cases when both algorithms achieve worse compression ratios than simple (B)CSF.

We consider this study to be the first step towards designing efficient representations for k-mer count tables occurring in data-intensive bioinformatics applications. One possible future direction is compression of RNA-Seq experiments where counts may translate expression levels of genes. Another example is metagenomics where different species may be present with different abundances which can be captured by k-mer counts. In such applications, efficient representation of k-mer counts can be particularly beneficial.

Part IV

Efficient reconciliation of genomic datasets of high similarity

Chapter 13

Context and motivation

13.1 Problem statement

Despite being faster than their alignment-based counterparts, algorithms based on k-mer sets are starting to struggle when applied to the large datasets produced nowadays [131, 73, 97]. To deal with this issue, a considerable effort has been put into developing optimized data structures, with succinct solutions [155, 97] and approximate membership data structures [173, 73, 16, 21, 140] being two examples.

In recent years, sketching techniques have been gaining increasing attention thanks to their capacity of drastically decreasing space usage. MinHash is probably the most well-known representative of this family of algorithms. As already mentioned in Chapter 5, MinHash comparisons of DNA sequence datasets were pioneered in Mash software [148] and subsequently used in several other tools. With this approach, input datasets are transformed into smaller "sketches" on which subsequent comparisons are performed. In short, sequences are first fragmented into their constituent k-mers which are then hashed, with each sketch storing only s minimum values, with s defined by the user. The fraction of shared hashes between two sketches is an unbiased estimator of the Jaccard similarity index [26]. A MinHash sketch can thus be viewed as a sample of the set of k-mers of the sequence it represents. Given that s is much smaller than the genome length, working with the sampled hashes leads to fast pairwise comparisons using small memory. However, when two sequences are close and share most of their k-mers, MinHash sketches of small size are not able to reliably estimate their degree of similarity since differences are likely to be missed during sampling. Here, we show a combination of Invertible Bloom Lookup Tables (a sketching technique initially conceived for the problem of set reconciliation) and syncmer-based sampling is able to efficiently estimate the Jaccard index of similar sequences.

13.2 Contributions

In Chapter 14, we propose an alternative approach to evaluate the difference in k-mer composition of two related datasets. Our method relies on the Invertible Bloom Lookup Table (IBLT) data structure by [69, 55] which is an extension of Bloom filters (see Sections 5.1.5 and 5.1.6), supporting deletions and, most importantly, enumeration (with high probability) of stored items. One of the applications of IBLTs is reconciliation of two sets: in a scenario considered in [69], a set A is stored in an IBLT which is then transmitted to the holder of another set B. By screening B against the IBLT of A it is possible to recover the items $A \setminus B$ and $B \setminus A$, with high probability. This is done through the so-called *peeling* procedure [50].

Inspired by ideas of [159] we make one step further, we recover $A \setminus B$ and $B \setminus A$ from IBLTs of both A and B, rather than a single sketch and the whole other set. A crucial property is that the *size of these IBLTs is bounded in terms of the symmetric difference size* $(A \setminus B) \cup (B \setminus A)$ rather than the size of the original sets. This provides a key to the efficiency of our solution when input sets are similar: even if input sets are very big, their difference can be recovered using a data structure (sketch) whose size is proportional to the size of said difference, and not to the size of the sets. Further, the symmetric difference allows us to estimate the Jaccard similarity, using information about input set sizes. Thus, whereas close datasets require larger MinHash sketches to be properly compared, our method, on the contrary, requires smaller memory.

Another ingredient of our solution is k-mer sampling. Intuitively, since two adjacent k-mers share $k-1$ bases, the information stored in the set of all k-mers appears highly redundant. One popular method of sampling k-mers from genomic sequences is based on minimizers [167]. Under this technique, consecutive sampled k-mers are within a bounded distance from each other and therefore no large portion of the sequence can remain unsampled. Another favorable property is that similar regions are likely to yield similar samples of minimizers. However, it has recently been shown that estimating Jaccard similarity based on minimizer sampling leads to a bias [12]. Here we propose to replace minimizers by syncmers [52]. Syncmers provide another way of k-mer sampling which has certain advantages over minimizers. As opposed to minimizers, syncmers are not context-dependent: for a k-mer to be a syncmer depends on the k-mer alone regardless the context where it occurs, and, under standard randomness assumptions on involved hash functions, all k-mers have equal chance to be syncmers. As a consequence, syncmer sampling leads to an unbiased estimate of Jaccard similarity, as the fraction of syncmers among shared k-mers (intersection) is expected to be the same as that among all k-mers (union). We experimentally validate that this is, indeed, the case.

By combining syncmer sampling with IBLTs, we obtain a space-efficient method for accurately estimating Jaccard similarity for similar datasets. For datasets of high similarity, the proposed method is superior to the popular MinHash algorithm [148], both in terms of memory and precision. We also propose an application of this technique to retrieve k-mers that differ between two given datasets. Our method computes a superset of those k-mers with a limited number of spurious k-mers. In particular, under the assumption that each k-mer occurs once, our method computes the exact set differences between involved k-mer sets. We validate our algorithms on both simulated data and on real datasets made of *SARS-CoV-2* and *Staphylococcus Pneumoniae* genomes in Section 14.3.

Our implementation of IBLTs is available at `https://github.com/yhhshb/km-peeler`

Chapter 14

KM-peeler: Invertible Bloom Lookup Tables for fast k-mer set differences

The problem of retrieving symmetric set differences using small space has been extensively studied in network applications under the name of "set reconciliation". As the name "reconciliation" suggests, distributed systems often have the need to synchronize slightly divergent copies of the same set stored in different nodes. Instead of exchanging whole sets, the optimal solution consists in exchanging only the unique items needed to recreate perfect copies. However, in order to know the differences to be sent, whole sets differences have to be computed first, nullifying the intuition of only exchanging the minimum required information necessary for reconciliation.

Invertible Bloom Lookup Tables [55, 69] solve this problem by transforming sets into compressed sketches. IBLTs are dimensioned depending on the number of expected differences between the involved sets. Nodes can thus exchange these small sketches, subtract the elements of their own (uncompressed) set and retrieve the difference from the so-updated resulting sketch. Note how Jaccard similarity can be reframed in terms of such differences and set sizes, making IBLTs suitable sketches for it. In case of k-mer sets, space can be further reduced by sampling k-mers before insertion into IBLTs.

In the next section (Section 14.1) we introduce the remaining building blocks of our method not covered in Section 5.1. The method itself is introduced in Section 14.2, with an experimental evaluation in Section 14.3.

14.1 Key algorithmic ideas

14.1.1 Random sampling

Random sampling for a given sampling rate $1/\nu$, where ν is assumed to be integer, and a fixed random hash function $h : \Sigma^k \rightarrow [0..\nu - 1]$ with good statistical properties is performed by sampling (i.e. keeping) all k-mers q that satisfy $h(q) = 0$.

14.1.2 Minimizers

Unlike Locom which uses minimizers as a Locality Sensitive Hashing scheme for k-mers (sections 11.1.2 and 11.2.2), here they are employed in their original role of sampling technique [167] and [178]. Figure 14.1 presents this mode of operation. In this context, minimizers are defined by a triplet of parameters (k, w, h), where k is the k-mer length, w a window size, and h the function defining a k-mer order. Each window $S[i, w + k - 1]$

defines a minimizer which is the minimal k-mer among w k-mers occurring in $S[i, w+k-1]$ w.r.t. the order given by h. Two neighboring minimizers are thus separated by at most w positions making it impossible to have large stretches of the original sequence not covered by any minimizers.

Since two neighboring windows at positions i and $i+1$ are likely to share their minimizer, minimizers provide a way to sample k-mers from a sequence with bounded distance between consecutive samples. An advantage of this sampling strategy is that similar sequences will likely have similar lists of minimizers, which is useful for mapping algorithms [114, 91]. Under reasonable assumptions, the density of minimizers, i.e. the fraction of sampled k-mers, is $\frac{2}{w+1}$ [167, 52]. If minimizer positions in the original sequence are not important, they can be discarded and the resulting k-mer multiset can be reduced to a simple k-mer set.

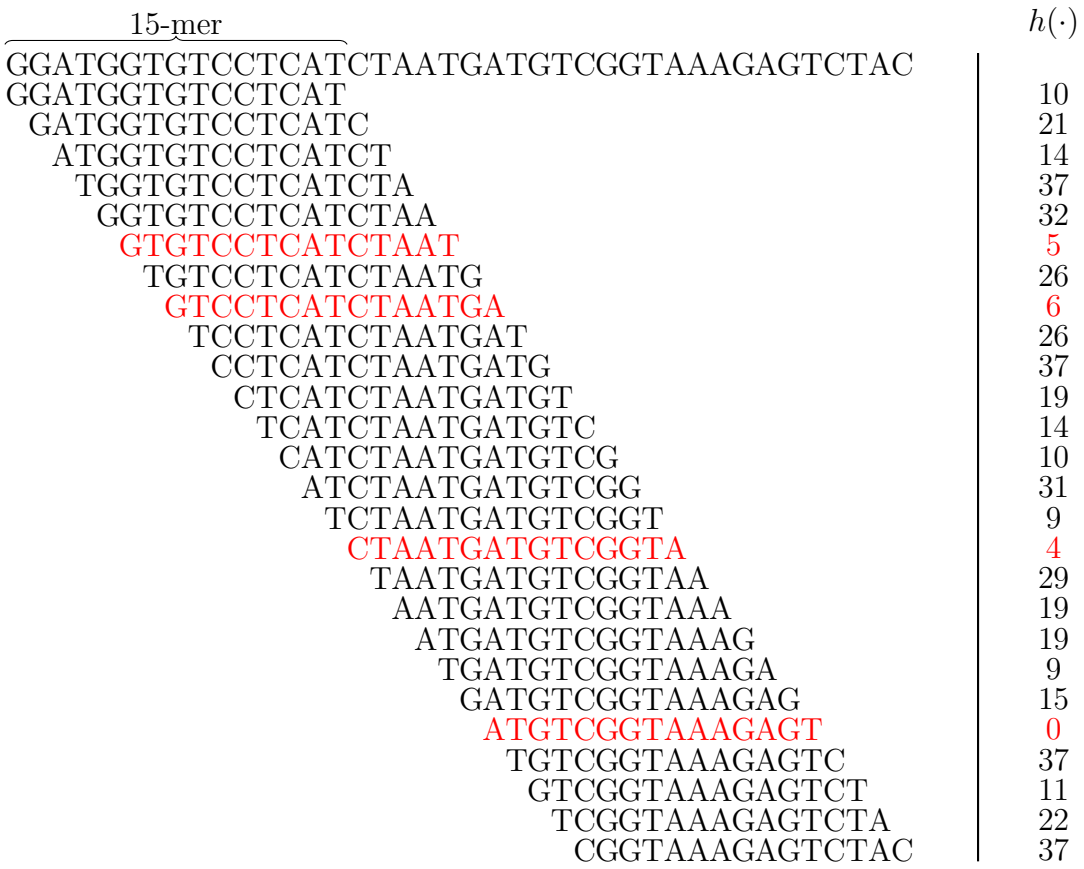


Figure 14.1: Example of minimizers as a sampling technique with a sequence of length 40, $k = 15$, $w = 8$. Note how two consecutive minimizers (highlighted in red) are never separated by more than $w = 8$ bases.

14.1.3 Syncmers

Minimizers are susceptible to mutations of any base of their window [52]. That is, a k-mer may cease to be a minimizer if a modified base occurs not only inside this k-mer, but also in its close neighborhood. Sampling with a higher density alleviates this problem, but it reduces the advantages of the methods because more minimizers are selected. Methods to generate minimizers with the best possible density exist [45, 54], but they are usually offline algorithms, limiting their potential applications outside alignment.

Syncmers are a family of alternative methods to minimizers that do not suffer from this issue [52]. Similarly to minimizers, syncmers are defined using a triplet of parameters

(k, z, h) where $z < k$ is used to decompose each k-mer into its constituent z-mers and h defines an order over them. A k-mer q is a *syncmer* (called *closed syncmers* in [52]) iff its minimal z-mer occurs as a prefix (position $i = 0$) or as a suffix (position $i = k - z + 1$) of q. Thus, a syncmer is defined by its sequence alone, regardless the context in which it occurs. For this reason, syncmer sampling has been shown to be more resistant to mutations thus improving the sensitivity of alignment algorithms [52].

Similar to minimizers, consecutive syncmers occur at bounded distance. More precisely, consecutive syncmers must overlap by at least z characters and therefore "pave" the sequence without gaps (except for the beginning and end), as shown in Figure 14.2. The fraction of syncmers among all k-mers is estimated to be $\frac{2}{k-z+1}$ [52].

Figure 14.2: Syncmers computed on the same sequence as Figure 14.1. $k = 15$ and $z = 4$. The minimum z-mer of each syncmer is highlighted in red.

14.2 KM-peeler

14.2.1 Set reconciliation from two IBLTs

Invertible Bloom Lookup Tables from Section 5.1.6 can be used to achieve set reconciliation between two sets A and B. In practice, this translates to recovering sets $A \setminus B$ and $B \setminus A$. Under a scenario described in [69], the holder of A stores it in an IBLT T_A which is then transmitted to the holder of B. Elements of B are then deleted from T_A. In the resulting IBLT, P-fields with $T_A[i].C = 1$ correspond to elements of $A \setminus B$ and those with $T_A[i].C = -1$ to $B \setminus A$. The peeling process is applied to either of such fields. Whenever $T_A[i].C = 1$, we delete $p = T_A[i].P$ from T_A on condition that $h_e(p) = T_A[i].H$. Similarly, whenever $T_A[i].C = -1$, we add (XOR) $p = T_A[i].P$ to T_A on condition that $h_e(p) = T_A[i].H$. The process lists all elements of both $A \setminus B$ and $B \setminus A$ w.h.p.

Inspired by work [159], we modify the above scheme in order to recover the symmetric difference between A and B from their respective IBLTs T_A and T_B, rather than from the IBLT of one set and the whole other set. To do this, we define T_A and T_B to be of the same size and to use the same hash functions. We then compute the difference of T_A and T_B, denoted T_{A-B} and defined through $T_{A-B}[i].C = T_A[i].C - T_B[i].C$, $T_{A-B}[i].P = T_A[i].P \oplus T_B[i].P$, and $T_{A-B}[i].H = T_A[i].H \oplus T_B[i].H$. Information about elements of $A \cap B$ is "cancelled out" in T_{A-B}, that is, T_{A-B} holds elements of $(A \setminus B) \cup (B \setminus A)$. Peeling then proceeds as usual, listing both $A \setminus B$ and $B \setminus A$ with the distinction made possible by looking at the sign of C.

A remarkable property of this scheme is that it allows one to recover set differences using a space proportional to the size of those differences regardless the size of the involved sets. Indeed, for the peeling process to succeed w.h.p., it is sufficient that the size of T_{A-B} be $O(n)$ where $n = |(A \setminus B) \cup (B \setminus A)|$ (see (5.2)). This is particularly suitable for the bioinformatics framework where we are often dealing with highly similar datasets, such as genomes of different individuals or closely related species. Other than that, adapting IBLTs for sets of genomic sequences is as simple as allocating enough space inside the IBLT buckets for their 2-bit representation.

14.2.2 Making buckets lighter

In the above scheme of IBLT difference, the H field becomes important as the case $T_{A-B}[i].C = 1$ (or $T_{A-B}[i].C = -1$) can occur due to a spurious "cancelling out" of distinct keys. However, to save space, we propose to get rid of the H field and replace the "checksum" verification by another test: if $T_{A-B}[i].C = 1$ (resp. $T_{A-B}[i].C = -1$), we check whether $p = T_{A-B}[i].P$ is a valid key by checking if $h_j(p) = i$ for one of $j \in [1..r]$. This allows us to save space at the price of additional verification time. This technique works particularly well for large IBLTs, but it becomes less effective for small ones, as the "false positive" probability is proportional to the size of the table.

14.2.3 Combining sampling and IBLTs for Jaccard similarity estimation

We now turn to our main goal: estimating Jaccard similarity of two k-mer sets using IBLTs. The common approach uses MinHash sketching as described in [148] (see Section 5.1.4). However, MinHash requires larger sketches to measure similarity of close datasets. One possible idea could be to store MinHash sketches in IBLTs in hope to use them for estimating Jaccard similarity through the IBLT difference scheme from the previous section. This, however, runs into an obstacle due to the fact that applying (5.1) requires knowledge of k-mers belonging to the sketch intersection, and not only to sketch differences. Comparisons involving bottom-s sketches need to stop after s distinct (hash) values have been processed. IBLTs do not allow this: knowing the symmetric difference between two MinHash sketches does not allow to retrieve the fraction of shared hashes in the union of size s. Thus, IBLTs are incompatible with bottom-s MinHash sketches, and the original MinHash implementation by [26] is not competitive in terms of computational resources since it requires s independent hash functions.

Rather than working with the entire sets of k-mers, we resort to sampling. It is known that sampling minimizers incurs a bias in estimating Jaccard similarity [12]. Instead, we propose to use syncmers, which don't suffer from being context-dependent thus resulting in an unbiased estimator of Jaccard similarity.

Our approach consists in storing sampled k-mers in IBLTs and apply the IBLT difference technique to recover set differences. Then, Jaccard similarity is estimated by

$$J(A,B) = \frac{|A| - |A \setminus B|}{|A| + |B \setminus A|} = \frac{|B| - |B \setminus A|}{|B| + |A \setminus B|}. \tag{14.1}$$

Note that cardinalities $|A|$ and $|B|$ can be easily retrieved from respective IBLTs T_A and T_B by summing all counter values and dividing by r.

14.2.4 IBLT dimensioning with syncmers

Dimensioning an IBLT holding syncmers requires estimating the expected number of differences in the set difference of involved k-mer sets. Assuming that input datasets are close genomic sequences of size L related by a mutation rate bounded by p_m and that k is sufficiently large so that k-mer occurrences are unique, we can estimate the set difference. Each mutation results in $2k$ k-mers in the set difference (k k-mers on each side), and therefore the size of set difference is estimated to be $2kp_mL$. Taking into account density $\frac{2}{k-z+1}$ of syncmers (Section 14.1.3), we obtain the estimation

$$n = \frac{4kLp_m}{k-z+1}. \tag{14.2}$$

14.2.5 Approximating k-mer set differences

The method of Section 14.2.3 allows estimating Jaccard similarity on k-mers by Jaccard similarity on syncmers. Here we describe how we can extend these ideas in order to recover *all* k-mers from $K(S_1) \setminus K(S_2)$ and $K(S_2) \setminus K(S_1)$, where S_1, S_2 are input datasets and $K(S)$ denotes the set of k-mers of a dataset S.

Note first that a straightforward way of doing this, through IBLTs of $K(S_1)$ and $K(S_2)$, requires a considerable space because a single mutation generates a difference of k k-mers. Using syncmers, we can "pack" k-mers into longer strings, compute the differences and then recover k-mers from them. The set of recovered k-mers, however, will be a superset of exact differences.

To achieve this, instead of storing syncmers, we store in IBLTs *extended* syncmers of length $2k - z$. Extended syncmers are obtained by extending each syncmer to the right by $k - z$ bases. Since successive syncmers overlap by at least z bases, this ensures that each k-mer belongs to at least one extended syncmer.

By applying the IBLT difference technique (Section 14.2.3), we obtain the extended syncmers that differ between the two datasets, from which we extract k-mers and discard those shared between the two obtained sets. It may still happen that the sets we obtain are *supersets* of exact differences, due to the fact that an extended syncmer can contain a k-mer which belongs to another extended syncmer common to both datasets. However, we state that for a sufficiently large k, the fraction of common k-mers in those sets will be small enough, which we illustrate experimentally in Section 14.3.5. In the extreme case where each k-mer occurs once, our method computes exact k-mer set differences.

14.2.6 IBLT for collections of MinHash sketches

Another application of IBLTs is the reconciliation of whole sequence databases by storing whole MinHash sketches inside IBLTs. MinHash sketches are used as a proxy to detect unique sequences in each dataset so that only them need to be synchronized. In particular, sequence reconciliation can be useful to synchronize multiple databases of reference genomes used by targeted alignments. For example, in Metalign [107] metagenomes are aligned to multiple references selected from a given database. Multiple copies of the same database can be enriched by their users with new sequences that need to be shared to maintain synchronization. On the other hand, having too many similar sequences to choose from would negatively affect performances.

Transforming sequences into MinHash sketches is an elegant way to cluster them together with sensitivity controlled by the MinHash parameter s. Multiple strains or similar organism tend to have the same MinHash sketches for s small-enough. After reducing the multiset of MinHash sketches to a simple set, IBLTs can provide further compression and the ability to quickly retrieve differences.

We do not provide results for this kind of application, since it naturally follows from the exactness of IBLT listing. The only difference compared to storing sequences is that each bucket must allocate enough space for whole MinHash sketches.

14.3 Results

Our IBLT implementation is available at `https://github.com/yhhshb/km-peeler.git`. Buckets do not store an explicit hash field, following the idea presented in Section 14.2.2. Furthermore, we split the m buckets into r sub-tables following the implementation of [69]. Slice j is indexed by hash function $h_j(\cdot)$. Each k-mer (syncmer) coming from simulated sequences (sections 14.3.1 and 14.3.2) is stored as-is, without further processing. On

the other hand, experiments involving real-world sequences transform k-mers into their canonical form before any other operation (MinHash sketching, sampling, insertion into IBLTs).

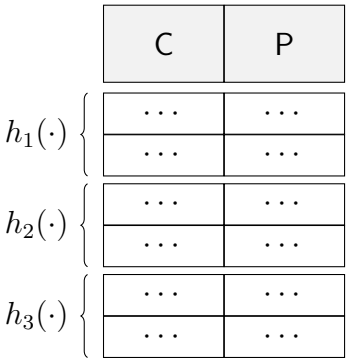


Figure 14.3: Unlike the IBLT presented in Figure 5.4, our implementation ignores hash field H. Furthermore, tables are split into r independent slices following the analysis of [69].

14.3.0.1 Datasets

The datasets used in this study are:

- `covid`: subsample of 50 *SARS-CoV-2* genomes[1]. Sequence names are provided in Table 16.1.
- `spneu`: subsample of 28 *Streptococcus Pneumoniae* genomes from [30] whose names are reported in Table 16.2. The subsample has been chosen to contain very close strains, with pairwise mutation rates between them not exceeding 0.0005.

14.3.1 Comparison of different sampling approaches

Random sampling, minimizers and syncmers have been compared by computing Jaccard similarities between pairs of synthetic sequences. Each pair is constructed by first generating a uniform random sequence of length L and then mutating it through independent substitutions. Points in Figures 14.4 are averages over $T = 500$ independent trials. For fairness of comparison, parameters for uniform sampling, minimizers and syncmers have been chosen to guarantee the same sampling rate $1/\nu$. We know that $c_s \approx 2/(k - z + 1)$, $c_m \approx 2/(w + 1)$ and $c_{ns} \approx 1/\nu$ are the densities of syncmers, minimizers and random sampling, respectively. Thus, given parameters k and z, setting the minimizer window length as $w = k - z$ and choosing a sampling rate $1/\nu = c_s$ ensures about the same number of sampled k-mers for all algorithms. As Figure 14.4 shows, syncmers do not have the previously reported biased behavior of minimizers [12], but they are statistically similar to random sampling. Nevertheless, we choose syncmer sampling as a way to reduce IBLT memory usage in Section 14.3.3 in anticipation of future applications that might take advantage of the bounded distance guarantee.

14.3.2 Space performance of IBLTs

In order to demonstrate the space efficiency of IBLTs in our framework, we compare them against a solution based on KMC k-mer counting software [101]. KMC provides an

[1] https://www.ncbi.nlm.nih.gov/datasets/coronavirus/genomes/

efficient way for storing, manipulating and querying sets of k-mers. Unlike other counting tools (Jellyfish [135] or DSK [165]), KMC allows easy sorting of its output which leads to an efficient way to compute Jaccard similarity.

We compared memory taken by IBLTs vs. KMC databases for storing syncmers issued from two similar sequences. For this, we applied the same procedure as in Section 14.3.1: mutating a random sequence of length L with mutation probability p_m. Sampled syncmers from both sequences are stored respectively in IBLTs and KMC databases. Figure 14.5 reports average space taken by the two data structures. Each bar is the average over $T = 100$ trials, except for case $L = 10$M for which $T = 10$. IBLTs were dimensioned (see (5.2)) to guarantee peelability of all T sketches with high probability.

Figure 14.5a clearly demonstrates the advantage of IBLTs when the mutation rate is small. For larger p_m and long sequences, the number of differences reach a point where exact data structures become preferable, as illustrated by Figure 14.5b for $p_m = 0.01$ and sequences of length 10M.

In our experiments, subtracting one IBLTs from another is dominated by the time taken to load/save the sketches, and not by performing the actual difference. Even in more complex scenarios, subtraction remains a very simple operation that can be performed by accessing one bucket at a time in any given order. On the other hand, the amount of time required by listing the content of an IBLT varies greatly and depends on the set of items stored in it.

14.3.3 Accuracy of Jaccard similarity estimation from IBLTs of syncmers

Figures 14.6 and 14.7 report comparisons of both IBLTs and MinHash sketches on `covid` and `spneu` datasets respectively. Both plots show the average absolute error of Jaccard estimate computed over all pairs of sequences of the respective dataset. Exact Jaccard similarities computed over the full k-mer sets are used as ground truth. MinHash sketches (line MinHash in the plots) were implemented using Mash [148]. All sketch sizes (in bytes) are fixed beforehand with both MinHash sketches and IBLTs dimensioned accordingly in order to fit the allocated memory. The number of bits allocated for payload field P in our IBLT implementation is set to be the minimum multiple of 8 larger than or equal to $2k$. As Mash [148] uses 32- or 64-bit hashes, we used $k = 15$ in our experiments in order to force both methods to use 32-bit representations.

In all experiments, IBLTs storing syncmers (line syncmers + IBLT) showed the best precision. For covid genomes (Figure 14.6), full MinHash sketches become competitive for larger sketch sizes. Unlike MinHash, the average error of IBLTs remains constant across all reported cases because over-dimensioning only increases the probability of successful listing. For the `spneu` dataset (Figure 14.7), MinHash errors are about twice those of IBLTs across all allocated sketch sizes confirming that IBLTs are more memory-efficient. The general conclusion is that if sequences to be compared are highly similar, IBLTs storing syncmers are more efficient than MinHash sketches, with the latter being better suited to quickly provide an overview over more diverging datasets.

14.3.4 Sampling syncmers for further space reductions

Since syncmer sampling rate ($\frac{2}{k-z+1}$) cannot be made arbitrarily small for a given k, we also tested the effect of additional downstream sampling of syncmers, before inserting them into IBLTs. To this end, Figure 14.8 reports a comparison of syncmers sampled with different sampling rates $1/\nu$. We observe that downstream sampling of syncmers

comes at the cost of decreased precision for both datasets (Figure 14.8a and 14.8b), but it might be useful to further reduce space.

The effect on IBLT space of further sampling syncmers has been explored on the `covid` dataset with sampling rates $1/\nu = 0.5, 0.25, 0.125, 0.0625$. Results are shown in Figure 14.9. All pairwise sketch differences are guaranteed to be always peelable. Decreasing the sampling rate leads to reduced sketch sizes since fewer items need to be inserted inside IBLTs. However, starting at $\eta = 8$ sampling starts to be decreasingly effective, with IBLTs that are not half the size of their predecessor. This phenomenon can be explained by taking into account the suppression of hash field H explained in Section 14.2.2. As a direct consequence of this choice, spurious counters arising from collisions are detected with probability that depends on the number of buckets in the sketch. For large sampling rates, IBLTs can be too small to successfully recover from collisions. The only available solution in this case is to over dimension the sketch nullifying the advantages of sampling. Thus, for very small input sets, it is advisable to use an IBLT implementation with explicit hash field in its buckets.

14.3.5 Approximating k-mer set differences

We tested the method from Section 14.2.5 of approximating k-mer set differences on both the `covid` dataset and on two random datasets. Each random dataset contains 50 sequences of length 30000 obtained by first generating a uniform random sequence which is then mutated 49 times using a mutation probability p_m.

Recall that the method of Section 14.2.5 allows one to compute a *superset* of the symmetric difference $(K(S_1) \setminus K(S_2)) \cup (K(S_2) \setminus K(S_1))$ of sets of k-mers occurring in datasets S_1 and S_2. Here we measure the precision of this method, that is the number of spurious k-mers found by the algorithm. Those are k-mers actually belonging to $K(S_1) \cap K(S_2)$ but output by the algorithm as if they belong to $(K(S_1) \setminus K(S_2)) \cup (K(S_2) \setminus K(S_1))$.

Table 14.1 summarizes the experiments. Columns 'diff' and 'err' show the average/maximum cardinality of the true set difference and spurious k-mers, respectively, over all pairs of sequences. In the case of random datasets, sequences were generated with mutation probabilities $p_m = 0.01$ and $p_m = 0.001$.

	`covid`		random			
			$p_m = 0.001$		$p_m = 0.01$	
	diff	err	diff	err	diff	err
average	325.29	11.03	1708.78	60.03	15342.81	357.57
max	661	31	2396	110	17047	486

Table 14.1: True size of symmetric difference of k-mer sets and its overestimate. For each experiment, 'diff' is the average/maximum size of the true symmetric difference, and 'err' is the average/maximum number of spurious k-mers reported as being in the symmetric difference. p_m is the mutation probability used to generate sequences from a random one.

We observe that the number of spurious k-mers remains small, on average within about 3% of the true set difference size.

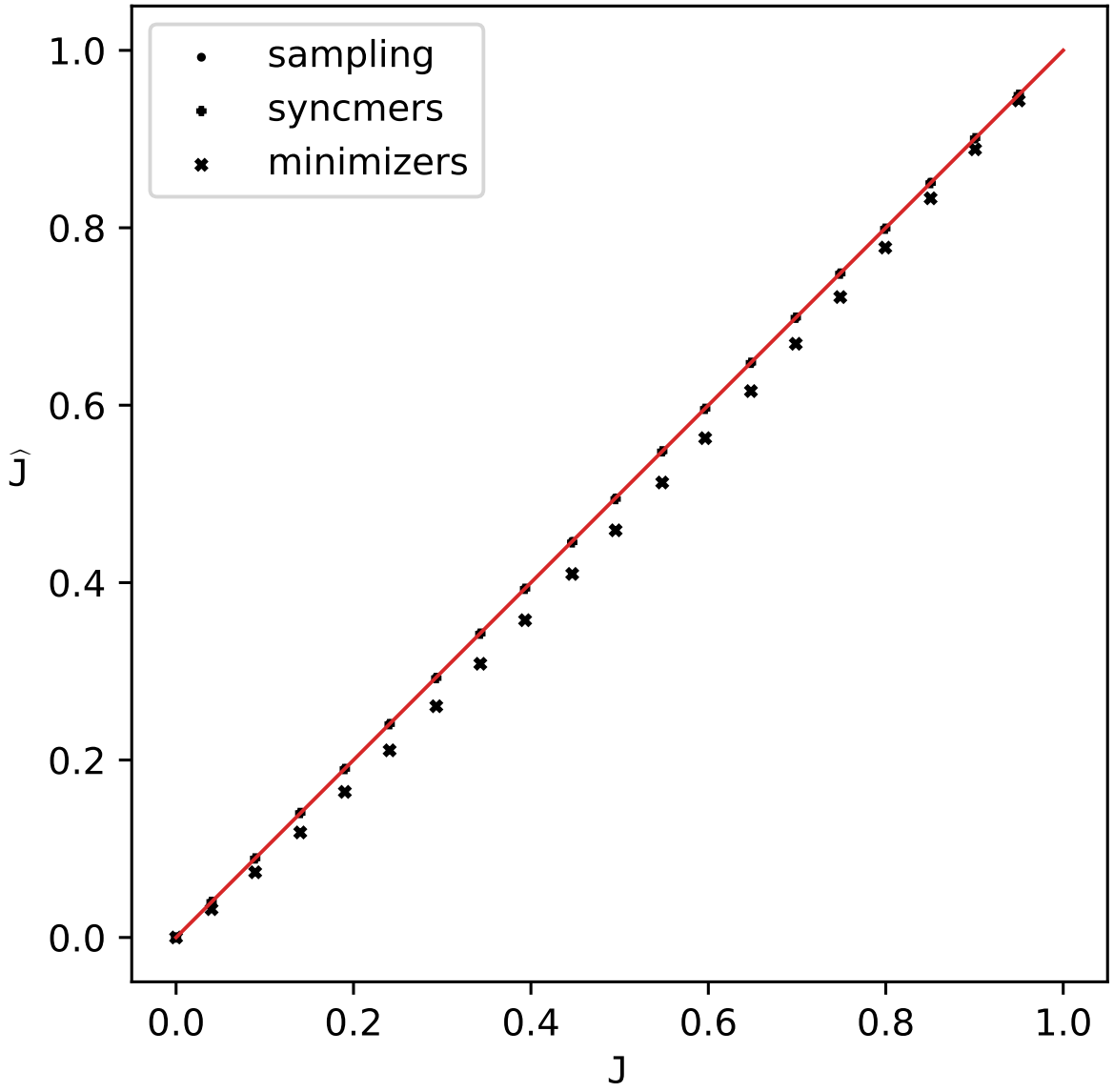
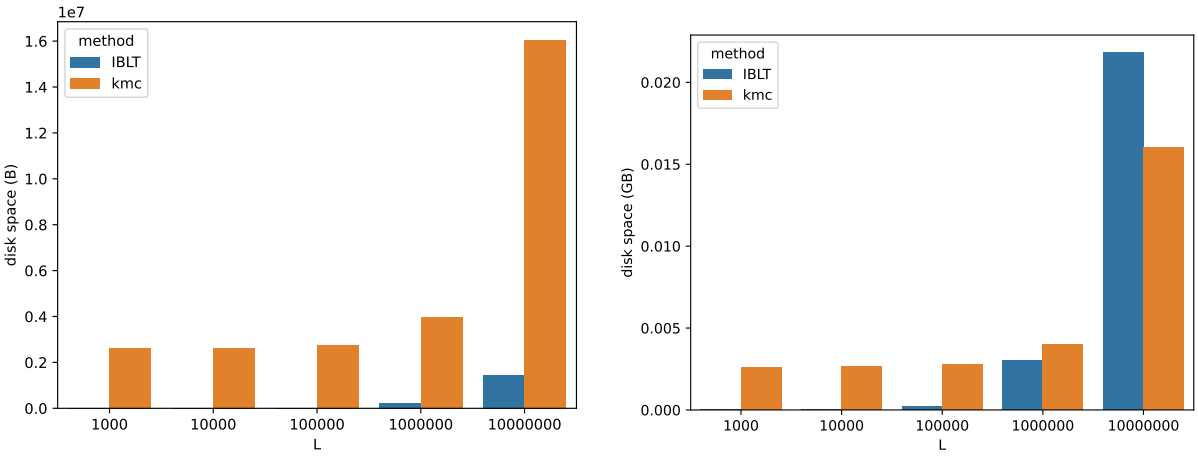


Figure 14.4: Minimizers present a non-negligible bias as opposed to syncmers and random sampling which are unbiased (and overlap in the plot). Each measurement was repeated 500 times on random sequences of length $L = 10\text{K}$ with $k = 15$, $w = 11$ (for minimizers) and $z = 4$ (for syncmers). Sampling rate is given by $1/\nu = 2/(k - z + 1) = 1/6$.



(a) $p_m = 0.001$ (b) $p_m = 0.01$

Figure 14.5: Space taken by IBLTs depends on the similarity between stored sets. For very similar sequences (mutation rate $p_m = 0.001$, Figure 14.5a), IBLTs are more efficient than KMC. Their advantage appears reduced for increased p_m and large sequences (Figure 14.5b).

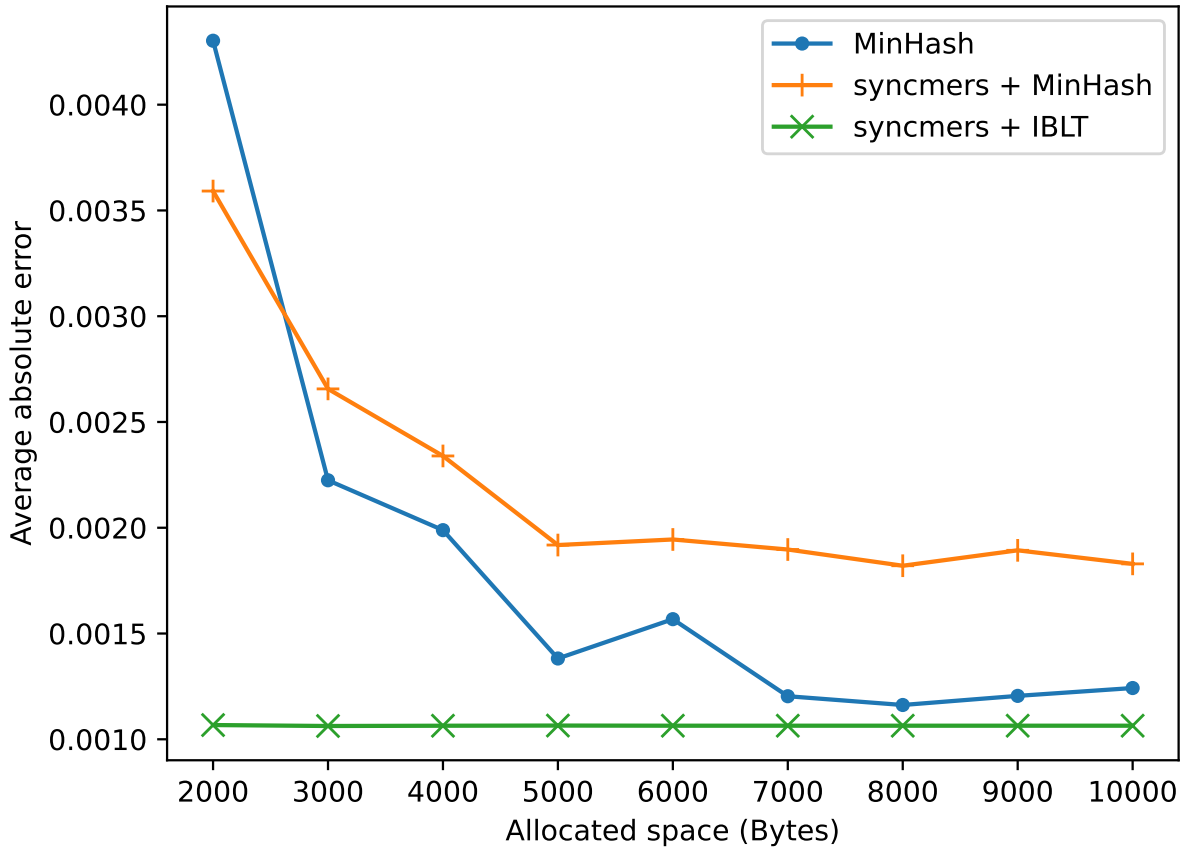


Figure 14.6: Comparison between IBLTs and MinHash for computing pairwise Jaccard on the `covid` dataset. The x-axis reports the amount of space allocated for each sketch while the y-axis reports the average absolute error. $k = 15$ and $z = 4$ in all tests. Sketch size for MinHash and table size for IBLTs are chosen to fit the allocated memory.

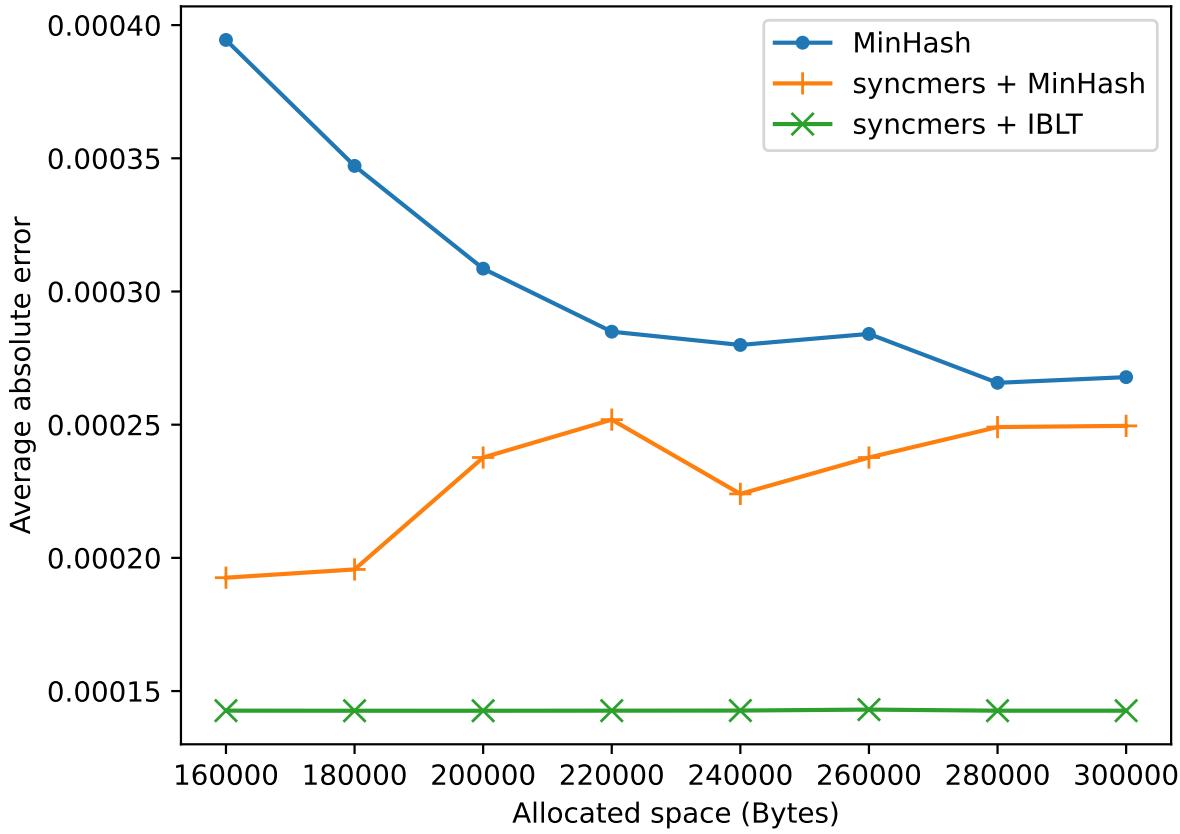
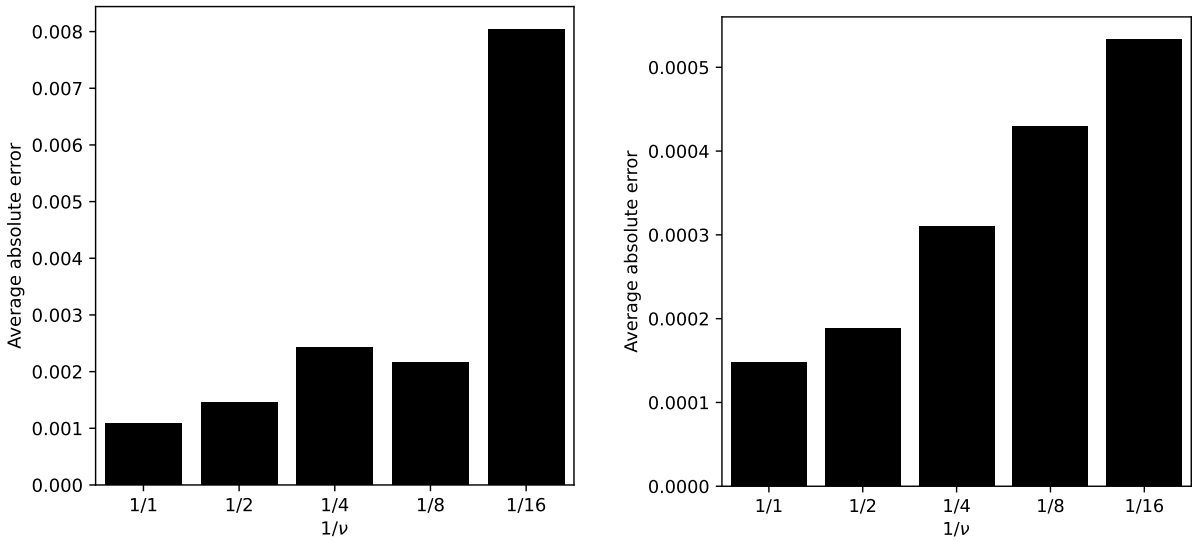


Figure 14.7: Comparison between IBLTs and MinHash for computing pairwise Jaccard on the `spneu` dataset with the same setting as Figure 14.6.



(a) `covid` dataset

(b) `spneu` dataset

Figure 14.8: Effect of sampling syncmers before IBLT insertion on the average absolute error. $1/\nu$ is the compression rate used for sampling syncmer sets before IBLT insertion. $\nu = 1$ means no sampling (full syncmer sets).

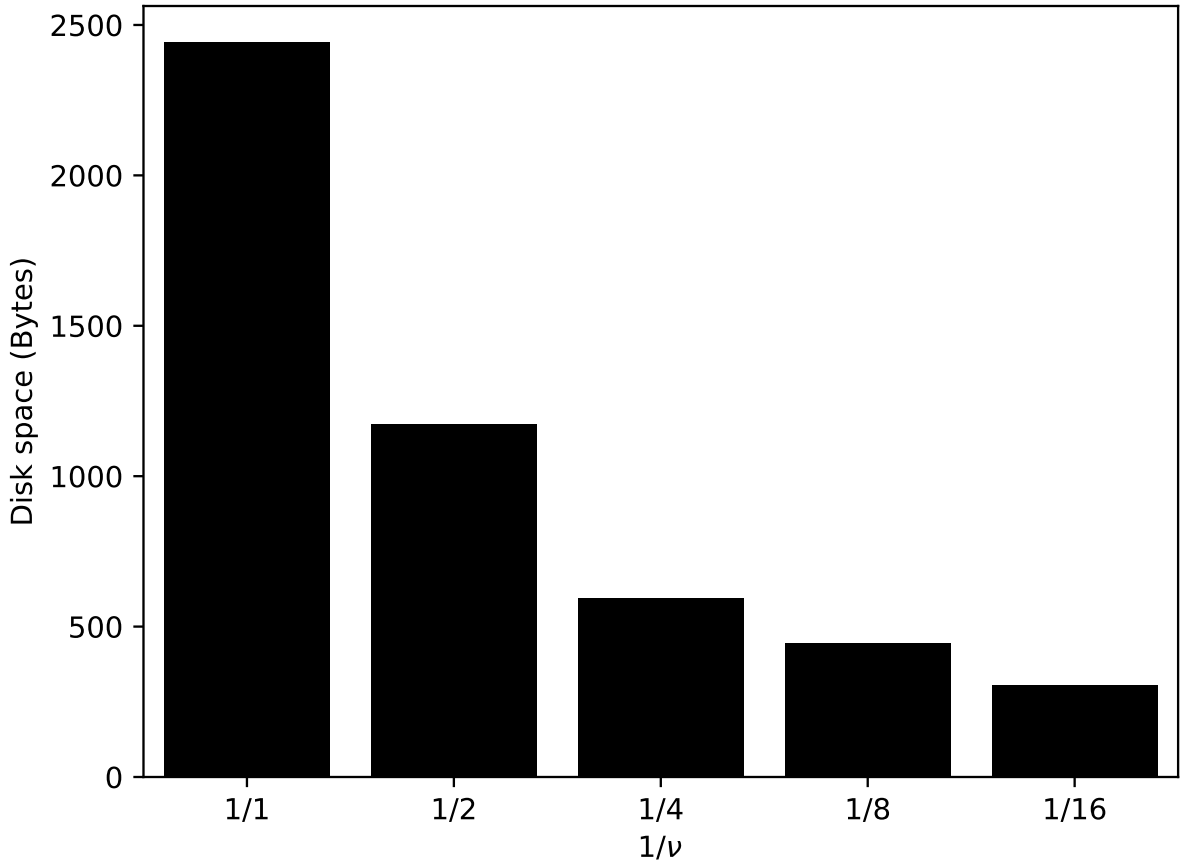


Figure 14.9: IBLT size when using syncmers ($k = 15$, $z = 4$) combined with sampling. Additional sampling helps in reducing IBLT space at the cost of additional errors as seen in Figure 14.8. However, not storing hash filed H imply diminishing returns for compression ratios > 4 since recognizing spurious buckets becomes harder (as described in Section 14.2.2).

Chapter 15

Discussion

We showed that whenever involved datasets are similar enough and their similarity can be bounded *a priori*, IBLTs lead to a more space-efficient and, at the same time, more accurate method for estimating Jaccard similarity of underlying k-mer sets. This is achieved by combining IBLTs with syncmers as a means to sampling k-mers and further reduce the space of our data structures. As opposed to minimizers, syncmers provide an unbiased estimator of Jaccard index, which was confirmed in our experiments. Thus, IBLTs combined with syncmers constitute a powerful alternative to MinHash sketching for estimating Jaccard similarity of similar datasets.

Various bioinformatics fields where similar sequences are the norm can potentially benefit from IBLTs. In pan-genomics, for example, IBLTs could not only provide a more space-efficient similarity estimations between new sequences and references but also a way to quickly find the involved differences. This is possible thanks to the exact nature of successful listing operations. k-mers found in the symmetric difference can be used as seeds to quickly find the divergent positions between sequences. This opens up new possibilities as k-mers belonging to a dataset can be used to infer information about genetic variation, specific mutation, etc. To this end, we extended our method to return exact symmetric difference supersets of whole k-mer sets. We believe this method can be extended to compute *exact* set differences with the help of additional space-efficient data structures and plan to explore this in our future work.

Our ideas may have further useful applications, for example to reconciliation of datasets located on remote computers, in which case IBLTs could avoid transmitting whole collections of strings. Another example is a selection of sufficiently diverse datasets avoiding redundancy (see Section 14.2.6). Note finally that IBLTs may also act as filters for filtering out dissimilar datasets: in this case, non-peelability of the difference IBLT is an indicator of dissimilarity.

Chapter 16

Appendix

Datasets used in experiments

Table 16.1: Names of covid genomes used for Figure 14.6

BS001151.1	LR877722.1	LR883214.1	MT520216.1
MT706180.1	MT757082.1	MT800758.1	MT834020.1
MT970159.1	MT971010.1	MT973151.1	MW064390.1
MW064919.1	MW064981.1	MW153809.1	MW153954.1
MW154711.1	MW156712.1	MW184416.1	MW184648.1
MW190904.1	MW190957.1	MW191020.1	MW191146.1
MW206148.1	MW276931.1	MW321243.1	MW321430.1
MW593629.1	MW631874.1	MW669599.1	MW681303.1
MW681489.1	MW693959.1	MW696216.1	MW702101.1
MW708072.1	MW708184.1	MW708826.1	MW720341.1
MW733722.1	MW738615.1	MW749542.1	MW776764.1
MW820211.1	MW850083.1	MW863243.1	MW868532.1
MW868533.1	MW871079.1		

Table 16.2: Names of *S.Pneumoniae* genomes used for Figure 14.7

BZ2I7.fa	R34-3087.fa	007649.fa	R34-3097.fa
4PYM0.fa	JBYFY.fa	T8Z8O.fa	R34-3044.fa
O61U7.fa	81LMX.fa	O0RHB.fa	R34-3083.fa
R34-3025.fa	WAMFH.fa	O8I1E.fa	R34-3164.fa
CCV1H.fa	0U64I.fa	6893Z.fa	1VDX8.fa
R34-3074.fa	R34-3227.fa	LS3OB.fa	UTEDZ.fa
REAOU.fa	R34-3229.fa	067094.fa	4K4C9.fa

Conclusion

In this thesis: i) we designed two memory-efficient representations of k-mer counts which can be combined with third-party k-mer set representation to serve as more efficient alternatives to k-mer count tables, and ii) we studied the problem of quickly computing k-mer sets differences with applications to Jaccard similarity estimation.

The first problem was addressed in **Part II** and **Part III**.

Part II introduced *Set-Min sketch*, a novel sketching technique geared towards counts. Inspired by Count-Min sketch we demonstrate how to take better advantage of the typical skewed distribution of k-mer counts by replacing each counter with a set of elements. By storing distinct count values in each cell, our sketch re-use multiple times the most common elements for better memory efficiency. At query time, sets belonging to cells associated to the query are intersected. If there are no collisions (multiple elements in the intersection) the query returns the true value associated to the key, without errors. When collisions do occur, Set-Min sketch is nevertheless able to limit errors by once again taking advantage of the skewed distributions of k-mers. We make use of these insights in the theoretical analysis of Set-Min sketch (Chapter 8) where we bound (in expectation) the total cumulative error of our sketch and provide a practical dimensioning procedure.

We shifted our attention from sketching techniques to a combination of Locality Sensitive Hashing and succinct data structures in **Part III** where we present *Locom*. Locality Sensitive Hashing based on k-mer minimizers is a way to bucket similar counts together, by taking advantage of the observation that similar or successive k-mers tend to appear the same number of times. Compared to Set-Min sketch, whose bucketing is completely random, Locom's buckets are more likely to contain similar counts leading to better overall compression. Buckets are stored using Bloom-enhanced Compressed Static Functions (BCSFs), our modified version of the only available implementation of Compressed Static Functions available at the time of writing. Our enhancements remove the space lower-bound of 1 bit/key making BCSFs more suitable for representing skewed distributions than their predecessor. We present two variations of Locom: AMB and FIL, whose difference lies on how collisions are dealt with.

For the second problem we proposed a solution based on Invertible Bloom Lookup Tables (IBLTs) in **Part IV**. Our method to retrieve the symmetric set difference of two sets directly works with their IBLTs, without having to keep one set uncompressed. Further, we show how syncmers are unbiased estimators of Jaccard similarity unlike minimizers. For sequences of high similarity, the combination of these two results leads to a more memory efficient sketching technique than MinHash. Finally, we show how a slight modification of these ideas allows for efficient computation of supersets of *exact* k-mer set differences, at the cost of small post-processing.

In conclusion, this work opens new perspectives in designing efficient data structures and sketching techniques for representing counts associated to k-mers, as well as efficient methods tailored to the ever-increasing number of similar genomes sequenced each year. These objectives are achieved by taking advantage of the statistical characteristics of k-mers in order to produce highly adapted algorithms specifically designed for this kind of tasks.

Future work and perspectives

Various open problems and future directions appeared during the development of our methods. We hereby provide some perspective and ideas for expansion.

- **Frequency-aware phylogeny reconstruction.** As briefly mentioned in Chapter 4, frequency-aware alignment-free methods do not seem to be better than purely compositional ones in phylogeny reconstruction. The utility of k-mer frequencies for inferring biological relations between species has been questioned in [120] with experimental large scale benchmarks supporting this observation [224]. On the other hand, works like [89] show how multiplicities are helpful during mapping, as they allow the comparison of very repetitive regions. It is therefore of high interest to better understand the relation between these two, seemingly opposing findings. A preliminary question to this problem is how to use local, frequency-aware mappings in the more general settings of phylogeny reconstruction.

- **Further analysis of current count sketches under specific regimes.** We mentioned in Chapter 5.1.3 how Count-Min sketch [42] can achieve better error guarantees if the distribution of counts follow a Zipf distribution (Section 5.1.2). On the other hand, in Section 5.1.3 we mentioned the conservative update strategy of Count-Min sketch which we use in Part II to modify Set-Min sketches into Max-Mins. However, up until recently [64] no formal analysis of Count-Min sketch under conservative updates existed. Future directions can thus consist in providing formal analyses of useful modifications of already available sketching methods.

- **Other weighted schemes for k-mer sets.** While in this thesis we focused on efficient representations of k-mer counts, other weighting schemes for k-mers can be devised. An example of such alternative weightings was presented in Part III where k-mers were associated to the number of datasets they appeared in (document frequency). An obvious question is whether additional weighting schemes can benefit from our solutions of Part II and Part III. Additionally, since not all weighting schemes are guaranteed to be power-law distributed another interesting question is the development of efficient representations of counts distributed arbitrarily. A good direction toward this goal is to extend the idea of taking advantage of the high similarity of neighboring k-mers as we or [154] already do.

- **Additional Compressed Static Function implementations.** The only available CSF implementation available today is the one of [67], used by our methods in Part III. It is therefore desirable for alternative methods to appear in the near future. A wider choice not only benefits programs (CSFs in Locom can be swapped quite easily since they are used as black boxes), but also developers, who would be able to choose the best CSF to their problem (note that [67] only provides a Java implementation).

- **Compressed multi-dataset count representations.** All our count-aware methods focus on the representation of k-mer counts coming from one single dataset.

Nevertheless, representing k-mer multiplicities across multiple datasets is an interesting problem as it can be viewed as the natural extension of our ideas. The problem is not new by itself, with current solutions achieving succinct representations of counts by Run-Length-Encoding (RLE) [132] or using combinations of efficient binary matrix implementations and de Bruijn graphs [98]. However, effective extension of CSFs to multiple k-mer multisets is still an open problem left for further developments.

- **Invertible Bloom Lookup Tables for multisets.** Our IBLT implementation presented in Part IV is limited to simple sets. Future work will be dedicated to extension to multisets (that is sketching k-mer count tables). One interesting direction would be to merge IBLTs with Count-Min sketches in order to support peeling of weights similarly to Counter Braids [126].

- **IBLTs for seeding.** Related to the previous idea is how to extend IBLTs to work as fast methods to detect possible differences between two sequences. The current state of our implementation require to scan the original sequences to find the exact positions of the mismatches and possible false positives. A more efficient solution would be to augment IBLTs to store k-mer positions. This case is hindered by the need to deal with collisions resulting from the same k-mer appearing in different locations.

- **Other applications of IBLTs.** Instead of extending the current implementation of IBLTs to support other types of queries, further research directions can be discovered by asking: are there other applications where our IBLTs can be useful? A possible answer worth exploring is fast VCF file comparisons, especially between sets of SNPs (Single Nucleotide Polymorphisms).

Bibliography

[1] Omar Ahmed, Massimiliano Rossi, Sam Kovaka, Michael C. Schatz, Travis Gagie, Christina Boucher, and Ben Langmead. Pan-genomic matching statistics for targeted nanopore sequencing. *iScience*, 24(6):102696, June 2021.

[2] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st symposium on Principles of Database Systems - PODS '12*, page 5, Scottsdale, Arizona, USA, 2012. ACM Press.

[3] Stephen F. Altschul. Amino acid substitution matrices from an information theoretic perspective. *Journal of Molecular Biology*, 219(3):555–565, June 1991.

[4] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, October 1990.

[5] Sasha K. Ames, David A. Hysom, Shea N. Gardner, G. Scott Lloyd, Maya B. Gokhale, and Jonathan E. Allen. Scalable metagenomic taxonomy classification using a reference genome database. *Bioinformatics*, 29(18):2253–2260, September 2013.

[6] Sepehr Assadi, Sanjeev Khanna, and Yang Li. Tight bounds for single-pass streaming complexity of the set cover problem. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 698–711, Cambridge MA USA, June 2016. ACM.

[7] Daniel N. Baker and Ben Langmead. Dashing: fast and accurate genomic distances with HyperLogLog. *Genome Biology*, 20(1):265, December 2019.

[8] Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey A. Gurevich, Mikhail Dvorkin, Alexander S. Kulikov, Valery M. Lesin, Sergey I. Nikolenko, Son Pham, Andrey D. Prjibelski, Alexey V. Pyshkin, Alexander V. Sirotkin, Nikolay Vyahhi, Glenn Tesler, Max A. Alekseyev, and Pavel A. Pevzner. SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of Computational Biology: A Journal of Computational Molecular Cell Biology*, 19(5):455–477, May 2012.

[9] Yuwei Bao, Jack Wadden, John R. Erb-Downward, Piyush Ranjan, Weichen Zhou, Torrin L. McDonald, Ryan E. Mills, Alan P. Boyle, Robert P. Dickson, David Blaauw, and Joshua D. Welch. SquiggleNet: real-time, direct classification of nanopore signals. *Genome Biology*, 22(1):298, October 2021.

[10] Ran Ben Basat, Gil Einziger, Michael Mitzenmacher, and Shay Vargaftik. Faster and More Accurate Measurement through Additive-Error Counters. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pages 1251–1260, Toronto, ON, Canada, July 2020. IEEE.

[11] Djamal Belazzougui and Rossano Venturini. Compressed static functions with applications. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, SODA '13, pages 229–240, New Orleans, Louisiana, January 2013. Society for Industrial and Applied Mathematics.

[12] Mahdi Belbasi, Antonio Blanca, Robert S. Harris, David Koslicki, and Paul Medvedev. The minimizer jaccard estimator is biased and inconsistent. *bioRxiv*, 2022.

[13] Gaëtan Benoit, Mahendra Mariadassou, Stéphane Robin, Sophie Schbath, Pierre Peterlongo, and Claire Lemaitre. SimkaMin: fast and resource frugal de novo comparative metagenomics. *Bioinformatics (Oxford, England)*, 36(4):1275–1276, February 2020.

[14] 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.

[15] Guillaume Bernard, Cheong Xin Chan, and Mark A. Ragan. Alignment-free microbial phylogenomics under scenarios of sequence divergence, genome rearrangement and lateral genetic transfer. *Scientific Reports*, 6(1):28970, September 2016.

[16] 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.

[17] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 1970.

[18] O. Bonham-Carter, J. Steele, and D. Bastola. Alignment-free genetic sequence comparisons: a review of recent approaches by word analysis. *Briefings in Bioinformatics*, 15(6):890–905, November 2014.

[19] Alexander Bowe, Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. Succinct de Bruijn Graphs. In Ben Raphael and Jijun Tang, editors, *Algorithms in Bioinformatics*, Lecture Notes in Computer Science, pages 225–235, Berlin, Heidelberg, 2012. Springer.

[20] Robert S. Boyer and J. Strother Moore. MJRTY—A Fast Majority Vote Algorithm. In Robert S. Boyer, editor, *Automated Reasoning: Essays in Honor of Woody Bledsoe*, Automated Reasoning Series, pages 105–117. Springer Netherlands, Dordrecht, 1991.

[21] Phelim Bradley, Henk C Den Bakker, Eduardo P. C. Rocha, Gil McVean, and Zamin Iqbal. Ultra-fast search of all deposited bacterial and viral genomic data. *Nature biotechnology*, 37(2):152–159, February 2019.

[22] Lauren M. Bragg, Glenn Stone, Margaret K. Butler, Philip Hugenholtz, and Gene W. Tyson. Shining a Light on Dark Sequencing: Characterising Errors in Ion Torrent PGM Data. *PLOS Computational Biology*, 9(4):e1003031, April 2013.

[23] Nicolas L Bray, Harold Pimentel, Páll Melsted, and Lior Pachter. Near-optimal probabilistic RNA-seq quantification. *Nature Biotechnology*, 34(5):525–527, May 2016.

[24] Heinz Breu. A theoretical understanding of 2 base color codes and its application to annotation, error detection, and error correction. Technical report, Applied Biosystems, 2008.

[25] Karel Brinda. *Novel computational techniques for mapping and classification of Next-Generation Sequencing data*. Theses, Université Paris-Est, November 2016.

[26] 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.

[27] C. Titus Brown and Luiz Irber. sourmash: a library for MinHash sketching of DNA. *Journal of Open Source Software*, 1(5):27, September 2016.

[28] M Burrows and DJ Wheeler. Technical report 124. *Palo Alto, CA: Digital Equipment Corporation*, 1994.

[29] Karel Břinda, Michael Baym, and Gregory Kucherov. Simplitigs as an efficient and scalable representation of de Bruijn graphs. *bioRxiv*, page 2020.01.12.903443, January 2020.

[30] Karel Břinda, Alanna Callendrello, Kevin C. Ma, Derek R. MacFadden, Themoula Charalampous, Robyn S. Lee, Lauren Cowley, Crista B. Wadsworth, Yonatan H. Grad, Gregory Kucherov, Justin O'Grady, Michael Baym, and William P. Hanage. Rapid inference of antibiotic resistance and susceptibility by genomic neighbour typing. *Nature Microbiology*, 5(3):455–464, March 2020.

[31] Amit Chakrabarti and Anthony Wirth. Incidence Geometries and the Pass Complexity of Semi-Streaming Set Cover. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1365–1373. Society for Industrial and Applied Mathematics, January 2016.

[32] Diptarka Chakraborty, Elazar Goldenberg, and Michal Koucký. Streaming algorithms for embedding and computing edit distance in the low distance regime. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 712–725, Cambridge MA USA, June 2016. ACM.

[33] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theoretical Computer Science*, 312(1):3–15, January 2004.

[34] Rayan Chikhi and Guillaume Rizk. Space-efficient and exact de Bruijn graph representation based on a Bloom filter. *Algorithms for Molecular Biology*, 8(1):22, September 2013.

[35] Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via Sampling with Applications to Finding Matchings and Related Problems in Dynamic Graph Streams. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1326–1344. Society for Industrial and Applied Mathematics, January 2016.

[36] Benny Chor, David Horn, Nick Goldman, Yaron Levy, and Tim Massingham. Genomic dna k-mer spectra: models and modalities. *Genome Biology*, 10(10):R108, Oct 2009.

[37] Tobias Christiani. DartMinHash: Fast Sketching for Weighted Sets. Technical Report arXiv:2005.11547, arXiv, May 2020. arXiv:2005.11547 [cs] type: article.

[38] Thomas Conway, Jeremy Wazny, Andrew Bromage, Justin Zobel, and Bryan Beresford-Smith. Gossamer–a resource-efficient de novo assembler. *Bioinformatics (Oxford, England)*, 28(14):1937–1938, July 2012.

[39] Graham Cormode. Count-Min Sketch. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, pages 511–516. Springer US, Boston, MA, 2009.

[40] Graham Cormode and Marios Hadjieleftheriou. Finding the frequent items in streams of data. *Communications of the ACM*, 52(10):97–105, October 2009.

[41] Graham Cormode and Muthu Muthukrishnan. Approximating Data with the Count-Min Sketch. *IEEE Software*, 29(1):64–69, January 2012.

[42] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, April 2005.

[43] Graham Cormode and S. Muthukrishnan. Summarizing and Mining Skewed Data Streams. In *Proceedings of the 2005 SIAM International Conference on Data Mining*, pages 44–55. Society for Industrial and Applied Mathematics, April 2005. 5th SIAM International Conference on Data Mining, SDM 2005 ; Conference date: 21-04-2005 Through 23-04-2005.

[44] M. Csűrös, L. Noé, and G. Kucherov. Reconsidering the significance of genomic word frequencies. *Trends in Genetics*, 23(11):543–546, November 2007.

[45] Dan DeBlasio, Fiyinfoluwa Gbosibo, Carl Kingsford, and Guillaume Marçais. Practical universal k-mer sets for minimizer schemes. In *Proceedings of the 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, BCB '19, page 167–176, New York, NY, USA, 2019. Association for Computing Machinery.

[46] Clara Delahaye and Jacques Nicolas. Sequencing DNA with nanopores: Troubles and biases. *PLOS ONE*, 16(10):e0257521, October 2021.

[47] A. L. Delcher. Fast algorithms for large-scale genome alignment and comparison. *Nucleic Acids Research*, 30(11):2478–2483, June 2002.

[48] A. L. Delcher, S. Kasif, R. D. Fleischmann, J. Peterson, O. White, and S. L. Salzberg. Alignment of whole genomes. *Nucleic Acids Research*, 27(11):2369–2376, January 1999.

[49] Luca Denti, Marco Previtali, Giulia Bernardini, Alexander Schönhuth, and Paola Bonizzoni. MALVA: Genotyping by Mapping-free ALlele Detection of Known VAriants. *iScience*, 18:20–27, August 2019.

[50] Martin Dietzfelbinger, Andreas Goerdt, Michael Mitzenmacher, Andrea Montanari, Rasmus Pagh, and Michael Rink. Tight thresholds for cuckoo hashing via xorsat. In *Proceedings of the 37th International Colloquium Conference on Automata, Languages and Programming*, ICALP'10, page 213–225, Berlin, Heidelberg, 2010. Springer-Verlag.

[51] R. C. Edgar. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32(5):1792–1797, March 2004.

[52] Robert Edgar. Syncmers are more sensitive than minimizers for selecting conserved k-mers in biological sequences. *PeerJ*, 9:e10805, February 2021.

[53] Barış Ekim, Bonnie Berger, and Rayan Chikhi. Minimizer-space de Bruijn graphs: Whole-genome assembly of long reads in minutes on a personal computer. *Cell Systems*, 12(10):958–968.e6, October 2021.

[54] Barış Ekim, Bonnie Berger, and Yaron Orenstein. A Randomized Parallel Algorithm for Efficiently Finding Near-Optimal Universal Hitting Sets. In Russell Schwartz, editor, *Research in Computational Molecular Biology*, Lecture Notes in Computer Science, pages 37–53, Cham, 2020. Springer International Publishing.

[55] David Eppstein and Michael T. Goodrich. Straggler identification in round-trip data streams via Newton's identities and invertible Bloom filters. *IEEE Transactions on Knowledge and Data Engineering*, 23(2):297–306, 2011.

[56] Otmar Ertl. SuperMinHash - A New Minwise Hashing Algorithm for Jaccard Similarity Estimation. Technical Report arXiv:1706.05698, arXiv, June 2017. arXiv:1706.05698 [cs] type: article.

[57] Emmanuel Esposito, Thomas Mueller Graf, and Sebastiano Vigna. RecSplit: Minimal Perfect Hashing via Recursive Splitting. In *2020 Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX)*, Proceedings, pages 175–185. Society for Industrial and Applied Mathematics, December 2019.

[58] Cristian Estan and George Varghese. New directions in traffic measurement and accounting. In Matthew Mathis, Peter Steenkiste, Hari Balakrishnan, and Vern Paxson, editors, *Proceedings of the ACM SIGCOMM 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, August 19-23, 2002, Pittsburgh, PA, USA*, pages 323–336. ACM, 2002.

[59] 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.

[60] Huan Fan, Anthony R. Ives, Yann Surget-Groba, and Charles H. Cannon. An assembly and alignment-free method of phylogeny reconstruction from next-generation sequencing data. *BMC Genomics*, 16(1):522, July 2015.

[61] P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 390–398, Redondo Beach, CA, USA, 2000. IEEE Comput. Soc.

[62] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. In *In Aofa '07: Proceedings of the 2007 International Conference on Analysis of Algorithms*, 2007.

[63] Edward A. Fox, Qi Fan Chen, and Lenwood S. Heath. A faster algorithm for constructing minimal perfect hash functions. In *In Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Data Structures*, pages 266–273, 1992.

[64] Éric Fusy and Gregory Kucherov. Phase transition in count approximation by Count-Min sketch with conservative updates, July 2022. arXiv:2203.15496 [cs].

[65] Shea N. Gardner and Barry G. Hall. When Whole-Genome Alignments Just Won't Work: kSNP v2 Software for Alignment-Free SNP Discovery and Phylogenetics of Hundreds of Microbial Genomes. *PLoS ONE*, 8(12):e81760, December 2013.

[66] Shea N Gardner, Tom Slezak, and Barry G. Hall. kSNP3.0: SNP detection and phylogenetic analysis of genomes without genome alignment or reference genome: Table 1. *Bioinformatics*, 31(17):2877–2878, September 2015.

[67] Marco Genuzio, Giuseppe Ottaviano, and Sebastiano Vigna. Fast scalable construction of ([compressed] static | minimal perfect hash) functions. *Information and Computation*, 273:104517, August 2020.

[68] Sante Gnerre, Iain MacCallum, Dariusz Przybylski, Filipe J. Ribeiro, Joshua N. Burton, Bruce J. Walker, Ted Sharpe, Giles Hall, Terrance P. Shea, Sean Sykes, Aaron M. Berlin, Daniel Aird, Maura Costello, Riza Daza, Louise Williams, Robert Nicol, Andreas Gnirke, Chad Nusbaum, Eric S. Lander, and David B. Jaffe. High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proceedings of the National Academy of Sciences*, 108(4):1513–1518, January 2011.

[69] Michael T. Goodrich and Michael Mitzenmacher. Invertible Bloom lookup tables, 2011.

[70] Sara Goodwin, John D. McPherson, and W. Richard McCombie. Coming of age: ten years of next-generation sequencing technologies. *Nature Reviews Genetics*, 17(6):333–351, June 2016.

[71] Osamu Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162(3):705–708, December 1982.

[72] Sudipto Guha, Andrew McGregor, and David Tench. Vertex and Hyperedge Connectivity in Dynamic Graph Streams. In *Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 241–247, Melbourne Victoria Australia, May 2015. ACM.

[73] Gaurav Gupta, Minghao Yan, Benjamin Coleman, R. A. Leo Elworth, Todd Treangen, and Anshumali Shrivastava. Sub-linear Sequence Search via a Repeated And Merged Bloom Filter (RAMBO): Indexing 170 TB data in 14 hours. *arXiv:1910.04358 [cs, q-bio]*, December 2019. arXiv: 1910.04358.

[74] Bernhard Haeupler, Mark Manasse, and Kunal Talwar. Consistent Weighted Sampling Made Fast, Small, and Easy. Technical Report arXiv:1410.4266, arXiv, October 2014. arXiv:1410.4266 [cs] type: article.

[75] Torben Hagerup and Torsten Tholey. Efficient Minimal Perfect Hashing in Nearly Minimal Space. In Afonso Ferreira and Horst Reichel, editors, *STACS 2001*, Lecture Notes in Computer Science, pages 317–326, Berlin, Heidelberg, 2001. Springer.

[76] Sariel Har-Peled, Piotr Indyk, Sepideh Mahabadi, and Ali Vakilian. Towards Tight Bounds for the Streaming Set Cover Problem. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 371–383, San Francisco California USA, June 2016. ACM.

[77] Klas Hatje and Martin Kollmar. A Phylogenetic Analysis of the Brassicales Clade Based on an Alignment-Free Sequence Comparison Method. *Frontiers in Plant Science*, 3, August 2012.

[78] Bernhard Haubold, Nora Pierstorff, Friedrich Möller, and Thomas Wiehe. Genome comparison without alignment using shortest unique substrings. *BMC Bioinformatics*, 6(1):123, May 2005.

[79] S Henikoff and J G Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915–10919, November 1992.

[80] Desmond G. Higgins and Paul M. Sharp. CLUSTAL: a package for performing multiple sequence alignment on a microcomputer. *Gene*, 73(1):237–244, December 1988.

[81] Bryan W. Holland, Norbert Kučerka, and D. Peter Tieleman. SIMtoEXP: Software for Comparing Simulations to Experimental Scattering Data. *Biophysical Journal*, 106(2):384a, January 2014.

[82] 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.

[83] Mark Hollmer. Roche to close 454 life sciences as it reduces gene sequencing focus. Fierce Biotech, 2013.

[84] Human Microbiome Project Consortium. Structure, function and diversity of the healthy human microbiome. *Nature*, 486(7402):207–214, June 2012.

[85] Sergey Ioffe. Improved Consistent Sampling, Weighted Minhash and L1 Sketching. In *2010 IEEE International Conference on Data Mining*, pages 246–255, December 2010. ISSN: 2374-8486.

[86] Giuseppe F. Italiano, Nicola Prezza, Blerina Sinaimeri, and Rossano Venturini. Compressed Weighted de Bruijn Graphs. In Pawe13 Gawrychowski and Tatiana Starikovskaya, editors, *32nd Annual Symposium on Combinatorial Pattern Matching (CPM 2021)*, volume 191 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:16, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[87] Chirag Jain, Alexander Dilthey, Sergey Koren, Srinivas Aluru, and Adam M. Phillippy. A Fast Approximate Algorithm for Mapping Long Reads to Large Reference Databases. In S. Cenk Sahinalp, editor, *Research in Computational Molecular Biology*, Lecture Notes in Computer Science, pages 66–81, Cham, 2017. Springer International Publishing.

[88] Chirag Jain, Sergey Koren, Alexander Dilthey, Adam M Phillippy, and Srinivas Aluru. A fast adaptive algorithm for computing whole-genome homology maps. *Bioinformatics*, 34(17):i748–i756, September 2018.

[89] Chirag Jain, Arang Rhie, Nancy F. Hansen, Sergey Koren, and Adam M. Phillippy. Long-read mapping to repetitive reference sequences using Winnowmap2. *Nature Methods*, 19(6):705–710, June 2022.

[90] Chirag Jain, Arang Rhie, Haowen Zhang, Claudia Chu, Brian P Walenz, Sergey Koren, and Adam M Phillippy. Weighted minimizer sampling improves long read mapping. *Bioinformatics*, 36(Supplement_1):i111–i118, July 2020.

[91] Chirag Jain, Luis M. Rodriguez-R, Adam M. Phillippy, Konstantinos T. Konstantinidis, and Srinivas Aluru. High throughput ANI analysis of 90K prokaryotic genomes reveals clear species boundaries. *Nature Communications*, 9(1):5114, November 2018.

[92] Se-Ran Jun, Gregory E. Sims, Guohong A. Wu, and Sung-Hou Kim. Whole-proteome phylogeny of prokaryotes by feature frequency profiles: An alignment-free method with optimal feature resolution. *Proceedings of the National Academy of Sciences*, 107(1):133–138, January 2010.

[93] Scott D. Kahn. On the Future of Genomic Data. *Science*, 331(6018):728–729, February 2011.

[94] M. Kapralov, Y. T. Lee, C. N. Musco, C. P. Musco, and A. Sidford. Single Pass Spectral Sparsification in Dynamic Streams. *SIAM Journal on Computing*, 46(1):456–477, January 2017.

[95] Mikhail Karasikov, Harun Mustafa, Daniel Danciu, Marc Zimmermann, Christopher Barber, Gunnar Rätsch, and André Kahles. MetaGraph: Indexing and Analysing Nucleotide Archives at Petabase-scale. Technical report, bioRxiv, November 2020. Type: article.

[96] Mikhail Karasikov, Harun Mustafa, Amir Joudaki, Sara Javadzadeh-no, Gunnar Rätsch, and André Kahles. Sparse Binary Relation Representations for Genome Graph Annotation. *Journal of Computational Biology*, 27(4):626–639, December 2019.

[97] Mikhail Karasikov, Harun Mustafa, Gunnar Rätsch, and André Kahles. Lossless indexing with counting de bruijn graphs. *bioRxiv*, 2022.

[98] Mikhail Karasikov, Harun Mustafa, Gunnar Rätsch, and Andre Kahles. Lossless indexing with counting de Bruijn graphs. *Genome Research*, page gr.276607.122, May 2022.

[99] Lee S. Katz, Taylor Griswold, Shatavia S. Morrison, Jason A. Caravas, Shaokang Zhang, Henk C. den Bakker, Xiangyu Deng, and Heather A. Carleton. Mashtree: a rapid comparison of whole genome sequence files. *Journal of Open Source Software*, 4(44):1762, December 2019.

[100] Parsoa Khorsand and Fereydoun Hormozdiari. Nebula: ultra-efficient mapping-free structural variant genotyper. *Nucleic Acids Research*, 49(8):e47, January 2021.

[101] Marek Kokot, Maciej D13ugosz, and Sebastian Deorowicz. KMC 3: counting and manipulating k-mer statistics. *Bioinformatics*, 33(17):2759–2761, 05 2017.

[102] Sergey Koren, Brian P. Walenz, Konstantin Berlin, Jason R. Miller, Nicholas H. Bergman, and Adam M. Phillippy. Canu: scalable and accurate long-read assembly via adaptive k -mer weighting and repeat separation. *Genome Research*, 27(5):722–736, May 2017.

[103] David Koslicki and Hooman Zabeti. Improving MinHash via the containment index with applications to metagenomic analysis. *Applied Mathematics and Computation*, 354:206–215, August 2019.

[104] Sam Kovaka, Yunfan Fan, Bohan Ni, Winston Timp, and Michael C. Schatz. Targeted nanopore sequencing by real-time mapping of raw electrical signal with UNCALLED. *Nature Biotechnology*, 39(4):431–441, April 2021.

[105] Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature methods*, 9(4):357–359, March 2012.

[106] Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L. Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology*, 10(3):R25, March 2009.

[107] Nathan LaPierre, Mohammed Alser, Eleazar Eskin, David Koslicki, and Serghei Mangul. Metalign: efficient alignment-based metagenomic profiling via containment min hash. *Genome Biology*, 21(1):242, September 2020.

[108] Chris-André Leimeister, Salma Sohrabi-Jahromi, and Burkhard Morgenstern. Fast and accurate phylogeny reconstruction using filtered spaced-word matches. *Bioinformatics*, page btw776, January 2017.

[109] Téo Lemane, Paul Medvedev, Rayan Chikhi, and Pierre Peterlongo. kmtricks: Efficient construction of Bloom filters for large sequencing data collections. *bioRxiv*, page 2021.02.16.429304, February 2021.

[110] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Finding Similar Items*, page 68–122. Cambridge University Press, 2 edition, 2014.

[111] M. J. Levene, J. Korlach, S. W. Turner, M. Foquet, H. G. Craighead, and W. W. Webb. Zero-Mode Waveguides for Single-Molecule Analysis at High Concentrations. *Science*, 299(5607):682–686, January 2003.

[112] H. Li and R. Durbin. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, 25(14):1754–1760, July 2009.

[113] Heng Li. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, 32(14):2103–2110, July 2016.

[114] Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, September 2018.

[115] M. Li, X. Chen, X. Li, B. Ma, and P.M.B. Vitanyi. The Similarity Metric. *IEEE Transactions on Information Theory*, 50(12):3250–3264, December 2004.

[116] Ping Li. 0-Bit Consistent Weighted Sampling. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 665–674, Sydney NSW Australia, August 2015. ACM.

[117] R. Li, C. Yu, Y. Li, T.-W. Lam, S.-M. Yiu, K. Kristiansen, and J. Wang. SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics*, 25(15):1966–1967, August 2009.

[118] Ruiqiang Li, Hongmei Zhu, Jue Ruan, Wubin Qian, Xiaodong Fang, Zhongbin Shi, Yingrui Li, Shengting Li, Gao Shan, Karsten Kristiansen, Songgang Li, Huanming Yang, Jian Wang, and Jun Wang. De novo assembly of human genomes with massively parallel short read sequencing. *Genome Research*, 20(2):265–272, February 2010.

[119] Yang Li and XifengYan. MSPKmerCounter: A Fast and Memory Efficient Approach for K-mer Counting. Technical Report arXiv:1505.06550, arXiv, May 2015. arXiv:1505.06550 [cs, q-bio] type: article.

[120] Yuanning Li, Kyle T. David, Xing-Xing Shen, Jacob L. Steenwyk, Kenneth M. Halanych, and Antonis Rokas. Feature frequency profile-based phylogenies are inaccurate. *Proceedings of the National Academy of Sciences*, 117(50):31580–31581, December 2020.

[121] Antoine Limasset, Jean-François Flot, and Pierre Peterlongo. Toward perfect reads: self-correction of short reads via mapping on de Bruijn graphs. *Bioinformatics*, 36(5):1374–1381, March 2020.

[122] Antoine Limasset, Guillaume Rizk, Rayan Chikhi, and Pierre Peterlongo. Fast and scalable minimal perfect hashing for massive key sets. *arXiv:1702.03154 [cs]*, February 2017. arXiv: 1702.03154.

[123] D J Lipman, S F Altschul, and J D Kececioglu. A tool for multiple sequence alignment. *Proceedings of the National Academy of Sciences*, 86(12):4412–4415, June 1989.

[124] Lin Liu, Yinhu Li, Siliang Li, Ni Hu, Yimin He, Ray Pong, Danni Lin, Lihua Lu, and Maggie Law. Comparison of Next-Generation Sequencing Systems. *Journal of Biomedicine and Biotechnology*, 2012:e251364, July 2012.

[125] Yongchao Liu, Jan Schröder, and Bertil Schmidt. Musket: a multistage k-mer spectrum-based error corrector for Illumina sequence data. *Bioinformatics*, 29(3):308–315, 11 2012.

[126] Yi Lu, Andrea Montanari, Balaji Prabhakar, Sarang Dharmapurikar, and Abdul Kabbani. Counter braids: a novel counter architecture for per-flow measurement. *ACM SIGMETRICS Performance Evaluation Review*, 36(1):121–132, June 2008.

[127] Brian B Luczak, Benjamin T James, and Hani Z Girgis. A survey and evaluations of histogram-based statistics in alignment-free sequence comparison. *Briefings in Bioinformatics*, 20(4):1222–1237, July 2019.

[128] B. S. Majewski, N. C. Wormald, G. Havas, and Z. J. Czech. A Family of Perfect Hashing Methods. *The Computer Journal*, 39(6):547–554, 01 1996.

[129] Mark Manasse, Frank McSherry, and Kunal Talwar. Consistent Weighted Sampling. *ICDM*, June 2010.

[130] Udi Manber and Gene Myers. Suffix Arrays: A New Method for On-Line String Searches. *SIAM Journal on Computing*, 22(5):935–948, October 1993.

[131] 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.

[132] Camille Marchet, Zamin Iqbal, Daniel Gautheret, Mikaël Salson, and Rayan Chikhi. REINDEER: efficient indexing of k-mer presence and abundance in sequencing datasets. *Bioinformatics*, 36(Supplement_1):i177–i185, July 2020.

[133] Camille Marchet, Mael Kerbiriou, and Antoine Limasset. BLight: Efficient exact associative structure for k-mers. *Bioinformatics (Oxford, England)*, page btab217, April 2021.

[134] Guillaume Marçais, Dan DeBlasio, Prashant Pandey, and Carl Kingsford. Locality-sensitive hashing for the edit distance. *Bioinformatics*, 35(14):i127–i135, July 2019.

[135] Guillaume Marçais and Carl Kingsford. A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics*, 27(6):764, March 2011.

[136] A M Maxam and W Gilbert. A new method for sequencing DNA. *Proceedings of the National Academy of Sciences*, 74(2):560–564, February 1977.

[137] Andrew McGregor. Graph stream algorithms: a survey. *ACM SIGMOD Record*, 43(1):9–20, May 2014.

[138] Kurt Mehlhorn. On the program size of perfect and universal hash functions. *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 170–175, 1982.

[139] Michael Molloy. The pure literal rule threshold and cores in random hypergraphs. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '04, pages 672–681, USA, January 2004. Society for Industrial and Applied Mathematics.

[140] Martin D Muggli, Bahar Alipanahi, and Christina Boucher. Building large updatable colored de Bruijn graphs via merging. *Bioinformatics*, 35(14):i51–i60, July 2019.

[141] Kary B. Mullis, François Ferré, and Richard A. Gibbs. *The Polymerase Chain Reaction*. Springer Science & Business, 1994.

[142] Allan H. Murphy. The Finley Affair: A Signal Event in the History of Forecast Verification. *Weather and Forecasting*, 11(1):3–20, March 1996.

[143] Harun Mustafa, André Kahles, Mikhail Karasikov, and Gunnar Rätsch. Metannot: A succinct data structure for compression of colors in dynamic de Bruijn graphs. Technical report, bioRxiv, March 2018. Type: article.

[144] Ingo Müller, Peter Sanders, Robert Schulze, and Wei Zhou. Retrieval and Perfect Hashing Using Fingerprinting. In Joachim Gudmundsson and Jyrki Katajainen, editors, *Experimental Algorithms*, Lecture Notes in Computer Science, pages 138–149, Cham, 2014. Springer International Publishing.

[145] Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, March 1970.

[146] Cédric Notredame, Desmond G Higgins, and Jaap Heringa. T-coffee: a novel method for fast and accurate multiple sequence alignment11Edited by J. Thornton. *Journal of Molecular Biology*, 302(1):205–217, September 2000.

[147] Brian D. Ondov, Gabriel J. Starrett, Anna Sappington, Aleksandra Kostic, Sergey Koren, Christopher B. Buck, and Adam M. Phillippy. Mash Screen: high-throughput sequence containment estimation for genome discovery. *Genome Biology*, 20(1):232, November 2019.

[148] 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.

[149] 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*, pages 775–787, Chicago Illinois USA, May 2017. ACM.

[150] 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.

[151] W R Pearson and D J Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences*, 85(8):2444–2448, April 1988.

[152] Heather E. Peckham, Stephen F. McLaughlin, Jingwei N. Ni, Michael D. Rhodes, Joel A. Melek, Kevin J. McKernan, and Alan P. Blanchard. Solid^{TM} sequencing and 2-base encoding. 2007.

[153] Yu Peng, Henry C. M. Leung, S. M. Yiu, and Francis Y. L. Chin. IDBA – A Practical Iterative de Bruijn Graph De Novo Assembler. In Bonnie Berger, editor, *Research in Computational Molecular Biology*, Lecture Notes in Computer Science, pages 426–440, Berlin, Heidelberg, 2010. Springer.

[154] Giulio Ermanno Pibiri. On Weighted K-Mer Dictionaries. Technical report, bioRxiv, May 2022. Type: article.

[155] Giulio Ermanno Pibiri. Sparse and Skew Hashing of K-Mers. Technical report, bioRxiv, April 2022. Type: article.

[156] Giulio Ermanno Pibiri and Roberto Trani. Parallel and External-Memory Construction of Minimal Perfect Hash Functions with PTHash. Technical Report arXiv:2106.02350, arXiv, June 2021. arXiv:2106.02350 [cs] type: article.

[157] Giulio Ermanno Pibiri and Roberto Trani. PTHash: Revisiting FCH Minimal Perfect Hashing. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21, pages 1339–1348, New York, NY, USA, July 2021. Association for Computing Machinery.

[158] Armando J Pinho, Paulo JSG Ferreira, Sara P Garcia, and João MOS Rodrigues. On finding minimal absent words. *BMC Bioinformatics*, 10(1):137, December 2009.

[159] Ely Porat and Ohad Lipsky. Improved sketching of hamming distance with error correcting. In Bin Ma and Kaizhong Zhang, editors, *Combinatorial Pattern Matching*, pages 173–182, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[160] David T. Pride, Richard J. Meinersmann, Trudy M. Wassenaar, and Martin J. Blaser. Evolutionary Implications of Microbial Genome Tetranucleotide Frequency Biases. *Genome Research*, 13(2):145–158, February 2003.

[161] Amatur Rahman, Rayan Chikhi, and Paul Medvedev. Disk compression of k-mer sets. *Algorithms for Molecular Biology*, 16(1):10, June 2021.

[162] Amatur Rahman and Paul Medvedev. Representation of k-mer sets using spectrum-preserving string sets. *bioRxiv*, page 2020.01.07.896928, January 2020.

[163] Atif Rahman, Ingileif Hallgrímsdóttir, Michael Eisen, and Lior Pachter. Association mapping from sequencing reads using k-mers. *eLife*, 7:e32920, June 2018.

[164] Milan Randić, Jure Zupan, and Alexandru T. Balaban. Unique graphical representation of protein sequences based on nucleotide triplet codons. *Chemical Physics Letters*, 397(1):247–252, October 2004.

[165] Guillaume Rizk, Dominique Lavenier, and Rayan Chikhi. DSK: k-mer counting with very low memory usage. *Bioinformatics*, 29(5):652–653, March 2013.

[166] Raffaella Rizzi, Stefano Beretta, Murray Patterson, Yuri Pirola, Marco Previtali, Gianluca Della Vedova, and Paola Bonizzoni. Overlap graphs and de Bruijn graphs: data structures for de novo genome assembly in the big data era. *Quantitative Biology*, 7(4):278–292, December 2019.

[167] Michael Roberts, Wayne Hayes, Brian R. Hunt, Stephen M. Mount, and James A. Yorke. Reducing storage requirements for biological sequence comparison. *Bioinformatics*, 20(18):3363–3369, December 2004.

[168] Mostafa Ronaghi, Samer Karamohamed, Bertil Pettersson, Mathias Uhlén, and Pål Nyrén. Real-Time DNA Sequencing Using Detection of Pyrophosphate Release. *Analytical Biochemistry*, 242(1):84–89, November 1996.

[169] Will PM Rowe, Anna Paola Carrieri, Cristina Alcon-Giner, Shabhonam Caim, Alex Shaw, Kathleen Sim, J. Simon Kroll, Lindsay J. Hall, Edward O. Pyzer-Knapp, and Martyn D. Winn. Streaming histogram sketching for rapid microbiome analytics. *Microbiome*, 7(1):40, March 2019.

[170] Jue Ruan and Heng Li. Fast and accurate long-read assembly with wtdbg2. *Nature Methods*, 17(2):155–158, February 2020.

[171] Douglas B Rusch, Aaron L Halpern, Granger Sutton, Karla B Heidelberg, Shannon Williamson, Shibu Yooseph, Dongying Wu, Jonathan A Eisen, Jeff M Hoffman, Karin Remington, Karen Beeson, Bao Tran, Hamilton Smith, Holly Baden-Tillson, Clare Stewart, Joyce Thorpe, Jason Freeman, Cynthia Andrews-Pfannkoch, Joseph E Venter, Kelvin Li, Saul Kravitz, John F Heidelberg, Terry Utterback, Yu-Hui Rogers, Luisa I Falcón, Valeria Souza, Germán Bonilla-Rosso, Luis E Eguiarte, David M Karl, Shubha Sathyendranath, Trevor Platt, Eldredge Bermingham, Victor Gallardo, Giselle Tamayo-Castillo, Michael R Ferrari, Robert L Strausberg, Kenneth Nealson, Robert Friedman, Marvin Frazier, and J. Craig Venter. The Sorcerer II Global Ocean Sampling Expedition: Northwest Atlantic through Eastern Tropical Pacific. *PLoS Biology*, 5(3):e77, March 2007.

[172] Randall K. Saiki, David H. Gelfand, Susanne Stoffel, Stephen J. Scharf, Russell Higuchi, Glenn T. Horn, Kary B. Mullis, and Henry A. Erlich. Primer-Directed Enzymatic Amplification of DNA with a Thermostable DNA Polymerase. *Science*, 239(4839):487–491, January 1988.

[173] K. Salikhov, G. Sacomoto, and G. Kucherov. Using cascading Bloom filters to improve the memory usage for de Brujin graphs. *BMC Algorithms for Molecular Biology*, 9(1):2, 2014.

[174] F. Sanger and A. R. Coulson. A rapid method for determining sequences in DNA by primed synthesis with DNA polymerase. *Journal of Molecular Biology*, 94(3):441–448, May 1975.

[175] Shahab Sarmashghi, Kristine Bohmann, M. Thomas P. Gilbert, Vineet Bafna, and Siavash Mirarab. Skmer: assembly-free and alignment-free sample identification using genome skims. *Genome Biology*, 20(1):34, February 2019.

[176] Melanie Schirmer, Rosalinda D'Amore, Umer Z. Ijaz, Neil Hall, and Christopher Quince. Illumina error profiles: resolving fine-scale variation in metagenomic sequencing data. *BMC Bioinformatics*, 17(1):125, March 2016.

[177] Melanie Schirmer, Umer Z. Ijaz, Rosalinda D'Amore, Neil Hall, William T. Sloan, and Christopher Quince. Insight into biases and sequencing errors for amplicon sequencing with the Illumina MiSeq platform. *Nucleic Acids Research*, 43(6):e37–e37, 01 2015.

[178] Saul Schleimer, Daniel S. Wilkerson, and Alex Aiken. Winnowing: local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 76–85, San Diego, California, June 2003. Association for Computing Machinery.

[179] Sebastian Schmidt, Shahbaz Khan, Jarno Alanko, and Alexandru I. Tomescu. Matchtigs: minimum plain text representation of kmer sets. Technical report, bioRxiv, February 2022. Type: article.

[180] Ariya Shajii, Deniz Yorukoglu, Yun William Yu, and Bonnie Berger. Fast genotyping of known SNPs through approximate k -mer matching. *Bioinformatics*, 32(17):i538–i544, September 2016.

[181] Christina Huan Shi and Kevin Y. Yip. K-mer counting with low memory consumption enables fast clustering of single-cell sequencing data without read alignment. Technical report, bioRxiv, August 2019. Type: article.

[182] Yoshihiro Shibuya, Djamal Belazzougui, and Gregory Kucherov. Space-Efficient Representation of Genomic k-Mer Count Tables. In Alessandra Carbone and Mohammed El-Kebir, editors, *21st International Workshop on Algorithms in Bioinformatics (WABI 2021)*, volume 201 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:19, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[183] Yoshihiro Shibuya, Djamal Belazzougui, and Gregory Kucherov. Efficient reconciliation of genomic datasets of high similarity. *bioRxiv*, page 2022.06.07.495186, June 2022. Type: article.

[184] Yoshihiro Shibuya, Djamal Belazzougui, and Gregory Kucherov. Set-Min Sketch: A Probabilistic Map for Power-Law Distributions with Application to k-Mer Annotation. *Journal of Computational Biology*, 29(2):140–154, February 2022.

[185] Yoshihiro Shibuya, Djamal Belazzougui, and Gregory Kucherov. Space-efficient representation of genomic k-mer count tables. *Algorithms for Molecular Biology*, 17(1):5, March 2022.

[186] Yoshihiro Shibuya and Matteo Comin. Better quality score compression through sequence-based quality smoothing. *BMC Bioinformatics*, 20(9):302, November 2019.

[187] Yoshihiro Shibuya and Matteo Comin. Indexing k-mers in linear space for quality value compression. *Journal of Bioinformatics and Computational Biology*, 17(05):1940011, October 2019.

[188] Yoshihiro Shibuya and Gregory Kucherov. Set-min Sketch: a Probabilistic Map for Power-Law Distributions with Application to k-mer Annotation. *bioRxiv*, page 2020.11.14.382713, November 2020.

[189] Moustafa Shokrof, C. Titus Brown, and Tamer A. Mansour. MQF and buffered MQF: quotient filters for efficient storage of k-mers with their counts and metadata. *BMC Bioinformatics*, 22(1):71, February 2021.

[190] Anshumali Shrivastava. Exact Weighted Minwise Hashing in Constant Time. *ArXiv*, 2016.

[191] Anshumali Shrivastava. Simple and Efficient Weighted Minwise Hashing. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[192] Jared T. Simpson, Kim Wong, Shaun D. Jackman, Jacqueline E. Schein, Steven J.M. Jones, and İnanç Birol. ABySS: A parallel assembler for short read sequence data. *Genome Research*, 19(6):1117–1123, June 2009.

[193] Gregory E. Sims, Se-Ran Jun, Guohong A. Wu, and Sung-Hou Kim. Alignment-free genome comparison with feature frequency profiles (FFP) and optimal resolutions. *Proceedings of the National Academy of Sciences*, 106(8):2677–2682, February 2009.

[194] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, March 1981.

[195] Brad Solomon and Carl Kingsford. Fast Search of Thousands of Short-Read Sequencing Experiments. *Nature biotechnology*, 34(3):300, March 2016.

[196] Chen Sun and Paul Medvedev. Toward fast and accurate SNP genotyping from whole genome sequencing data for bedside diagnostics. *Bioinformatics*, 35(3):415–420, February 2019.

[197] Fatih Taşyaran, Kerem Yıldırır, Kamer Kaya, and Mustafa Kemal Taş. One Table to Count Them All: Parallel Frequency Estimation on Single-Board Computers. Technical Report arXiv:1903.00729, arXiv, March 2019. arXiv:1903.00729 [cs] type: article.

[198] Luke R. Thompson, Jon G. Sanders, Daniel McDonald, Amnon Amir, Joshua Ladau, Kenneth J. Locey, Robert J. Prill, Anupriya Tripathi, Sean M. Gibbons, Gail Ackermann, Jose A. Navas-Molina, Stefan Janssen, Evguenia Kopylova, Yoshiki Vázquez-Baeza, Antonio González, James T. Morton, Siavash Mirarab, Zhenjiang Zech Xu, Lingjing Jiang, Mohamed F. Haroon, Jad Kanbar, Qiyun Zhu, Se Jin Song, Tomasz Kosciolek, Nicholas A. Bokulich, Joshua Lefler, Colin J. Brislawn, Gregory Humphrey, Sarah M. Owens, Jarrad Hampton-Marcell, Donna Berg-Lyons, Valerie McKenzie, Noah Fierer, Jed A. Fuhrman, Aaron Clauset, Rick L. Stevens, Ashley Shade, Katherine S. Pollard, Kelly D. Goodwin, Janet K. Jansson, Jack A. Gilbert, and Rob Knight. A communal catalogue reveals Earth's multiscale microbial diversity. *Nature*, 551(7681):457–463, November 2017.

[199] Daniel Ting. Count-Min: Optimal Estimation and Tight Error Bounds using Empirical Error Distributions. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2319–2328, London United Kingdom, July 2018. ACM.

[200] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, September 1995.

[201] Igor Ulitsky, David Burstein, Tamir Tuller, and Benny Chor. The Average Common Substring Approach to Phylogenomic Reconstruction. *Journal of Computational Biology*, 13(2):336–350, March 2006.

[202] Robert Vaser, Ivan Sović, Niranjan Nagarajan, and Mile Šikić. Fast and accurate de novo genome assembly from long uncorrected reads. *Genome Research*, 27(5):737–746, May 2017.

[203] S. Vinga. Information theory applications for biological sequence analysis. *Briefings in Bioinformatics*, 15(3):376–389, May 2014.

[204] Peter Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory (swat 1973)*, pages 1–11, October 1973. ISSN: 0272-4847.

[205] Jia Wen and YuYan Zhang. A 2D graphical representation of protein sequence and its numerical characterization. *Chemical Physics Letters*, 476(4):281–286, July 2009.

[206] W J Wilbur and D J Lipman. Rapid similarity searches of nucleic acid and protein data banks. *Proceedings of the National Academy of Sciences*, 80(3):726–730, February 1983.

[207] Derrick E Wood and Steven L Salzberg. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome Biology*, 15(3):R46, 2014.

[208] Wei Wu, B. Li, Ling Chen, and Chengqi Zhang. Consistent Weighted Sampling Made More Practical. *WWW*, 2017.

[209] Yiqing Yan, Nimisha Chaturvedi, and Raja Appuswamy. Accel-Align: a fast sequence mapper and aligner based on the seed–embed–extend method. *BMC Bioinformatics*, 22(1):257, May 2021.

[210] Dingqi Yang, Bin Li, Laura Rettig, and Philippe Cudré-Mauroux. HistoSketch: Fast Similarity-Preserving Sketching of Streaming Histograms with Concept Drift. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 545–554, November 2017. ISSN: 2374-8486.

[211] Dingqi Yang, Bin Li, Laura Rettig, and Philippe Cudré-Mauroux. D^2HistoSketch: Discriminative and Dynamic Similarity-Preserving Sketching of Streaming Histograms. *IEEE Transactions on Knowledge and Data Engineering*, 31(10):1898–1911, October 2019.

[212] Lianping Yang, Xiangde Zhang, Tianming Wang, and Hegui Zhu. Large Local Analysis of the Unaligned Genome and Its Application. *Journal of Computational Biology*, 20(1):19–29, January 2013.

[213] Huiguang Yi and Li Jin. Co-phylog: an assembly-free phylogenomic approach for closely related organisms. *Nucleic Acids Research*, 41(7):e75, April 2013.

[214] Ye Yu, Djamal Belazzougui, Chen Qian, and Qin Zhang. Memory-efficient and Ultra-fast Network Lookup and Forwarding using Othello Hashing. *arXiv:1608.05699 [cs]*, November 2017. arXiv: 1608.05699.

[215] Ye Yu, Jinpeng Liu, Xinan Liu, Yi Zhang, Eamonn Magner, Erik Lehnert, Chen Qian, and Jinze Liu. SeqOthello: querying RNA-seq experiments at scale. *Genome Biology*, 19(1):167, October 2018.

[216] Yun William Yu and Griffin M. Weber. HyperMinHash: MinHash in LogLog space. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2020.

[217] Daniel R. Zerbino and Ewan Birney. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research*, 18(5):821–829, May 2008.

[218] Haoyu Zhang and Qin Zhang. EmbedJoin: Efficient Edit Similarity Joins via Embeddings. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 585–594, Halifax NS Canada, August 2017. ACM.

[219] Haoyu Zhang and Qin Zhang. MinJoin: Efficient Edit Similarity Joins via Local Hash Minima. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1093–1103, Anchorage AK USA, July 2019. ACM.

[220] Qingpeng Zhang, Jason Pell, Rosangela Canino-Koning, Adina Chuang Howe, and C. Titus Brown. These are not the k-mers you are looking for: efficient online k-mer counting using a probabilistic data structure. *PloS One*, 9(7):e101271, 2014.

[221] XiaoFei Zhao. BinDash, software for fast genome distance estimation on a typical personal laptop. *Bioinformatics*, 35(4):671–673, February 2019.

[222] Hongyu Zheng, Carl Kingsford, and Guillaume Marçais. Lower Density Selection Schemes via Small Universal Hitting Sets with Short Remaining Path Length. In Russell Schwartz, editor, *Research in Computational Molecular Biology*, Lecture Notes in Computer Science, pages 202–217, Cham, 2020. Springer International Publishing.

[223] Andrzej Zielezinski, Hani Z. Girgis, Guillaume Bernard, Chris-Andre Leimeister, Kujin Tang, Thomas Dencker, Anna Katharina Lau, Sophie Röhling, Jae Jin Choi, Michael S. Waterman, Matteo Comin, Sung-Hou Kim, Susana Vinga, Jonas S. Almeida, Cheong Xin Chan, Benjamin T. James, Fengzhu Sun, Burkhard Morgenstern, and Wojciech M. Karlowski. Benchmarking of alignment-free sequence comparison methods. *Genome Biology*, 20(1):144, July 2019.

[224] Andrzej Zielezinski, Susana Vinga, Jonas Almeida, and Wojciech M. Karlowski. Alignment-free sequence comparison: benefits, applications, and tools. *Genome Biology*, 18(1):186, October 2017.

Abstract

Sequence comparison is one of the fundamental concepts of bioinformatics. Alignment-free algorithms representing sequences as (multi-)sets of their constituent k-mers are widely used in practice due to their simplicity and speed. However, exact data structures storing k-mer multiplicities are usually suboptimal as they do not take advantage of the characteristic Power-Law distribution of k-mer counts. Sketching techniques, on the other hand, suffer from large count estimation errors.

This thesis addresses the problem of efficiently storing counts for known sets of k-mers and the problem of computing differences (and by extension, Jaccard similarity) of very similar sets in small space. Two independent solutions are provided for the first problem. The first one consists in a sketching-based solution inspired by Count-Min sketch, with provable better error guarantees for k-mers following a heavily skewed distribution. The second method starts by improving on a previous implementation of Compressed Static Functions, by extending its applicability to data of entropy < 1. The resulting algorithm and data structure are then paired with minimizer-based bucketing in order to produce two additional compression strategies.

For the second topic of this work, we propose Invertible Bloom Lookup Tables (IBLTs) as an efficient way to compute symmetric differences of similar k-mer sets. The combination of IBLTs and syncmers allows, in some instances, to estimate Jaccard similarity more precisely than comparable MinHash sketches. Further, the full versatility of IBLTs is demonstrated by computing approximated set differences of full k-mer sets.

Résumé

La comparaison de séquences est l'un des concepts fondamentaux de la bioinformatique. En pratique, dans plusieurs applications, les algorithmes "sans alignement" sont souvent employés en raison de leur majeure vitesse et simplicité par rapport aux solutions classiques. Dans ce cadre, les séquences sont représentées comme des ensembles de k-mers. Cependant, les structures de données pour stocker de k-mers avec leurs multiplicités n'utilisent pas le fait que celles-ci suivent des distributions "Power-Law" très asymmétriques. D'autre part, les techniques classiques d'esquissage (sketching) pour l'estimation de poids présentent des erreurs trop grandes pour la plupart d'applications bioinformatiques.

Dans cette thèse nous introduisons deux méthodes alternatives visées à représenter les poids associés aux k-mers de manière efficace. La première est une nouvelle esquisse inspirée à Count-Min sketch, adaptée à de distributions de k-mers particulièrement asymétriques. Notre deuxième technique est plutôt une famille de méthodes avec pour base une version améliorée d'une précédente implantation de Fonctions Statiques Compressées ("Compressed Static Function" ou CSFs). Les CSFs sont après combinées avec un regroupement basé sur les minimizers afin de produire deux stratégies de compression supplémentaires.

Finalement, le dernier argument de cette thèse porte sur l'introduction d' "Invertible Bloom Lookup Tables" (IBLTs) afin de calculer efficacement de différences symétriques entre ensembles de k-mers. La combinaison d'IBLTs avec de l'échantillonnage basé sur les syncmers permet une plus efficace estimation de la similarité de Jaccard par rapport aux esquisses MinHash quand les ensembles en question sont très similaires. La pleine versatilité d'IBLTs est démontrée en approximant la différence entre ensembles de k-mers complets, sans aucun type d'échantillonnage.