Utilizing alignment-free methods to enable quantitative gene expression analysis of large collections of sequencing data

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

am

Fachbereich Mathematik und Informatik

Freie Universität Berlin

vorgelegt von

Mitra Darja Darvish

Berlin

August 2023

Gutachter:

Prof. Dr. Knut Reinert, *Freie Universität Berlin*

Prof. Dr. Zamin Iqbal, *University of Bath*

Tag der Disputation: 8. Dezember 2023

Abstract

Due to advances in sequencing technologies, the amount of sequencing data is continuously increasing and has reached an amount that calls for new data management methods, to actually utilize the sequencing data. In the last years, a number of different indices haven been developed to simplify the data, thereby reducing the amount of space needed and enabling analysis on large collections of sequencing data.

In this thesis, the index *Needle* will be introduced, which allows (semi-)quantitative analyses on large data sets and outperforms other existing solutions with regards to both space and speed.

Needle, like other indices, is based on alignment-free methods because in this way the costly step of classical sequence analyses, the alignment, can be omitted. Alignment-free methods are based on short subsequences of the actual sequence data. There are multiple different methods to determine these subsequences and this thesis provides a detailed analysis and comparison to determine the best method for such indices. Moreover, the benchmarking application *minions* is introduced, which will make comparisons between these methods easier as adding future new methods is simple.

Needle is capable of utilizing large collections of sequencing data and determining their gene expressions. Three analyses are performed, which act as a proof of concept for how *Needle* can be utilized for large collections of sequencing data. Therefore, *Needle* is applied in this thesis to find cancer signatures, a newly annotated mouse transcript and tissue specific differentially expressed genes for different large data sets.

In summary, indices like *Needle* are needed to actually take advantage of the data wealth currently present in the biological and medical research field.

Zusammenfassung

Dank moderner Sequenzierungstechniken steigt die Menge an Sequenzdaten kontinuierlich und hat Ausmaße angenommen, die neue Methoden zur Datenverwaltung erfordern, wenn die Daten nicht nur gespeichert, sondern wiederverwendet und ausgewertet werden sollen. In den letzten Jahren sind daher verschiedene neue Indexe entwickelt worden, die die Sequenzdaten vereinfachen und dadurch weniger Speicherplatz verbrauchen sowie Analysen auf den Daten ermöglichen.

In dieser Arbeit wird der Index *Needle* vorgestellt, der (semi-)quantitative Analysen erlaubt und im Vergleich zu anderen bestehenden Indexen am besten performt hinsichtlich Speicherverbrauch und Laufzeit.

Needle sowie die anderen Indexe beruhen auf Alignement-freien Methoden, da so der zeit- und speicherintensive Schritt in der klassischen Sequenzanalyse, das Alignieren, umgangen werden kann. Dies wird erreicht, indem nur kurzen Teilsequenzen, die in den gegebenen Sequenzdaten vorkommen, betrachtet werden. Es gibt zahlreiche Methoden diese kurzen Teilsequenzen zu bestimmen und hier wird eine detaillierte Analyse vorgenommen, um zu bestimmen, welche Methode am besten geeignet ist. Gleichzeitig wird die Applikation *minions* vorgestellt, die Vergleiche zwischen den verschiedenen Methoden erleichtern soll, indem zukünftige neue Methoden leicht hinzugefügt werden können.

Needle ermöglicht es nicht nur große Mengen an Sequenzdaten in einem kleinen Index abzuspeichern, sondern auch vielfältige Analysen durchzuführen. In dieser Arbeit wird *Needle* genutzt, um Krebssignaturen, ein neu-annotiertes Transkript und unterschiedlich exprimierte Gene zu finden in verschiedenen großen Datensätzen.

Zusammenfassend lässt sich festhalten, dass es Indexe wie *Needle* braucht, um den Datenreichtum in der biologisch-medizinischen Wissenschaft tatsächlich nutzen zu können.

Selbstständigkeitserklärung

Name: Darvish
Vorname: Mitra Darja

Ich erkläre gegenüber der Freien Universität Berlin, dass ich die vorliegende Dissertation selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe. Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht. Diese Dissertation wurde in gleicher oder ähnlicher Form noch in keinem früheren Promotionsverfahren eingereicht.

Mit einer Prüfung meiner Arbeit durch ein Plagiatsprüfungsprogramm erkläre ich mich einverstanden.

Berlin, 29.08.2023

Unterschrift: _____

Acknowledgements

I want to thank Prof. Knut Reinert for giving me the opportunity to work on this topic in the first place as well as for his guidance and mentorship throughout my doctoral journey.

I am also grateful for all the help and supervision I received through the International Max Planck Research School for Biology and Computation, especially from my thesis advisory committee consisting of Prof. Martin Vingron and Prof. Bernhard Renard.

I am indebted to my colleagues, who were not only always open for discussions and never tired to answer my questions, but also made the whole experience enjoyable.

Outside of the academic context, I would like to express my sincere appreciation to my family and friends, who kept me sane all these years. Without my parents and my brother's unwavering support and their never ending belief in me I would never have managed to come so far.

A special thanks goes to my partner in crime Joshua Kim, who never is short of kind words when I need them the most and listens to me even late at night when he would rather fall asleep.

Lastly, I want to use the opportunity to thank Prof. Beate Rau and her team for saving my life and thereby making this experience possible in the first place.

Contents

1 Introduction **1**

2 Alignment-free quantification **5**

2.1 Submers . . . 6

2.1.1 k-mers . . . 6

2.1.2 Strobemers . . . 8

2.1.3 Comparison of k-mers and strobemers . . . 14

2.2 Representative submers . . . 25

2.2.1 Modmers . . . 26

2.2.2 Minimizers . . . 26

2.2.3 Syncmers . . . 28

2.2.4 Universal hitting sets . . . 31

2.2.5 Minimum decycling sets . . . 32

2.2.6 Polar sets . . . 32

2.2.7 Prefix words . . . 33

2.2.8 Sigmer . . . 33

2.2.9 Fleximer . . . 33

2.2.10 Comparison of different methods . . . 33

2.2.11 Conclusion of the comparison . . . 45

2.3 Transcript quantification . . . 46

2.3.1 Overview of existing transcript quantification application . . . 46

2.3.2 Needle count . . . 47

2.3.3 Comparison of transcript quantification applications . . . 48

3 Indexing big data **55**

3.1 Bloom filter . . . 56

3.2 Colored de Bruijn graph . . . 57

3.3 Error handling . . . 58

3.4 Non-quantitative indices . . . 59

3.4.1 SeqOthello . . . 59

3.4.2 Mantis . . . 60

3.4.3 Bifrost . . . 62
3.4.4 MetaGraph . . . 63
3.4.5 Sequence Bloom Tree . . . 64
3.4.6 COBS and MetaProFi . . . 67
3.4.7 RAMBO . . . 69
3.4.8 PAC . . . 70
3.4.9 Raptor . . . 71
3.5 Quantitative indices . . . 71
3.5.1 Reindeer . . . 72
3.5.2 Gazelle . . . 73
3.5.3 Counting DBG . . . 74
3.6 Needle . . . 75
3.6.1 Workflow Needle . . . 75
3.6.2 IBF . . . 78
3.6.3 HIBF . . . 79
3.6.4 Counting IBF . . . 82
3.6.5 Implementation . . . 85
3.6.6 Normalization . . . 86
3.6.7 Dynamic update . . . 87
3.6.8 Parallelization . . . 87
3.7 Comparison of quantitative indices . . . 88
3.7.1 Accuracy . . . 89
3.7.2 Space and speed . . . 93
3.7.3 Conclusion . . . 99

4 Utilizing large collections of sequencing data 101
4.1 Determining genes of interest . . . 101
4.2 Finding a newly annotated mouse transcript . . . 103
4.2.1 Data and building the Needle index . . . 104
4.2.2 Finding Xert in the original data . . . 104
4.2.3 Finding Xert in mouse randomly picked RNA-Seqs . . . 105
4.2.4 Conclusion . . . 107
4.3 Cancer signatures . . . 108
4.3.1 Data and building the Needle index . . . 109
4.3.2 Finding TNBC signature . . . 109
4.3.3 Finding NAT signature . . . 112
4.3.4 Conclusion . . . 112

5 Conclusion 115

References **117**

A Appendix **133**

A.1 Repositories 133

A.2 Alignment-free quantification 133

A.2.1 Submers 133

A.2.2 Representative submers 133

A.3 Indexing big data 142

A.3.1 Overview of indices 142

A.3.2 Space and Speed Analyses 142

Chapter 1

Introduction

In 2001, the Human Genome Project revealed the first draft sequence of the human genome [1] and thereby marked a turning point in the fields of biology and medicine by introducing big data to these fields [2]. While the first draft sequence of the human genome took 15 months to complete and had a cost of around $ 300 million [3], nowadays sequencing can be cheaper than $ 1,000 [3] and the current Guinness world record for fastest DNA sequencing technique takes a bit more than five hours [4]. The constant development in the field might decrease the cost and time consumption even further. As a result, sequencing is a standard method in biological and medical research to investigate a wide range of questions, from causes for diseases over differences between species to the function of singular genes [2].

This leads to a large amount of sequencing data and to projects aiming to provide a database for this data like the Sequence Read Archive [5] or GenBank [6]. Currently, the Sequence Read Archive contains more than 30 petabytes of sequencing data [7, 8]. Considering the exponential growth over the last years (as depicted in Figure 1.1) and projects like the GenomeAsia 100K project [9] or the American 1,000,000 genome project [10], which seek to sequence even more data, the size of these databases are expected to grow even further.

Accumulating and storing all this data only makese sense, however, if it can be either analyzed on a large scale or if can be reused after the first analysis.

Biomedical research could be advanced further by analyzing sequencing data on such a large scale [11]. For instance, population studies depend on a large collections of sequencing data to account for the diversity within a population or in-between multiple populations. Understanding individuals genomes and the differences better could lead to more targeted medicine and help with fulfilling the dream of personalized medicine [12]. Moreover, machine learning methods are entering the field of bioinformatics, including sequence analysis, due to their success in other fields like image analysis [13, 14]. As machine learning methods rely on large collections of data for their training, they are dependent on the possibility to utilize existing

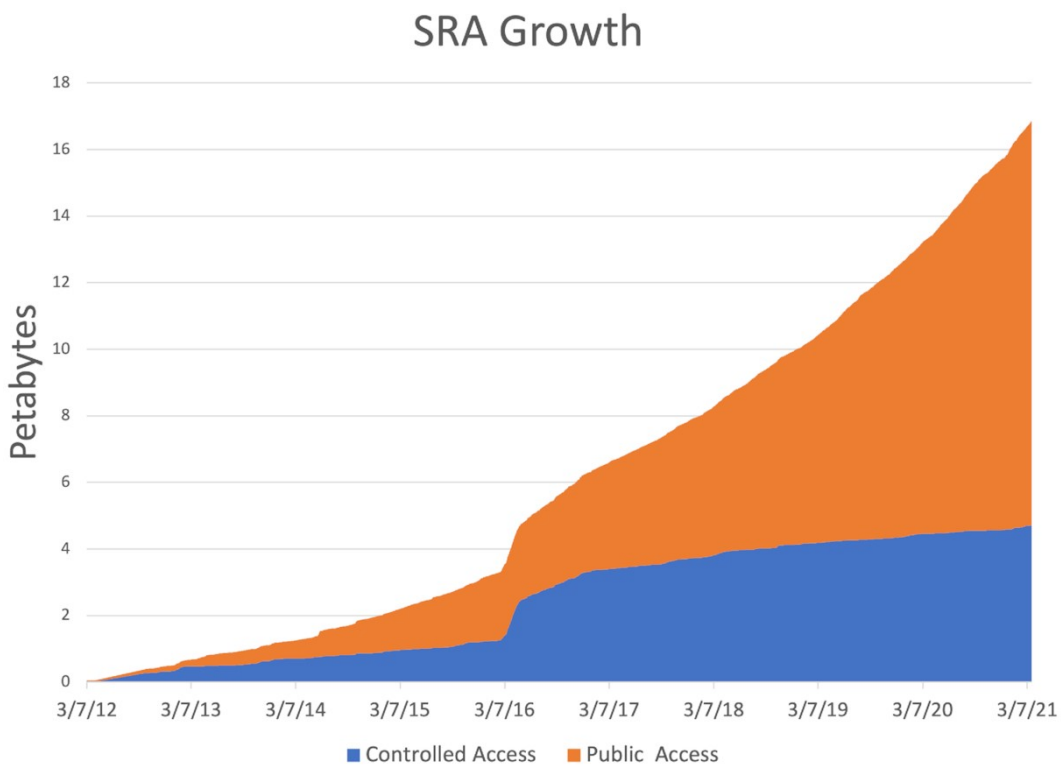


Figure 1.1: **Growth of the Sequence Read Archieve.** The figure is taken from [7].

data [14].

The need for analyzing large collections of sequencing data led to databases like the Cancer Genome Atlas [15], which enable such analysis by preprocessing the data. While such databases are a valuable resource [16], databases like this need to constantly update their processed data to incorporate new information like a newly annotated transcript [17]. Furthermore, currently these databases only cover a small percentage of the available sequencing data, for example the Cancer Genome Atlas covers around 2 Petabytes of sequencing data [17], which is roughly 7 % of the whole Sequence Read Archive.

Reusing sequencing experiments can advance research as well by finding a new hypothesis, validating a thesis or by providing more data for a study without the need to sequence. If the sequencing databases were searchable for transcripts, one could use them to find possible candidates connected to a condition of interest. For instance, finding transcripts that are present in cancer samples, but not in healthy tissue and then do more refined research on these found transcripts. Similarly, if a research group has found a transcript in their experiment to be involved in a disease, they could validate their findings by searching all existing sequencing data of that disease. However, reusing sequencing experiments can only be achieved under the condition that experiments of interests can be found in the sequencing

databases. Currently though, experiments can either be retrieved solely based on the given metadata, not on the sequences itself, or by searching through a subset of the database [11]. Both approaches are limiting and the reliance on metadata is especially problematic as it can be incomplete or outdated, therefore making it impossible to find all experiments of interest.

The reason why analyses on a large collection of sequencing data or determining all experiments of interest based on the actual sequencing data is not possible, is that established strategies to handle sequencing data do not scale to the amount of existing sequencing data. This holds for strategies indexing sequencing data like the Burrows-Wheeler Transform [18], the FM-index [19] or the suffix tree [20] as well as for strategies to analyze the data, which typically includes a costly alignment step, where the sequencing information is aligned to a known genome.

Approaches that work well in other fields handling similar amounts of data like online search engines are not applicable to sequencing data for a number of reasons. Among others, sequencing data, especially when considering metagenomics, has a higher magnitude of unique terms compared to documents based on natural languages. Moreover, search engines are typically optimized for the top ten results and on exact matches. In biomedical research due to the diversity of biological data, approximate results are of interest and so are all results not only the top [21].

Furthermore, one important field of interpreting sequencing data is to determine gene expression. Gene expression is a measure of transcriptional activity of a gene, which is related to abundance of its respective protein or non-coding RNA [22]. By determining gene expression of sequencing data, the underlying mechanisms of phenotypes can be better understood. This can improve medical research by aiding the development of treatments, when comparing the gene expressions between healthy and disease phenotypes [23]. Therefore, in order to utilize the large sequencing data collections completely, it is necessary to have a quantitative index.

To understand the requirements such a quantitative index needs to fulfill, the typical analysis of a small sequencing data set is sketched here.

Gene expression can be measured based on microarrays or on RNA sequencing (RNA-Seq), but as RNA-Seq is nowadays the more common approach [24], the focus of this thesis will lay on RNA-Seq.

There are different protocols for RNA-Seq (see [25] for an overview). Depending on the protocol, the preparation of a sample differs as well as the outcome. In general, RNA is isolated as a first step. Then either adapters are attached to the RNA directly or to cDNA, which has been synthesized from the RNA. Following this step is amplification, increasing the amount of RNA/cDNA, which then is sequenced from one end (single read) or both ends (paired). The resulting sequences are called reads and their lengths depend on the used sequencing technology [22, 25]. Most

commonly, these reads have a length of 75 to 400 bp (short reads), but recently with advance of sequencing technology long reads of length $1,000$ to $50,000$ bp (long reads) gained more and more traction [25, 26]. Short and long reads do not only differ in their read length, but also in the number of reads that can be produced, which is in the range of 10^9-10^{10} for short reads and 10^6-10^7 for long reads. Moreover, long reads have a higher error rate than short reads [25].

These reads are then typically matched with a reference genome or reference transcriptome by aligning them. Due to sequencing errors, mutations and repetitive regions, these alignments can be imperfect by either not aligning completely or uniquely [27]. Based on the number of these mapped reads, transcripts and genes can be quantified and then compared within a sample or between samples. Depending on the comparison, different normalization methods are appropriate because transcripts and genes within a sample have different lengths and the longer a transcript or a gene, the more likely it is to have a higher number of mapped reads. Additionally, comparisons between samples and experiments need to account for differences in sequencing depth [28].

Aligning the reads to a reference is costly in terms of space consumption and run time. Therefore, in order to handle large collections of sequencing data, an alternative to aligning became necessary and was found with alignment-free methods [14]. Alignment-free methods are not new [14, 29], but applications employing these methods are a recent development, see for example [30, 31].

How alignment-free methods can be applied to determine gene expression will be explained in detail in chapter 2, before exploring these methods on large collections of sequencing data in chapter 3.

Moreover, chapter 3 gives an overview of indices simplifying the sequencing data thereby reducing the space consumption and enabling faster searches in order to make sequencing data reusable and to enable (new) analyses on larger collections of data. Most of these methods do not provide a quantification, but solely answer the question of whether a transcript is present or absent in a sequencing experiment [32]. Therefore, the focus will be on the quantitative index by Darvish et al. named *Needle* [32].

Afterwards in chapter 4 analyses of large collections of sequencing data with *Needle* are shown to prove the usefulness of utilizing these data collections. These analyses include the finding of cancer signatures as well as validating a newly annotated mouse transcript.

Chapter 2

Alignment-free quantification

The typical workflow to quantify transcripts from a RNA-seq experiment is to align the reads to a known reference genome and count the number of aligned reads for a transcript. This count value is then normalized and used as an expression value, which allows the comparison to other transcripts within the same experiment or to the same transcript between different experiments [27].

This workflow has one major drawback: As the number of RNA-Seq experiments increases, so do the computational power and storage required for alignment.

For this reason, alignment-free methods were introduced to speed up analysis. Instead of aligning reads to a known genome or transcriptome, information of the sequence is obtained by counting the occurrences of words (subsequences of a certain length), considering common or unique substrings between samples or different methods based on information theory [33, 34]. The purpose of most developed alignment-free tools is not to determine transcript quantification though, instead they were designed for genome assembly and metagenomics (see [34] for an overview).

In 2014, *Sailfish* was introduced as the first alignment-free tool for transcript quantification [35], which was then shortly followed by *RNA-Skim* [36] in the same year, *kallisto* [37] in 2016 and *Fleximer* [38] and *Salmon* [39] in 2017.

The basis of most alignment-free methods for transcript quantification is the usage of small subsequences of a given sequence to determine its most likely origin in a transcriptome. These subsequences are from here on called *submers* (note that the term was used slightly differently by Edgar [40]).

Here, a new method called *Needle count* is presented. The idea of *Needle count* is to count the occurrences of submers in a sequence file and quantify a transcript simply by using the median of the count values of submers in the queried transcript. Despite its simplicity, *Needle count* is competitive with the above mentioned tools as will be shown later in 2.3.

Before going into the actual methods used by these tools, an overview of submers

is given as there has been a lot of development over the last years and an in-depth understanding is beneficial for understanding the implementation choices made for *Needle count*.

2.1 Submers

The purpose of submers is to simplify the sequencing data and to make analysis more affordable by reducing the run time and the memory consumption, while still capturing the underlying information. Therefore, the following three criteria are good metrics to compare submer methods:

1. Run time and set size
   The number of created submers is of interest because a smaller set size means the submers need less space. This leads to a smaller memory footprint and enables faster computation in downstream analysis, resulting in an overall decrease in run time.

2. Uniqueness of a submer
   The uniqueness of a submer is relevant for determining the genomic origin of a submer [41]. If a submer is unique in a genome, its origin can be exactly located under the condition that there are no sequencing errors or mutations.

3. Impact of sequencing errors and mutations
   Sequencing errors and mutations play a crucial role in sequence comparison as they introduce mismatches or insertion and deletions between otherwise similar sequences, therefore the impact and the handling of sequencing errors and mutations is of interest.

To determine the best submers for a task is still discussed and researched (for example see [42]), but so far the research has mostly concentrated on alignment-based applications, here the focus will be on alignment-free applications.

2.1.1 k-mers

The most frequently used submers are k-mers, which consist of all subsequences in a given sequence of length k [43, 44]. If the reverse complement strand is also considered, a k-mer is called canonical, if it is the smaller k-mer between itself and its reverse complement.

As an alternative to these contiguous k-mers, gapped k-mers (also called spaced k-mers [45, 46]) were introduced with the goal to make k-mers more robust to sequencing errors and mutations. A gapped k-mer is a k-mer, where some g positions in the k-mer are not considered (called gaps). For example, for $k = 5$ and $g = 2$

with a gap at the second and the fourth position (symbolized by the shape "10101" with "0" marking a gap) the k-mers "ACGTA" and "AGGCA" would be equal to each other as they are both seen as "AGA".

Run time and set size

Obtaining all k-mers in a sequence can be achieved in linear time by iterating over every position. Because it is necessary to consider every position, it is not possible to achieve a faster than linear time. A fast method to construct ungapped k-mers is to determine the first k-mer in a sequence and then shift to the next k-mer by dropping the first nucleotide of the k-mer and adding the next nucleotide to the end [47]. As this does not work for gapped k-mers because adjacent gapped k-mers do not overlap, obtaining gapped k-mers should take more time.

For a given sequence of length s_n, $s_n - k + 1$ k-mers are obtained. Usually and especially for genomes, not all of these k-mers are unique, therefore, the set of all k-mers is typically smaller than $s_n - k + 1$. In theory, a set could have a maximum size of 4^k, in practice, the size is considerably smaller.

Gaps do not influence the number of obtained k-mers. However if gapped k-mers have the same length as ungapped k-mers, the final set size could be smaller because the number of possible gapped k-mers is smaller than the number of all possible k-mers. More precisely, the difference is $4^k/4^{k-g} = 4^g$. For this reason, it is generally better to compare gapped k-mers to ungapped k-mers such that the number of considered nucleotides are the same, which is the case when the number of 1's in the shape of the gapped k-mers is equal to the k of the ungapped k-mer.

Uniqueness

The uniqueness of k-mers is dependent on the sequence and the choice of k. The greater k is, the more likely it is that the resulting k-mer is unique.

Impact of sequencing errors and mutations

A typical metric of sequence comparison with k-mers is based on the number of shared k-mers between two sequences. If a sequence A is the same as a sequence B, A shares all its k-mers with B. Mutations or sequencing errors can introduce mismatches, insertions or deletions between these two sequences, the so called k-mer lemma gives then the minimal number of shared k-mers, given e errors[1] and a sequence length of A s_A:

$$s_A - k + 1 - e \cdot k \tag{2.1}$$

[1]The term errors here includes mutations as well.

shared k-mers [48].

With an increasing number of possible mismatches e due to a high sequencing error rate or a high mutation rate, the k-mer lemma becomes less meaningful, as the number of minimal shared k-mers becomes 0.

For gapped k-mers, the k-mer lemma holds as well, but often returns too small a threshold [48]. Considering the example of the gapped k-mer "101" for a sequence of length $s_n = 5$ and one error $e = 1$, the k-mer lemma returns zero shared k-mers with the sequence without an error. However, even if the error is at the worst position in the sequence, at least one k-mer remains intact (as can be seen in Figure 2.1).

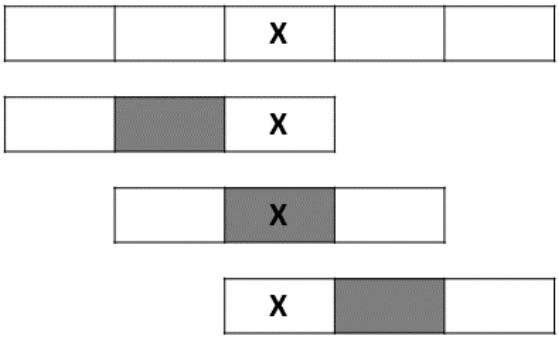


Figure 2.1: **Example of gapped k-mers "101" in a sequence with one error.** Gaps are highlighted in grey, the error is marked with an "X". The first and the last k-mer do not match due to the error, but the k-mer in the middle is still a match because the error falls into the gap.

2.1.2 Strobemers

In 2021, strobemers were proposed as an alternative to k-mers. Strobemers are motivated by the observation that k-mers (including gapped k-mers) do not handle insertions and deletions well when used for sequence comparison, because k-mers can not match over a region containing an insertion or deletion [41].

Strobemers are constructed by combining multiple shorter submers of a certain length with gaps over variable length [41]. The fact that the gaps are not predefined makes it possible to disregard insertions and deletions (see Figure 2.2 for an example).

Originally, Sahlin introduced three different types of strobemers: minstrobemers, randstrobemers and hybridstrobemers [41]. Recently, three new strobemer types were introduced [49], however, these three new types are beyond the scope of this work and not further discussed.

For minstrobemers, randstrobemers and hybridstrobemers, a user needs to choose an l for the length of the shorter submers called strobes, an order n defining how many strobes should be combined, a w_{min} marking the start of the sequence position

offsets for picking the strobe and a w_{len} giving the length of the windows. Strobemers are from here on denoted by (n, l, w_{min}, w_{len}). Because a strobemer consists of n l-mers, it can be compared to k-mers of length $n \cdot l$ [41].

Note that the definition is slightly different to Sahlin's definition [41] as there a w_{max} is used as the end of sequence position offset, instead of the length of the window, therefore $w_{max} = w_{len} + w_{min} - l$.

All strobemers are constructed by iterating over a given sequence: at every position i, the submer of length l is obtained (m_1), this is the first part of the strobemer, then the sequence for the positions $[i+w_{min} : i+w_{min}+w_{len})$ is considered and a strobe is picked according to the rules of the strobemer type, this is the second part of the strobemer (m_2). If $n == 2$, then the strobemer consists of only two strobes m_1 and m_2. If the order is greater than 2, then the sequence for the positions $[i + w_{min} + w_{len} : i + w_{min} + 2 \cdot w_{len})$ is considered for finding the third part of the strobemer (m_3). Generalized, for the jth part of the strobemer m_j with $j > 1$ the sequence in the window $[i + w_{min} + (j-2) \cdot w_{len} : i + w_{min} + (j-1)w_{len})$ is taken into account [41].

Similar to canonical k-mers, canonical strobemers are defined as the minimal strobemer between itself and its reverse complement when considering the reverse complement strand.

Minstrobemers

For minstrobemers, the m_j with $j > 1$ is picked by comparing all l-mers in the corresponding window $[i + w_{min} + (j-2) \cdot w_{len} : i + w_{min} + (j-1)w_{len})$ and picking the smallest submer [41] (see Figure 2.2 for an example).

Randstrobemers

For randstrobemers, the already picked l-mers in previous positions influence the choice of m_j with $j > 1$. This is achieved by calculating a hash value for all previous l-mers (m_1 to m_{j-1}) concatenated with every l-mer in the corresponding window $[i+w_{min}+(j-2)\cdot w_{len} : i+w_{min}+(j-1)w_{len})$, the l-mer in the window that minimizes the hash value is then picked as m_j. Any hash function accepting a sequence string and returning a numeric value would work for this approach [41] (see Figure 2.3 for an example). Finding a good hash function is not trivial and is an active field of research [50].

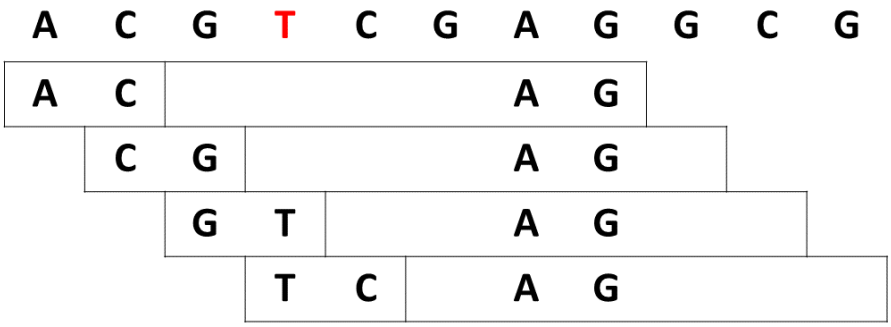


Figure 2.2: **Example of (2,2,2,6)-minstrobemers.** The first part of the minstrobemer is the first 2-mer, which is then combined with the smallest 2-mer in the following window. The second part of all minstrobemers does not change as the minimum remains the same. The resulting minstrobemers are: ACAG, CGAG, GTAG, TCAG. If the nucleotide 'T' marked in red is deleted, the first two minstrobemers remain the same. If a nucleotide is inserted before it, the second minstrobemer remains the same. This shows that strobemers can be more robust to insertions and deletions than k-mers, where all four k-mers would have been impacted by insertion or deletion.

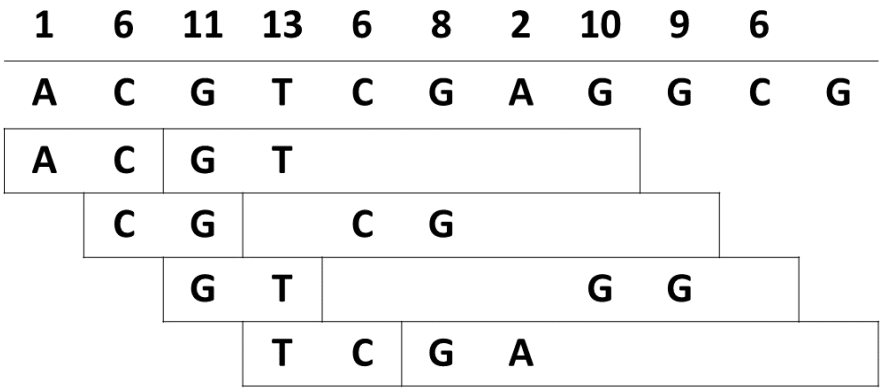


Figure 2.3: **Example of (2,2,2,6)-randstrobemers.** The first row gives hash values for every 2-mer starting at that position. The first part of the randstrobe (a) is the first 2-mer. The second part is picked by choosing the first 2-mer in the following window that minimizes function $f(a,b) = (h(a) + (hb))\%3$, where $h(x)$ refers to the hash values. This function is just used to illustrate the randstrobemers better, but any function accepting two hash values would work. For the first randstrobemer, this is $f(AC, GT) = (1 + 11)\%3 = 0$. Both 2-mers are then combined to one randstrobemer. The resulting randstrobemers are: ACGT, CGCG, GTGG, TCGA.

Hybridstrobemers

Hybridstrobemers intend to combine the approach of minstrobemers and randstrobemers. This is achieved by creating x disjoint groups of the l-mers in the corresponding window of m_j $[i + w_{min} + (j-2) \cdot w_{len} : i + w_{min} + (j-1)w_{len})$, the rth group is considered with $r = h(m_{j-1})\%x$ and the minimizer in this group is then picked as m_j (see Figure 2.4 for an example). Therefore, by letting the previous submer influence

the chosen group, previous submers like in randstrobemers have an influence while the choosing method in a group equals the approach of the minstrobemers [41].

In the study from Sahlin [41] $x = 3$ was used. Similar to the randstrobes any hash function h accepting a sequence string and returning a numeric value would work for this approach.

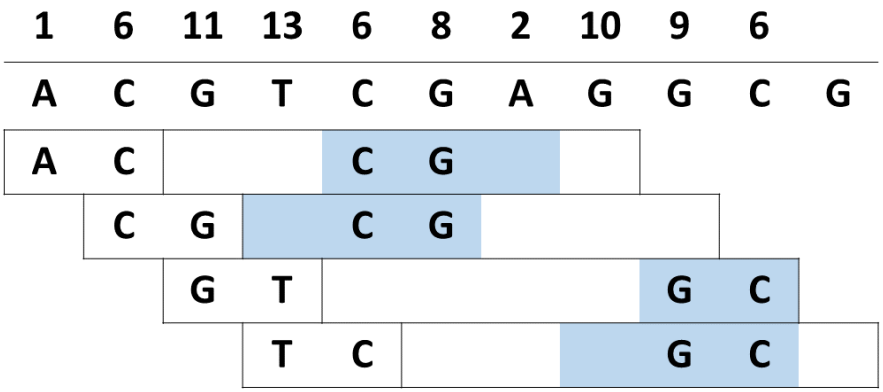


Figure 2.4: **Example of (2,2,2,6)-hybridstrobemers.** The first row gives hash values for every 2-mer starting at that position. The first part of the hybridstrobe is the first 2-mer. The second part is picked by dividing the following window in 3 disjoint groups, where the first two groups contain 2 2-mers and the last group one 2-mer as the window contains in total 5 2-mers. The subwindow containing the rth group (here marked in blue for each hybridstrobemer) is considered according to the function $r = h(m_{j-1})\%x$ with $x = 3$. In the rth group the 2-mer with the smallest hash value is picked. For the first hybridstrobemer, the $r = h(AC)\%3 = 1\%3 = 1$th group is considered, the smallest hash value in that subwindow is associated with CG, therefore this is the second part of the hybridstrobemer. Both 2-mers are then combined to one hybridstrobemer. The resulting hybidstrobemers are: ACCG, CGCG, GTGC, TCGC.

Run time and set size

Like k-mers, obtaining all strobemers takes linear time because an iteration over every position is necessary. Minstrobemers are expected to be the fastest strobemer type. This is because all strobes besides the first one are obtained by taking the minimum in a window and once a minimum is known for one window, the following strobemers only need to compare to this minimum, unless the minimum is the very first l-mer and falls out of the bounds of the window, then all l-mers in a window need to be compared, but this unlikely to happen with every shift.

Randstrobemers on the other hands are expected to have the longest run time as the strobes have to be calculated every shift again because they are depending on the first strobe.

Hybridstrobemers should be fast in computing a minimum as only a subset of

all l-mers in a window are taken into account, but because the subset changes depending on the first strobe, it is more likely that the minimum calculation has to be repeated every shift. Therefore, hybridstrobemers are expected to be slower than minstrobemers but faster than randstrobemers.

For a given sequence of length s_n, $s_n - (w_{len} \cdot (n-1) + l) + 1$ strobemers are obtained, but similar to k-mers not all of them are unique, therefore, the set size can be smaller. The theoretical maximum set size is for strobemers the same as it is for k-mers.

Uniqueness

The uniqueness of strobemers is dependent on the sequence and the choice of parameters. As minstrobemers are the most likely to share their strobes between shifts, they are also most likely to result in the least amount of unique strobemers.

Impact of sequencing errors and mutations

Considering two sequences A and B, which differ only in e positions due to mutations or sequencing errors, the minimal number of shared strobemers is: $s_A - (w_{len} \cdot (n-1) + l) + 1 - e \cdot (w_{len} \cdot (n-1) + l)$ with s_A being the sequence length of A. This is due to the fact that $(w_{len} \cdot (n-1) + l)$ is the span of the sequence that is taken into account during the strobemer construction (in all three strobemer examples above, this equals the length of both boxes).

However, this is a rather loose fit, as strobemers can be unaffected by errors (see above Figure 2.2 for an example). As the first strobe in a strobemer is always set, strobemers where the first strobe covers the error will always be impacted, therefore, at least l strobemers are impacted by one error (in Figure 2.2 the last two strobemers would be impacted by any error regarding the marked nucleotide 'T').

Hence, in the best case that only the first strobes are impacted by errors sequences A and B share $s_A - (w_{len} \cdot (n-1) + l) + 1 - e \cdot (l)$ strobemers.

For the other strobes one of the following three cases can apply, where m_j refers to any strobe besides the first one and the window j is the corresponding window m_j originates from:

1. The error occurs in the sequence of m_j.

2. The error occurs in the window j, but not in the sequence of m_j and has an impact on m_j as a different l-mer is now picked as strobe m_j.

3. The error occurs in the window j, but not in the sequence of m_j and it has no impact on m_j.

How likely it is that the first case occurs is depending on the number of l-mers considered in one window. If the number of l-mers in one window is small, then it is even possible that every l-mer is impacted by an error and therefore every strobemer covering the error position is affected. On the other hand, too large a window is also not desirable as the likelihood of a window spanning over more than one error increases with the window size.

The minimal number of unaffected l-mers in a window can be calculated with the k-mer lemma (Equation 2.2), which is $w_{len} - l + 1 - l \cdot e$, where $w_{len} - l + 1$ is the number of l-mers in one window. Therefore, the probability of case 1 can be estimated by: $1 - \frac{1}{w_{len} - l + 1 - l \cdot e}$.

The likelihood of the second case depends on the type of strobemers. For minstrobemers, an error can introduce a new minimum and therefore change the strobe m_j. If the distribution of l-mers was random, the probability that a new l-mer is the new minimum is $1/(w_{len} - l + 1)$. As an error can introduce multiple changed l-mers, this probability has to be multiplied with the number of changed l-mers.

Randstrobemers and hybridstrobemers on the other hand also depend on the previous strobes and therefore it is less likely that an error somewhere else in the window than in the actual strobe has an impact.

Therefore, the probable amount of strobemers impacted differs on the type of strobemer. Calculating a tight lower bound on the minimal number of shared strobemers similar to the formula based on the k-mer lemma (Equation 2.2) given above is hard due to the probable nature that errors have in strobemers.

Moreover, the distribution of affected strobemers also depends on the strobemer type.

Minstrobemers tend to pick similar strobes for adjacent strobemers as the minimum often remains the same. Therefore, if an error impacts the minimum directly by changing the sequence of the minimum or by creating a new minimum in the window, it is likely that the adjacent minstrobemers are impacted as well.

Randstrobemers on the other hand are more likely to pick different strobes for adjacent strobemers, therefore it is likely that an error hits one strobe, but it is rather unlikely that the adjacent randstrobemers are impacted as well.

Hybridstrobemers as a combination of the other two types will probably perform somewhere in the middle.

Therefore, one would expect the randstrobemers to be the best distributed strobemers.

2.1.3 Comparison of k-mers and strobemers

Besides a theoretical analysis, a practical evaluation and a comparison between different methods on actual data sets can give a better insight. Therefore, the project *minions* [51] was developed. *Minions* is a software project based on *SeqAn* [52] and offers two things: Firstly, it contains every submer method discussed here that is not part of *SeqAn* and therefore, *minions* can be used as library for these methods. Secondly, it consists of benchmarking methods and even has snakemake [53] files to easily reevaluate the methods. As finding submers is an active research field, *minions* can be easily used as a benchmarking tool when a new method is found by just adding the new method to minions and repeating previous analyses.

In *minions*, a hash value is stored for the submer rather than its actual sequence. The hash values is obtained by translating the submer sequence into a number in base 4, where the nucleotides A, C, G, T represent the numbers $0, 1, 2, 3$ respectively and then converting the result into base 10. This is done for the k-mer sequences as for the complete strobemer sequence. Note that Sahlin uses this hash function for the singular strobes and then combines them with an asymmetrical function in the following way $h(m_1)/2 + h(m_2)/3$ for order 2 and $h(m_1)/3 + h(m_2)/4 + h(m_3)/5$ for order 3, where $h(m_x)$ represents the hash value based on the sequence of the strobe [41]. In [54], Sahlin argues that a symmetrical function is useful as it can improve alignment results and therefore Sahlin applies the function $h(m_1)/2 + h(m_2)/2$ for order 2 [54].

Besides the run time analysis, all methods were used in their canonical form. The shape of the gapped k-mers were generated randomly for 4 and 8 gaps, because the design of the optimal number of gaps and their position is challenging and an ongoing research topic [55, 56], but random shapes have been shown to work well in practice [57].

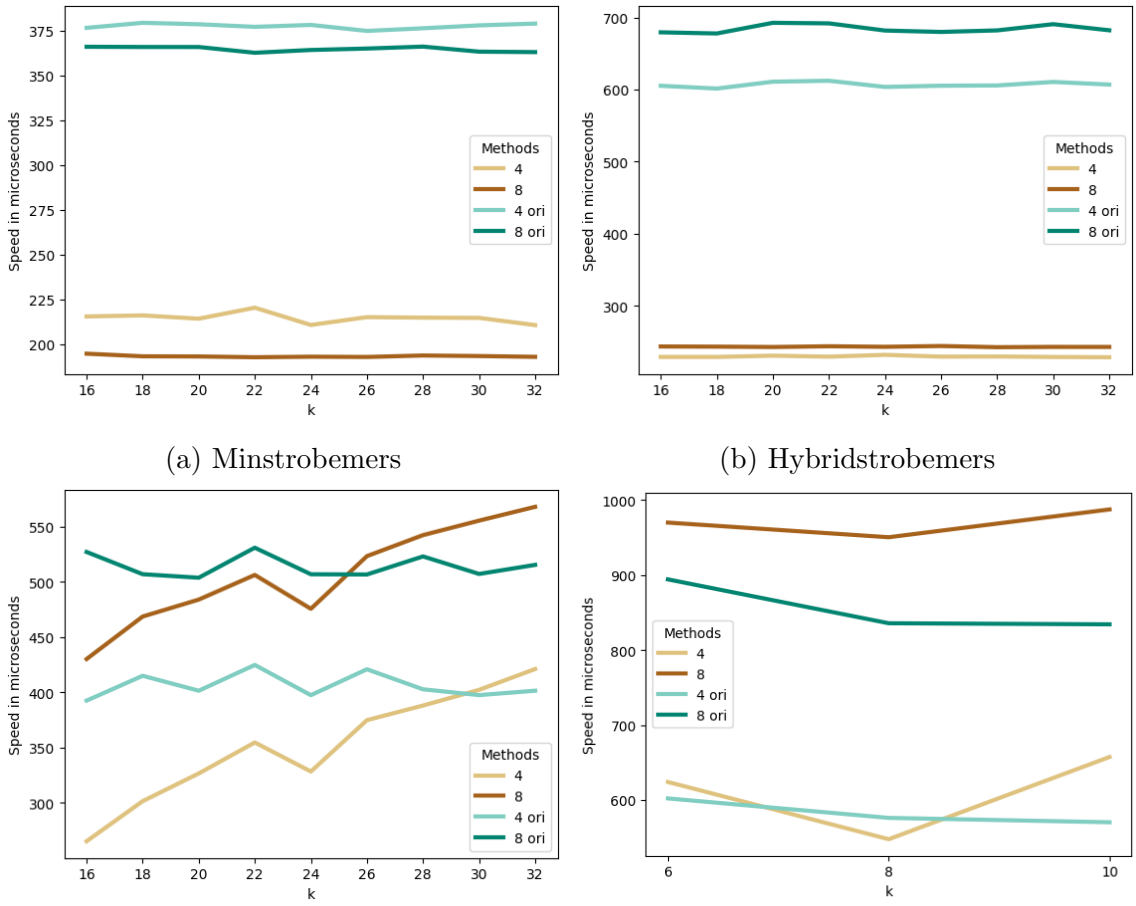
For the strobemers, the window length was chosen in such a way that the two strobes could have a maximal distance of 4 or 8, so they have the same span as the gapped k-mers.

Run time

The speed of the different methods was compared with the minions option *minions speed*, which returns the average time it took a method to process one sequence for all given sequences. For this analysis, $1,000$ random sequences of length $10,000$ *bp* were simulated with the tool *mason* [58] and tested with the different submer methods with different parameters. All submer methods were used in their non-canonical form, in order to eliminate possible influences from reversing the sequence.

The strobemer implementation used here and for all following comparisons is the

implementation integrated in *minions*, not the original one from Sahlin [41]. The minion implementation is faster than the original (see Figure 2.5) in almost all cases. The *minions* implementation is slower for randstrobemers of order 3 with all values of k and randstrobemers of order 2 with larger values of k.



(a) Minstrobemers

(b) Hybridstrobemers

(c) Randstrobemers

(d) Randstrobemers with order 3

Figure 2.5: **Average speed of strobemers for the original and the minion implementation.** 4 and 8 refer to the minion implementation of the strobemer type, which allow up to 4 or 8 gaps to the previous strobe, while 4 ori and 8 ori represent the implementation from Sahlin [41]. The strobemers have an order of 2, so the singular strobes have a length of $k/2$. With the exception of the randstrobemers with order 3, where k symbolizes the length of the singular strobe.

Moreover, the minion implementation uses less memory because it only returns the hash value of the strobemer, while the original implementation also returns the positions of the singular strobes. Therefore a comparison of the k-mers with the minion strobemers is a fairer comparison. Furthermore, the original implementation supports only order 2 for minstrobemers and hybridstrobemers, hence, the minion implementation also offers more features.

Figure 2.6 (and Figures A.1, A.2 for a window length allowing up to 8 gaps to the previous strobe and strobemers with order 3) show that the ungapped k-mer method

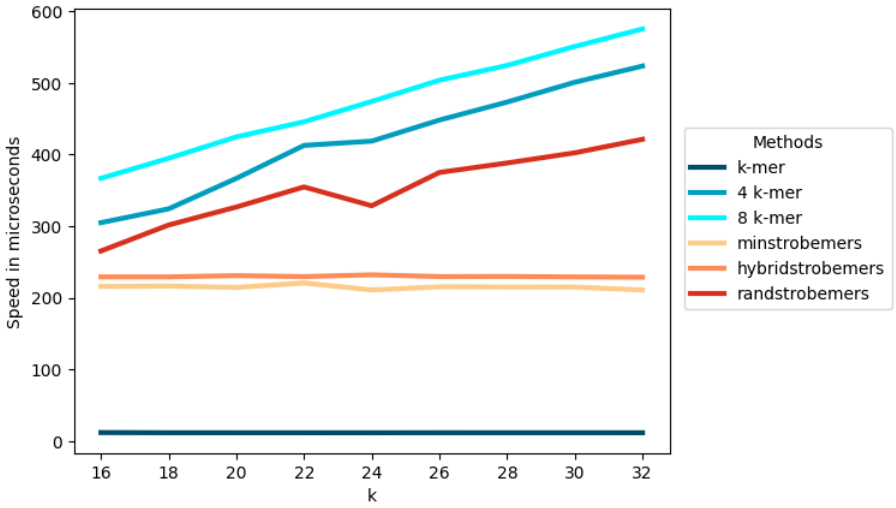


Figure 2.6: **Average speed of different submer methods.** 4 and 8 k-mer refer to the gapped $k+4$-mer or $k+8$-mer method with 4 or 8 gaps. The strobemers have an order of 2, so the singular strobes have a length of $k/2$ and they have a window length of $k/2+4$, so that a maximal gap of length 4 to the previous strobe is ensured.

is by far the fastest, while the minstrobemers and hybridstrobemers have a similar speed to each other and outperform the gapped k-mers. The randstrobemers is the slowest strobemer method. The randstrobemers and gapped k-mers are always the slowest methods, in Figure 2.6 the randstrobemers outperform the gapped k-mers, in the Figures A.1 and A.2 the gapped k-mers are faster.

The good performance of the ungapped k-mers is not surprising as they are the most simple method. However, the difference between strobemers and ungapped k-mers is in so far surprising, as the difference reported by Sahlin [41] is not that prominent.

Moreover, the fact that randstrobemers are slower than the other two strobemer methods aligns with the above formulated expectation, but it is reversed in Sahlin's analysis [41]. Sahlin hypothesized that his better performance of randstrobemers compared to hybridstrobemers is due the fact that randstrobemers do not have an overhead as they do not need a data structure storing all values within a window [41]. This also holds for the minion implementation, but the minion implementation does not store information about the positions of the singular strobes, therefore the overhead for the hybridstrobemers might have a smaller impact.

The fact that randstrobemers and gapped k-mers are the slowest methods shows that it is costly that no information from the previous adjacent submer can be kept. Therefore, at least in regards of the speed performance overlaps between submers are helpful.

Despite the relative difference in speed, all methods take less than a second and

therefore are reasonably fast.

Set size

The set size of the different methods was compared with the *minions* option *minions counts*, which returns the set size for a given sequence file. Here the human genome was used as input.

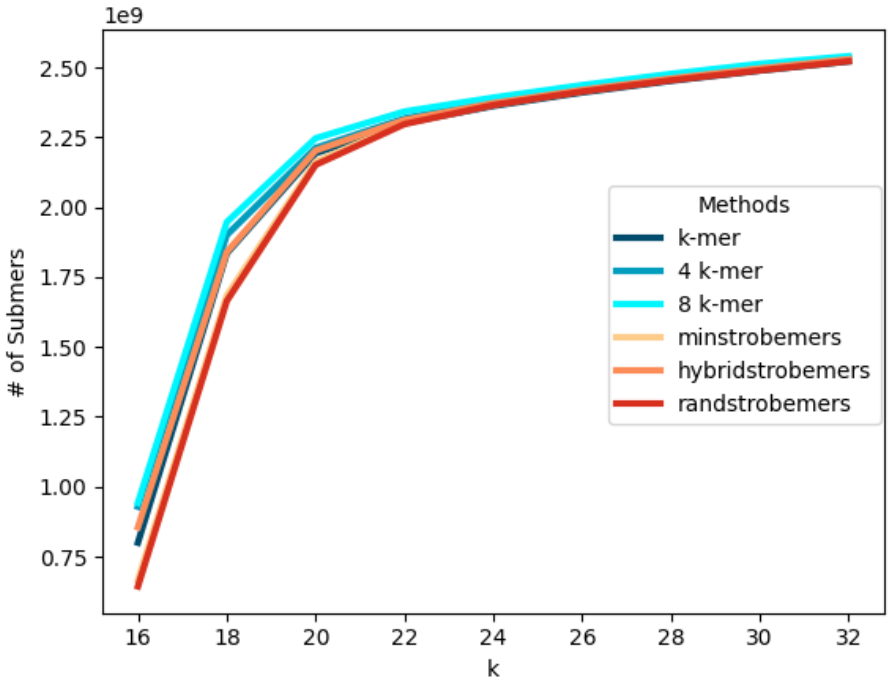


Figure 2.7: **Set size of different submer methods for the human genome.** 4 and 8 k-mer refer to the gapped $k+4$-mer or $k+8$-mer method with 4 or 8 gaps. The strobemers have an order of 2, so the singular strobes have a length of $k/2$ and they have a window length of $k/2+4$, a maximal gap of length 4 to the previous strobe is ensured.

Figure 2.7 (and Figures A.3, A.4 for a window length allowing up to 8 gaps to the previous strobe and strobemers with order 3) show that the strobemer methods have a smaller set size, but the difference is rather small and all methods seem to converge to the same set size.

Uniqueness

The uniqueness of the different methods was determined with the minions option *minions unique*, which returns the percentage of unique submers for the given files. For this analysis, the human genome was used.

The uniqueness shown in Figure 2.8 follows a very similar curve to the set size in Figure 2.7. The k-mer methods have a higher number of unique submers than

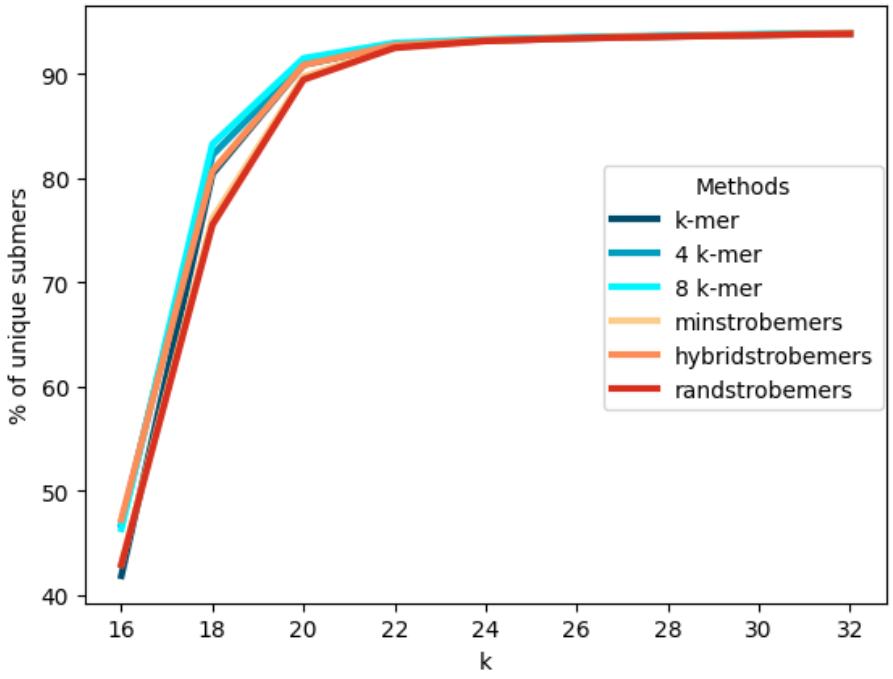


Figure 2.8: **Uniqueness of different submer methods for the human genome.** 4 and 8 k-mer refer to the gapped $k+4$-mer or $k+8$-mer method with 4 or 8 gaps. The strobemers have an order of 2, so the singular strobes have a length of $k/2$ and they have a window length of $k/2+4$, so that a maximal gap of length 4 to the previous strobe is ensured.

the strobemers, but the difference is insignificant and all methods seem to converge to the same value for increasing values of k.

For a submer length of 16, the percentage of unique submers is below 50 percent, which is concerning as 16 is a length commonly used [59, 60].

Moreover, as sequencing errors and mutations can impact submers, it is also of interest to know how unique the submers remain when considering errors. To determine this, the tool *genmap* [61] was applied. As *genmap* only supports ungapped k-mers, only these were checked, but based on the similar uniqueness without errors and the similar set sizes between the different methods, similar results can be expected for the other methods as well. Here, different ks with an error rate of 0 to 2 for the human genome were tested.

As shown in Figure 2.9, the percentage of unique k-mers increases with increasing k, but the more k grows the smaller is the increase. Furthermore, the percentage of unique k-mers is significantly smaller with errors, but with increasing k the difference becomes smaller as well.

Therefore, while there are differences between the submer methods, every method seems to be a valid choice when considering a greater value for k, which is also desirable due to its better performance in the presence of errors.

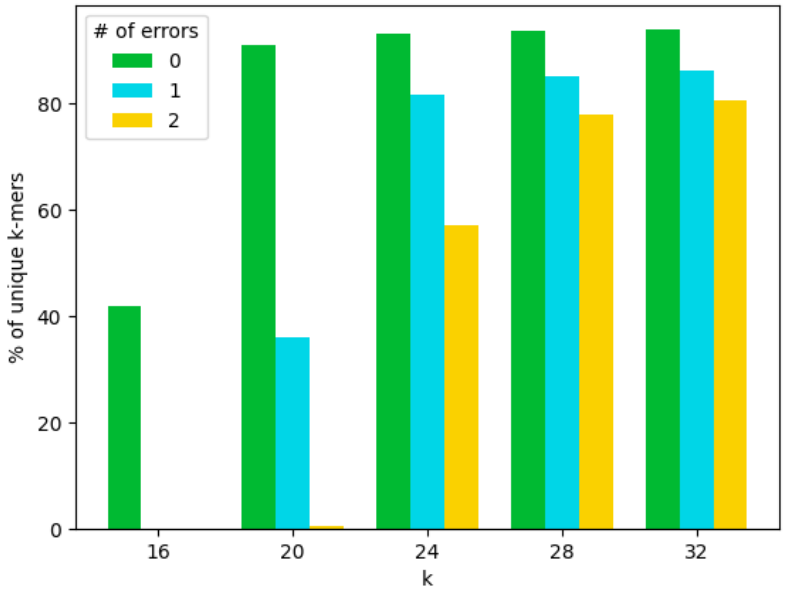


Figure 2.9: **Percentage of unique ungapped canonical k-mers in the human genome for different values of k and number of errors.**

Impact of sequencing errors and mutations

The impact of sequencing errors and mutations between two sequences can be measured by the match coverage [41] (also called conservation [40, 44]). A match between a sequence S and an erroneous or mutated sequence S' is defined as a submer that appears in both sequences at the same position and is therefore unchanged between the original sequence and the erroneous or mutated sequence. The match coverage is then the percentage of bases that are covered by the span of a matched submer. As the match coverage is based on the span of a submer, possible gaps in a submer are seen as covered as well. The match coverage in comparison to the number of matches is more stable due to overlapping submers (see Figure 2.10 for an example).

A single uncovered nucleotide or multiple adjacent uncovered nucleotides are called an island [41, 62]. The island size measures the amount of sequence without any information due to the lack of matching submers (see Figure 2.10).

Sahlin also considers a metric called sequence coverage, which measures how much of the sequence is conserved, even if the submer is not a match [41]. This is only of interest though, if alignment is involved, where even a submer with some level of discrepancy can be aligned to the reference. Hence, the sequence coverage is not of interest here.

The match coverage and the island size were determined by the minion option *minions match*, which returns the average values for all reads between two given

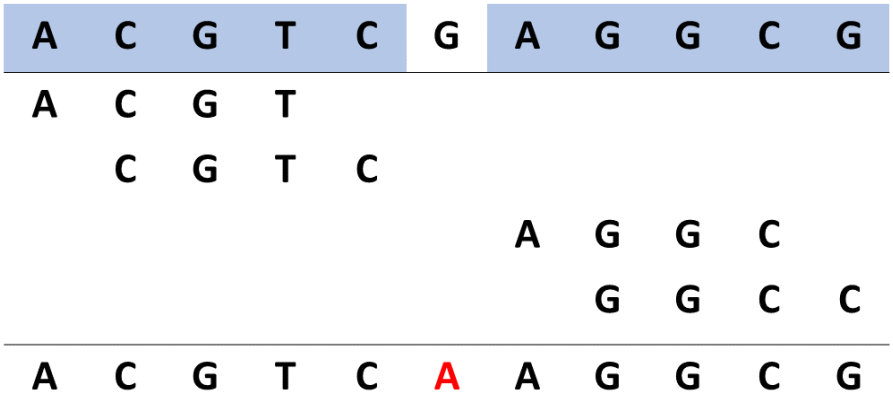
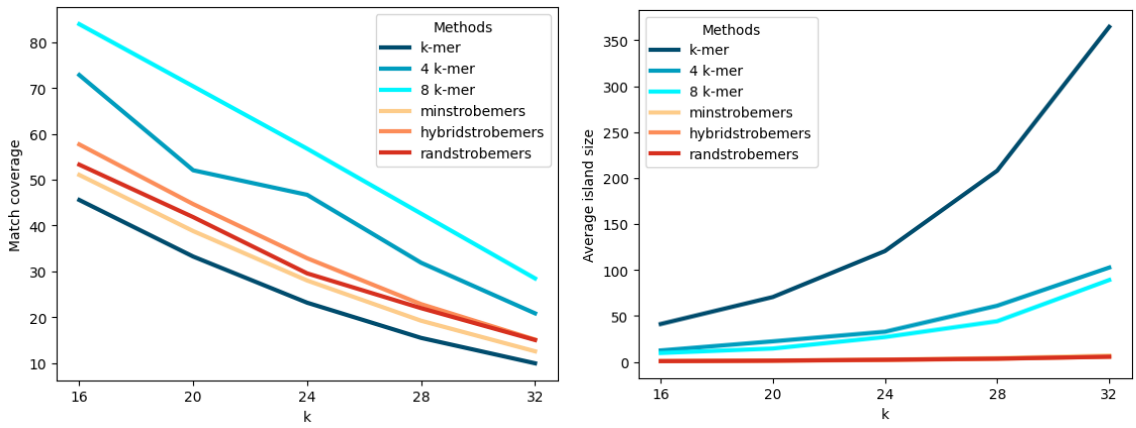


Figure 2.10: **Example for the match coverage and island size.** Two similar sequences (one on the top, one one the bottom) are shown, they only differ in one position marked in red in the bottom sequence. The number of matching 4-mers are 4 out of 8, shown in between the two sequences. The match coverage however is 10 out of 11 positions, based on the covered positions marked in blue in the top sequence. The match coverage better represents the similarity than the number of matches. There is one island as there is only one nucleotide not covered, which is marked in white. Therefore, the island size here is 1. If gapped k-mers with a shape of 1011 would have been used, then the gapped k-mer C-AG would also exist between both sequences and cover the different nucleotides, leading to a match coverage of 100 % and no islands.

sequencing files. Here, $100,000$ reads of length 100 bp were simulated based on the human transcriptome. A second file based on the previous one was created by randomly substituting $1, 2, 5$ and 10% of the nucleotides. These substitutions can represent either a sequencing error or a mutation. Neither sequencing errors nor mutations appear randomly though and can also consist of insertion or deletion [63]. However, insertions and deletions are less likely to occur as a sequencing error [64] and the impact of not considering them in the analysis should be minimal as the sequence coverage is not determined.

For a substitution rate of 1% the difference between the submer methods were not significant (see Figure A.6), but with an increase in the substitution rate differences became more apparent. Figure 2.11 (and Figure A.5 for a window length allowing up to 8 gaps to the previous strobe and strobemers of order 3) shows the match coverage and island size for the substitution rate of 10%. The ungapped k-mers have the lowest match coverage and the biggest average island size followed by the different strobemer methods for the match coverage and by the gapped k-mers for the average island size. It is interesting that the gapped k-mers have a higher match coverage than the strobemer methods, but a higher average island size, because a smaller average island size would be expected to lead to a better match coverage. That this correlation did not hold here could indicate that some methods had too



(a) Match Coverage

(b) Island Size

Figure 2.11: **Match coverage and island size of different submer methods with substitution rate 10.** 4 and 8 k-mer refer to the gapped $k+4$-mer or $k+8$-mer method with 4 or 8 gaps. The strobemers have an order of 2, so the singular strobes have a length of $k/2$ and they have a window length of $k/2+4$, so that a maximal gap of length 4 to the previous strobe is ensured.

many outliers in their island sizes, so that the average was skewed.

When comparing with strobemers of order 3 the 4 gap k-mers perform worse than the strobemer methods for increasing k, but the 8 gapped k-mers remain the best method. The strobemers used here can have up to 4 or 8 gaps between two strobes, but that does not necessarily mean they have to have gaps. For order 3, if 4 gaps are allowed between two strobes and there are three strobes in total, the strobemer could have in total 8 gaps and for 8 gaps 16 respectively. Therefore, the better performance of the strobemers of order 3 compared to the 4 gapped k-mers could be due to a higher number of gaps. Overall, it seems like the more gaps a submer has the better the match coverage.

However, these findings are contrary to the ones found by Sahlin [41], where randstrobemers of order 3 have the highest match coverage and the most sparse gapped k-mers the worst match coverage, even lower than the ungapped k-mers. Sahlin based his analysis on a simulated sequence as well, but also considered insertions and deletions with equal probability to the substitutions [41]. Therefore, insertions and deletions were the majority of the changes in his analysis. This is not a good representation of actual sequencing data as insertions and deletions are less likely to occur. It remains therefore an open question, which submer method would lead to the highest match coverage on actual sequencing data. An analysis on not simulated sequencing data seems like a promising approach to answer this question in the future. Nevertheless, such an analysis is not trivial as the ground truth needs to be determined beforehand.

Moreover, the impact of a sequencing error or a mutation is less severe, if it can

be accounted for. The k-mer lemma and the above described theoretical analysis of the impact of sequencing errors and mutation give some insight into finding a good threshold to determine, if a sequence with errors or mutation is still similar enough to the original sequence. In order to analyse this on a practical example, similar to the analysis of Seiler et al. [65], 64 highly similar genomes were simulated and then based on these genomes a total of $1,048,576$ reads of length 100 *bp* with 2 to 5 substitutions (called errors) were simulated. These sequencing files were then analyzed with *minions accuracy*, which builds a probabilistic data structure over all 64 genomes by storing their submer hash values and then checks for each read, how many submers are present in each genome. Given a threshold *minions accuracy* then considers a read originating from a certain genome, if the percentage of read submers found in a genome exceed the threshold. As the data is simulated, the actual origin of a read is known and the number of true positives, false positives, true negatives and false negatives can be determined.

	2 Errors		3 Errors		4 Errors	
Method	FP	OT	FP	OT	FP	OT
20-mer	**441**	0.5	**254**	0.25	47,457,980	0.02
4 gapped 24-mer	445	0.42	592	0.15	28,202,361	0.03
8 gapped 28-mer	504	0.32	1,297	0.15	**489,891**	0.09
$(2, 10, 10, 14)$-minstrobemer	446	0.37	657,009	0.09	-	-
$(2, 10, 10, 14)$-hybridstrobemer	459	0.37	9,167	0.12	45,729,774	0.02
$(2, 10, 10, 14)$-randstrobemer	451	0.37	176,923	0.1	60,551,536	0.02
$(2, 10, 10, 18)$-minstrobemer	504	0.23	43,932,027	0.02	-	-
$(2, 10, 10, 18)$-hybridstrobemer	504	0.26	5,010,511	0.06	43,869,773	0.02
$(2, 10, 10, 18)$-randstrobemer	504	0.27	5,055,046	0.06	43,963,988	0.02

Table 2.1: **Number of false positives and optimal threshold for different submer methods.** The table shows the number of false positives (FP) for the greatest threshold that still lead to zero false negatives (OT for optimal threshold). Three data sets with $2, 3$ and 4 errors were evaluated. The smallest number of false positives per column was marked in bold. In cases were there was no threshold with zero false negatives, FP and OT are marked with "-".

Table 2.1 shows the number of false positives for each submer method for the highest threshold, where the number of false negatives is still zero, because false positives are in most use cases less problematic than false negatives as false positives could be sorted out afterwards with other methods. All thresholds from 0.0 to 0.6 in 0.01 steps have been tested. The column for five errors was omitted because only the 8 gapped 28-mers and the $(2, 10, 10, 14)$-hybridstrobemers had results with zero false negatives.

For ungapped k-mers, the k-mer lemma can be used to determine a good threshold. Here, ungapped 20-mers were analyzed, according to the k-mer lemma (Eq.

2.2) $100 - 20 + 1 - 2 \cdot 20 = 41$ k-mers should be at least unaffected by two errors, 21 k-mers by three errors and 1 k-mer by four errors. In total, there are 81 k-mers in the sequence, hence a good threshold is $\frac{41}{81} \approx 0.506$ for two errors, ≈ 0.259 for three errors and ≈ 0.012 for four errorr. For five errors, the k-mer lemma (Eq. 2.2) would lead to an useless threshold of 0.0. As can be seen in Table 2.1, the thresholds as predicted by the k-mer lemma are really close to the ones practically measured and thereby proves the usefulness of the k-mer lemma for ungapped k-mers.

The k-mer lemma (Eq. 2.2) for the 4 gapped 24-mers would lead to a threshold of ≈ 0.377 for two errors and ≈ 0.065 for three errors, for four and five errors the threshold would be 0.0. For the 8 gapped 28-mers the k-mer lemma (Eq. 2.2) would lead to a threshold of ≈ 0.233 for two errors, for all errors the threshold would be 0.0. As expected, the k-mer lemma leads to poor predictions for the gapped k-mers, whose actual thresholds are significantly greater as the predicted ones.

For the strobemers, predicting a good threshold is not easy as explained above. If the errors would only impact the first strobe, then the thresholds should be ≈ 0.505, ≈ 0.258 and ≈ 0.01 for two, three and four errors respectively for strobemers with $w_{len} = 14$ and ≈ 0.325 for two errors for strobemers with $w_{len} = 18$, for all others the threshold would be 0.0. If the whole span of the strobemer would be considered, then the same thresholds as the one from the gapped k-mers would apply. Interestingly, the threshold for the span for two errors seems to be quite close to the actual ones, but for the other errors, this is not the case.

As predicted, randstrobemers and hybridstrobemers perform better than minstrobemers. Surprisingly, hybridstrobemers led to fewer false positives than randstrobemers.

Overall, all methods led to a similar number of false positives for two errors, while for three errors the ungapped k-mers have significantly fewer false positives than all other methods and all strobemer methods have significantly more false positives than the three k-mer methods. For four errors, the ungapped k-mers are not the best method anymore, instead the 8 gapped k-mers have the smallest number of false positives by far. As already mentioned above, only the 8 gapped k-mers and the $(2, 10, 10, 14)$-hybridstrobemers had thresholds with zero false negatives, there as well the 8 gapped k-mers outperforms the strobemer method.

These results show that finding a good threshold is not trivial and only for ungapped k-mers a solved problem. Furthermore, these result align with the results from the match coverage. As neither insertions or deletions have been considered, a future analysis including them would be beneficial to check, if the performance of the strobemers could be improved.

However, it should be noted that at least for the alignment-free application *Needle count*, mutations can be dealt with by searching for the changed submers

directly as the mutations of transcript of interests should be known beforehand.

The above theoretical and practical analyses of the impact of sequencing errors and finding a good threshold focuses on one singular read and its relationship to the reference genome. While there are applications, where reads are considered individually in the context of an alignment-free comparison (see for example [66, 67]), *Needle count* always considers all reads at once. Therefore, a single error has only a small impact as long as the genomic region an error covers incorrectly is also covered correctly by another read. The chances that a position in the genome is covered multiple times by different reads is depending on the read coverage. However, as the different submer methods handle errors differently, the needed read coverage is also based on the submer method.

Considering that high sequence coverage is affordable nowadays, that the quality of sequencing methods improved over the years with even long sequencing reads having an error rate of only 5 % [68] and considering the fact that multiple quality control mechanisms like trimming have been introduced [27], the importance of handling sequencing errors is not as important as it might seem at the first glance. Therefore, all methods seem to be a reasonable choice.

Conclusion of the comparison

Determining a good submer method is dependent on the task at hand but also on the order of priorities. The difference of the submer methods for the set size and the number of unique submers is small, but might be essential when space is a crucial factor.

In the speed and accuracy analysis, the difference between the methods seemed greater. However, the speed of even the slowest method is still fast and the accuracy analysis is based on a high percentage of substitutions, for a smaller percentage the difference becomes less prominent. Considering the fact that even for long reads the number of sequencing errors has been decreasing due to technology advances [68] and that sequence differences based on mutations are less severe for an approach like *Needle count*, which is not meant to find new mutations, any submer method seems to be reasonable enough.

Moreover, quite often only a representative set of submers (see the next section) are considered, this application of a representative submer method reduces the difference between strobemers and k-mers according to Maier et al. [49]. For that reason, a decision for a submer method can not be made without an analysis of the representative methods.

2.2 Representative submers

As the number of submers (k-mers or strobemers) is high and similar submers usually share the same information, different methods have been developed to find a representative set of submers. These representative sets aim to fulfill the two following main goals:

1. A small number of short representative submers to reduce the amount of data.

2. Finding a good representation, so as to minimize the amount of sequencing information lost.

These goals are usually opposed to each other: considering fewer submers leads to less information kept.

To measure if a representative submer method achieves the first goal the compression factor can be used as a metric. The compression factor as introduced by Edgar is defined by the number of submers divided by the number of representative submers, which is the inverse of the metric used in previous papers called density [40].

The second goal can be measured by multiple metrics and it is an active discussion, which are the best fitting ones [44, 69]. Similar to the analysis of the submers, the uniqueness of the set of representative submers and the impact of sequencing errors and mutations are good metrics to measure the representation and are included here in the practical analysis. Beyond these metrics, the expected distance between two adjacent representative submers can be determined. Distance thereby describes the difference between the positions of the first nucleotides of the adjacent representative submers [40]. The better distributed a method is, the more sequencing information is kept as close submers usually contain redundant information due to overlaps [40].

The developed methods of determining a representative set of submers can be distinguished by their construction, either they can be constructed directly from the given sequences (called "on the fly") or they are dependent on some kind of preprocessing step or additional information (see Table 2.2 for an overview).

A different way to distinguish these methods is by the impact errors have. Context dependent methods are methods, where an error outside the representative submer sequence can influence the picked representative submer, while context free representative submers can only be impacted by errors in their sequence [40]. All context free representative submer methods were marked with a '*' in Table 2.2.

As representative submer methods can work with both k-mers and strobemers, the variable k_l is introduced here, which is $k_l = k$ for k-mers and $k_l = n \cdot l$ for strobemers.

"on the fly"	preprocessing
Minimizers	Fleximers
Minimum decycling sets	Minimum decycling sets
Modmers*	Prefix words*
Syncmers*	Polar sets*
	Sigmers
	Universal hitting sets*
	Weighted Minimizers

Table 2.2: **Overview of representative submer methods** Submer methods marked with a '*' are context free methods.

2.2.1 Modmers

Modmers were introduced by Edgar as a baseline method for comparison (there called Modulo submer) [40]. A submer is considered a (k_l, m)-modmer, if the hash value of a submer of length k_l for a given hash function modulo m is zero. Therefore, the method is context free as an error can only impact a submer, if it appears in the submer sequence itself.

Assuming the given hash function leads to independent and identically distributed submers, the probability that one submer is selected as a modmer is $1/m$. Therefore, the expected number of representative submers for a sequence with n_s submers is n_s/m and hence, the compression factor of modmers is $n_s/(n_s/m) = m$.

The distance between two adjacent modmers has no maximal value, but the probability of larger distances decreases with each step because it means all submers in-between the adjacent ones are not modmers.

2.2.2 Minimizers

Minimizer were introduced independently of each other [70] by Roberts et al. [71] and Schleimer et al. (Schleimer et al. called it winnowing instead of minimizers) [72].

A minimizer is defined as the smallest submer of length k_l in a window of given length w, therefore minimizers are here denoted as (w, k_l)-minimizers (see Fig. 2.12 for an example). As adjacent windows share most of their submers due to their overlap, they often also share a minimizer, such a minimizer is only stored once, which reduces the memory cost.

As an error inside a window but not in the sequence of the minimizer itself can influence the minimizer by producing a new minimum, minimizers are not context free.

Using the lexicographical order as in the example in Fig. 2.12 to determine the smallest submer is less beneficial because it results in a skewed distribution. A ran-

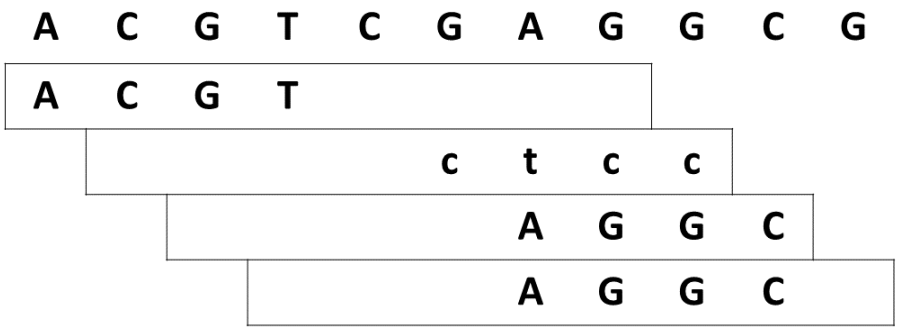


Figure 2.12: **Example of (8, 4)-minimizers.** The shown 4-mers are the smallest 4-mers in their respective window of size 8. The second minimizer originates from the reverse complement strand and is denoted by lower letters and has to be read from right to left. The last two windows share the same minimizer, as it is often the case for subsequent windows. They are shown twice, but are only stored once. The figure and the caption is taken from [32].

dom order on the other hand can prevent this [70] and has with $(w - k_l + 2)/2$ a smaller expected compression factor than minimizers based on the lexicographical order [40, 70, 72]. This formula for the compression factor holds under the assumption submers are independently and identically distributed, because then two adjacent windows share a minimizer, if the minimizer of the first window is not the first submer and if the minimizer in the second window is not the last submer. The likelihood that a submer is a minimizer in two adjacent window of length $w + 1$ is $\frac{1}{w+1-k_l+1} = \frac{1}{w-k_l+2}$. As two submers could lead to a new minimizer, the probability that a window shift introduces a new minimizer is $2 \cdot \frac{1}{w-k_l+2}$.

Even better compression factors can be achieved by applying different orders. Zheng et al. proposes an order based on the algorithm miniception [73], while Hoang et al. finds more compressed orders based on machine learning [74]. The machine learning approach turns the minimizers into a preprocessing method, while the miniception algorithm is evaluated on the fly, but is expected to slow down the construction. For these reasons these orderings were not further considered here.

Unlike modmers, minimizers guarantee that two adjacent minimizers have a maximal distance of $w - k_l + 1$ because every window contains a minimizer (this is called the window guarantee). Because each window shift can introduce a new minimizer, it is implied that each distance up to $w - k_l + 1$ is equally possible and follows an uniform distribution [40].

Minimizers for Strobemers

Instead of determining the minimizer based on the whole strobemer, Sahlin suggested to consider only the first strobe, if it is as minimizer given a window size,

the complete strobemer is [41]. If this is done with minstrobemers, the results are also known as paired minimizers [41, 75, 76]. However, this should result in most cases to the same minimizers as when the minimizers are determined on the whole sequence. For example, if "AC" is the minimal first strobe and all other first strobes are greater, then the smallest whole strobemer is also the one starting with "AC". A different result between these two approaches can only occur if the first strobe is the same between two strobemers. As for the whole strobemer approach the other strobes are taken into account, while for Sahlin's approach the tie is broken by picking the rightmost strobe. Because the results are expected to only differ slightly, Sahlin's approach was not used here.

Note, that Sahlin's minimizer approach makes a bigger difference when the minimizers are determined on the hash values and Sahlin's above described hash values [41] are used as then the other strobes influence the hash value more and can lead to completely different minimizers.

Weighted Minimizers

In highly repetitive regions, minimizers tend to be less informative when using a lexicographical or even a random order. Therefore, the weighted minimizers were introduced. Weighted minimizers are basically minimizers with an order function that discriminates against repetitive less informative minimizers, so that the chances that they are picked in a window are lower. This is achieved by giving as an additional input a list of these unfavorable minimizers. As this list has to be created beforehand for a specific genome, weighted minimizers can not be computed on the fly anymore and need a preprocessing step. In order to still have a small memory footprint, this input list is stored in a probabilistic data structure [77].

2.2.3 Syncmers

Syncmers were invented as a context free alternative to the context dependent minimizers by Edgar [40] and further generalized by Dutta et al. [69]. Syncmers are described by a subsequence length s, a list of positions pos and a boolean parameter $linear$. Therefore, syncmers are from here on denoted as $(k_l, s, pos, linear)$-syncmers.

A syncmer is any submer that has its smallest s-mer at any of the given pos positions (see Fig. 2.13 for an example). Syncmers which have a position list of length one are also called open syncmers [40] and syncmers which have a position list containing only the first and the last position are called closed syncmers [40]. If the boolean parameter $linear$ is false the syncmer is circular, which means the k_l-mers are seen as a cycle, for example for the 4-mer $ACGT$ with $s = 2$, the following

s-mers are obtained: AC, CG, GT, TA.

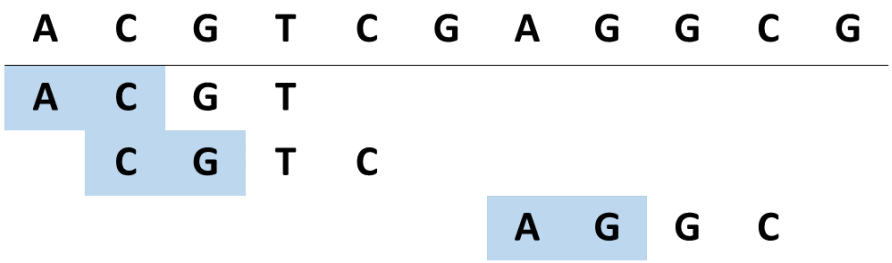


Figure 2.13: **Example of $(4, 2, [0], 1)$-syncmers.** The shown 4-mers are open syncmers as the smallest 2-mers are at the 0th-position of the 4-mers. As this does not hold for any other 4-mer in the sequence, there are only 3 syncmers.

Compression factor

The compression factor of syncmers is dependent on its parameters, because they influence the likelihood of a given submer being a syncmer.

Assuming that the s-mers are independent and identically distributed for a given submer, the probability that the submer is a linear syncmer is $n_{pos} \cdot \frac{1}{(k_l - s + 1)}$ with n_{pos} being the length of the list of position as their are n_{pos} positions where the smallest s-mer can be for it being a linear syncmer and there are $k_l - s + 1$ s-mers [40]. Therefore the compression factor is: $\frac{(k_l - s + 1)}{n_{pos}}$.

For circular syncmers, there are k_l s-mers in a k_l-submer, therefore the probability that a submer is a syncmer is $\frac{n_{pos}}{k_l}$ for circular syncmers [40]. Hence, the compression factors is $\frac{k_l}{n_{pos}}$.

Window guarantee

Linear closed non-canonical syncmers guarantee that two adjacent closed syncmers have a maximal distance of $k_l - s$ because the smallest s-mer in a sequence of length $2k_l - s$ will have to be at the start or end of a submer making it a closed syncmer (see Figure 2.14 for an example). Therefore, linear closed syncmers have a window guarantee of $k_l - s$ [40, 44].

Linear open non-canonical syncmers also have a maximal distance between two adjacent syncmers max_{los}, but it is harder to derive a closed form for this [40]. If a submer q is not an open syncmer, the smallest s-mer must be at a different position than the one position in pos. Without loss of generalization, let's assume the position in pos is set to zero ($pos = [0]$), then the smallest s-mer must be in the last $k - 1$ letters of the submer sequence. Assuming that this is not the last submer in the sequence, this smallest s-mer will be at the start of a submer that is maximal

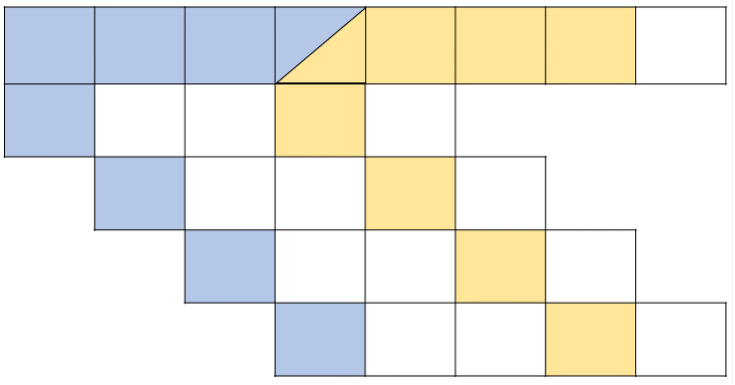


Figure 2.14: **Example of the window guarantee for closed non-canonical syncmers.** The here used syncmers are $(5, 2, [0, 2], 1)$-syncmers and the sequence length considered is $2k_l - s = 2 \cdot 5 - 2 = 8$. There are $8 - s + 1$ positions where a s-mer can start (marked in blue and yellow). These starting positions are at the start or end of at least one of the possible 5-mers of the sequence. Therefore, no matter which s-mer is the smallest, there will be at least one syncmer in the sequence of length $2k_l - s$.

$k - s$ positions away from q. This new submer might also not be an open syncmer, as this submer might now have an even smaller s-mer somewhere else. This new smallest s-mer will be at the start of another submer maximal $2 \cdot (k - s)$ positions away from q and might not be a syncmer, because another s-mer in the submer is smaller. This process can be repeated until the smallest possible s-mer is found [40].

For circular closed syncmers, it has not yet been proven, if a maximal distance exists or not [40]. While for circular open syncmers it can be shown that there is no maximal distance. As can be seen by the following counterexample for $(4, 2, 0, 0, 1)$-syncmers: The repetitive sequence "ACCACCACCA..." contains the 4-mers "ACCA", "CCAC" and "CACC", none of which are circular open syncmers [40].

In case more than two positions are considered for a syncmer, it is dependent on the positions if a window guarantee is provable. However, the more positions are considered for turning a submer into a syncmer, the less likely are great distances between adjacent syncmers.

Downsampled syncmers

Syncmers can be down sampled to obtain larger compression factors by using an additional parameter $d > 1$. Downsampled syncmers are syncmers, if the submer is a syncmer and the hash value of the complete submer itself is smaller than H/d with H being the largest possible hash value [40].

If the hash function is chosen in the way that the function is independent of the probability that a submer is a syncmer, the compression factor of downsampled

syncmers is the compression factor of the underlying syncmer $c_{syncmer}$ multiplied with d: $d \cdot c_{syncmer}$ [40].

There is no limit for the maximal distance between two adjacent downsampled syncmers.

Syncmers for strobemers

Instead of using the syncmers on strobemers, Sahlin proposed to build strobmers on top of syncmers. This is done by determining all syncmers in a sequence and then for every syncmer pair it with a following second syncmer (or more, if an order greater than 2 is picked) depending on the strobemer rules. Moreover, Sahlin adapted the strobemers for this approach by picking closer syncmers with a higher probability as the second strobe and Sahlin adapts the calculation of the hash value to a symmetric function with $h(m_1)/2 + h(m_2)/2$ [54].

Speed optimization

The speed of syncmers can be optimized by using prefix syncmers. A given submer is a prefix syncmer, if its prefix of length s' is a syncmer. The compression factor and the expected distance between prefix syncmers is identical to the underlying syncmer rule [40].

The speed up is a result of the fact that only a part of the submer needs to be considered and the syncmer rule can be chosen in the way that the number of possible syncmers is small enough to store them in a vector, so that they can be precomputed beforehand. Therefore, a syncmer can be determined by a look-up [40]. In that case, syncmers would be dependent on a preprocessing step though.

2.2.4 Universal hitting sets

A minimal universal hitting set is a minimal set of submers that guarantees that at least one of its members can be found in any sequence of length w. Therefore, a minimal universal hitting sets guarantees for any sequence that the representative submers have a distance from at most w, while ensuring a minimal set size for a given submer length [78–80]. The minimal universal hitting set belongs therefore in the category of preprocessing method, as the set needs to be determined beforehand for a given w and submer length.

While there has been no proof so far that finding an universal hitting set is NP-hard [78], currently only heuristic algorithms exist to determine minimal universal hitting sets [78–80].

2.2.5 Minimum decycling sets

The minimum decycling set is at its core an order for minimizers that should lead to a greater compression factor. The set is defined by considering all submers of length k_l and a directed graph that takes these submers as edges connecting nodes presenting $k_l - 1$-mers, if the outgoing node overlaps with the first $k_l - 1$-subsequence of the edge and the incoming node overlaps with the last $k_l - 1$-subsequence of the edge (this is called a complete De Bruijn graph of order k_l). From this graph a minimal set of submers is removed that leads to an acyclic graph. This removed set is then the minimal decycling set [80].

A representative submer is then determined by taking the minimal submer in a window that is included in the minimal decycling set as minimizer. If there is no submer in a window included in the minimal decycling set, then the classical minimizer approach is used for this window and the minimal submer of the window is taken. Pellow et al. showed experimentally that this approach leads to a lower density and therefore to a greater compression factor than a random minimizer order [80]. Therefore, the minimal decycling set has a window guarantee.

This approach can be a "on the fly" as the membership of an element in the minimal decycling set can be determined on the fly or a preprcessing method as the minimal decycling set can be determined beforehand for a given k_l. The advantage of the preprocessing method is that the membership lookups are quick and so this method is faster. However, as the set needs to be loaded in memory the preprocessing method will lead to a higher memory footprint. Therefore, the "on the fly" method on the other hand has a smaller memory footprint, but takes longer [80].

Furthermore, the minimum decylcing set as a preprocessing method in the implementation of Pellow et al. considers an exact lookup table [80]. But the set could also be stored in a probabilistic data structure as was done for the weighted minimizers [77] with a smaller memory footprint at the cost of adding false positive submers to the set.

Alternatively, similar to modmers, the window guarantee could be ignored and then only submers included in the set are considered representative [80].

2.2.6 Polar sets

Polar sets are a set of submers, where if a submer is included in the set, it is considered representative. The polar set guarantees that all representative submers are at least a given distance apart from each other for a given sequence. Unlike universal hitting sets, which work for all possible sequences, the polar sets are constructed for specific given sequences like a genome. However, despite being sequence specific, constructing a polar set guaranteeing the best spread of representative submers over

the given sequences is NP-hard and therefore heuristics are necessary for constructing a polar set [81].

2.2.7 Prefix words

Prefix words are sets of submers of length $s <= k_l$ and only submers whose prefix exist in this set are considered representative [82, 83]. Frith et al. first constructed these sets by minimizing the overlaps between submers [82]. Recently, Frith et al. proposed a set construction by optimizing the probability that any submer occurs in a given sequence length with the goal to find a good balance between the advantages of universal hitting sets and polar sets [83].

2.2.8 Sigmer

Sigmers need for their construction a transcriptome as an input, which is partitioned into groups by the user. One meaningful partition is the grouping of genes based on their location on the genome. Then, a sigmer is defined as a submer that only appears in one group but not the others and therefore is unique for this specific group [36].

2.2.9 Fleximer

Fleximers are sigmers of variable length. The length of a fleximer is kept between 25 *bp* and 80% of the given read length. Fleximers are constructed by creating a suffix tree on a given transcriptome. Like sigmers, this transcriptome needs to be divided into groups by the user. Then all prefixes that appear in only one group are considered and selected, if they meet the length requirement, are unique to a transcript and lead to a sufficient read coverage [38].

2.2.10 Comparison of different methods

The focus of this analysis will be on the "on the fly" methods modmers, minimizers and syncmers.

In order to enable a fair comparison between the different methods, the parameters were picked in such a way that the compression factor is the same. For minimizers five window sizes between 24 and 40 were chosen with k_l set to 20. The modulo for the modmers was set to the compression factor of the $(w, 20)$-minimizers. For the syncmers two linear syncmer types were tested, open syncmers where the position list only contains the position 0 and closed syncmers, where the position list contains the two values 0 and $k_l - s$. The s-mer size was determined for the first type by $s = k_l + 1 - cf$ and for the second type by $s = max(k_l + 1 - 2 \cdot cf, 1)$ with

cf being the compression factor of the minimizers. For the second syncmer type, s is defined to be at least 1, as the formula would otherwise lead to a negative value for s, therefore the compression factor for that syncmer type is 10 instead of 11 as it is for all other methods. Besides the speed analysis, all analyses were performed on canonical submers as the underlying submer method.

Speed

The speed of the different representative methods was compared similarly to the comparison between k-mers and strobemers with the minions option *minions speed* on the same data of $1,000$ sequences. All methods were built on top of ungapped non-canonical k-mers as that was the fastest submer method.

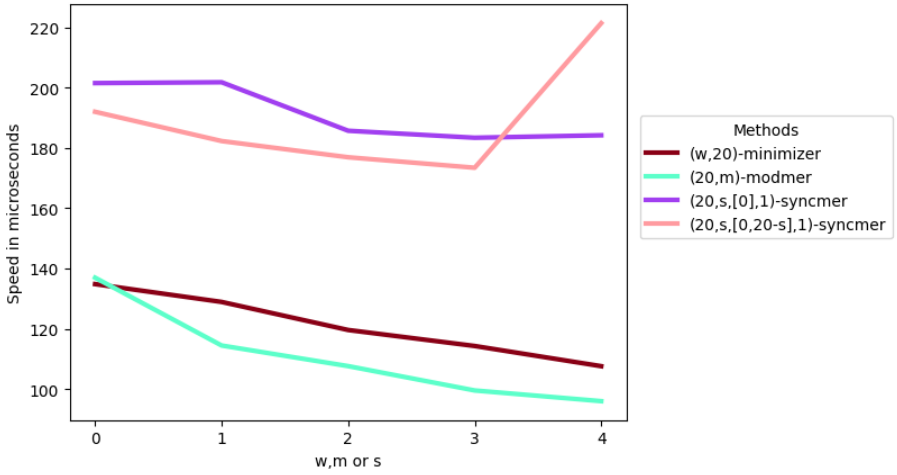


Figure 2.15: **Average speed of different representative methods.** The figure shows the speed in microseconds for the different representative methods built on top of k-mers with $k = 20$. The window size for minimizer, the modulo for modmers and the s size for syncmers changes for every value on the x-axis. The x-value indicates the position in the following lists: $[24, 28, 32, 36, 40]$ (minimizer), $[3, 5, 7, 9, 11]$ (modmers), $[18, 16, 14, 12, 10]$ ($(20, s, [0], 1)$-syncmers) and $[15, 11, 7, 3, 1]$ ($(20, s, [0, 20 - s], 1)$-syncmers).

Figure 2.15 shows that the modmer method is the fastest method with the minimizers being a close second. This is not surprising as the modmers are an even simpler method than the minimizer and only need to calculate the modulo value of a k-mer instead of comparing it to other k-mers.

The reason why the syncmers are so slow compared to the other two methods could be that it is costly to determine all the s-mers. However, similar to the minimizers, the smallest s-mer of a k-mer often can be determined by comparing it to the smallest s-mer in the previous k-mer and should therefore speed up the process. Interestingly, closed syncmers are a bit faster than open syncmers with the

small exception of closedsyncmers based on a s-mer size of 1, which led to a notable run time increase, probably because singular nucleotides are not that informative.

Furthermore, all methods become faster with an increase in the compression factor with the exception of the closedsyncmers with $s = 1$. This is probably due to two reasons. Firstly, with a greater compression factor more submers are considered not representative, so they can be discarded. Secondly, both minimizers and syncmers store k_1-mers or s-mers respectively, and thus only need to compare the next submer to the current minimum. The greater the compression factor, the more information is kept leading to a speed decrease.

Set size

The set size was determined with *minions counts* for the same parameters that were used in the speed analysis, which should ensure the same compression factor.

However, despite having the same compression factor, Figure 2.16 shows a notable difference in the set sizes of the different representative methods. While modmers and minimizers led to similar set sizes with modmers having a slightly smaller set size for decreasing compression factors, the syncmer methods led to a significantly higher set size for all compression factors. Open syncmers had thereby the highest set size with the exception of the last compression factor, where the closed syncmers take the lead. This increase for the last compression factor is similar to the outlier performance of the closed syncmers in Figure 2.15 and thereby proving further that $s = 1$ is not a good parameter choice.

The analysis was also done with strobemers as the underlying submer method. There is barely a difference between the different strobemer types. For strobemers of order 2, the minimizer set size is barely influenced through the change of submer method, while the set size of the modmers are closer to the ones of the minimizers for decreasing compression factors. The set size of the syncmer methods show an increase, but the order between them remains the same (see Figure 2.17).

For strobemers of order 3 as the underlying submer method, minimizers have the smallest set size for each strobemer type. Moreover, with strobemers of order 3 all representative methods led to a greater set size than with k-mers or strobemers of order 2 as the picked compression factors were greater (see Figure 2.18).

Therefore, the underlying submer method impacts the set size, but only impacts the comparison of the representative methods for order 3.

As already stated, the big differences between the different methods is quite surprising as the parameters were picked to guarantee the same compression factor. There are multiple possible explanations for this.

Firstly, the assumptions of the compression factor formula could be wrong. However, the assumption of independent and identically distributed submers is also used

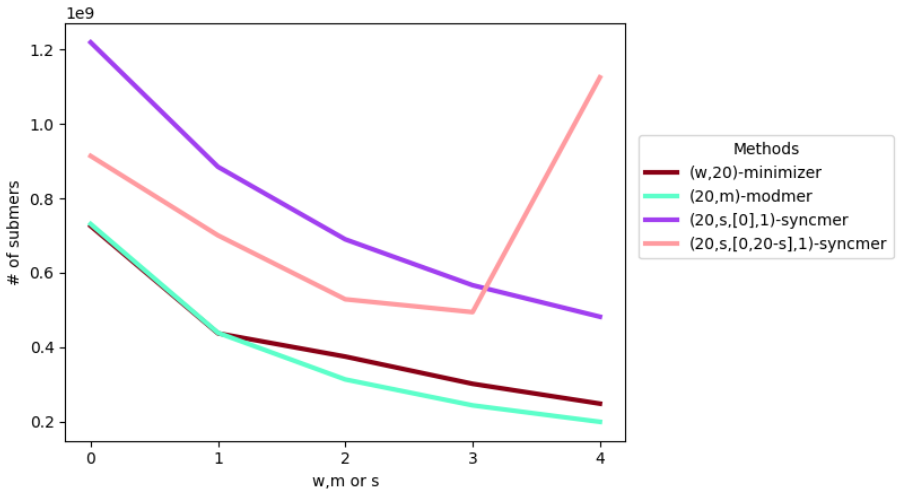
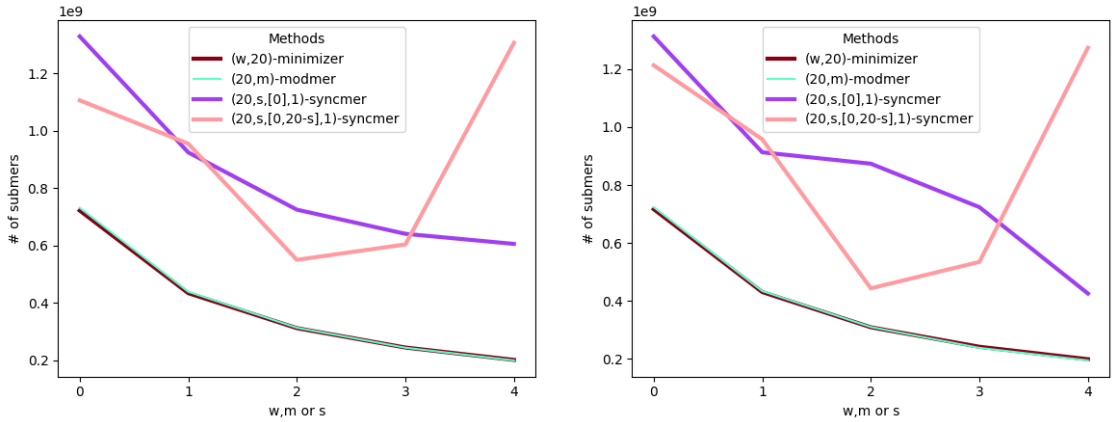


Figure 2.16: **Set size of different representative methods.** The figure shows the number of submers for the different representative methods built on top of k-mers with $k = 20$. The window size for minimizer, the modulo for modmers and the s size for syncmers changes for every value on the x-axis. The x-value indicates the position in the following lists: $[24, 28, 32, 36, 40]$ (minimizer), $[3, 5, 7, 9, 11]$ (modmers), $[18, 16, 14, 12, 10]$ ($(20, s, [0], 1)$-syncmers) and $[15, 11, 7, 3, 1]$ ($(20, s, [0, 20 - s], 1)$-syncmers).

in the compression factor formula for the minimizers, which seems to hold as the minimizer set size was not influenced by switching the underlying submer method from k-mers to strobemers.

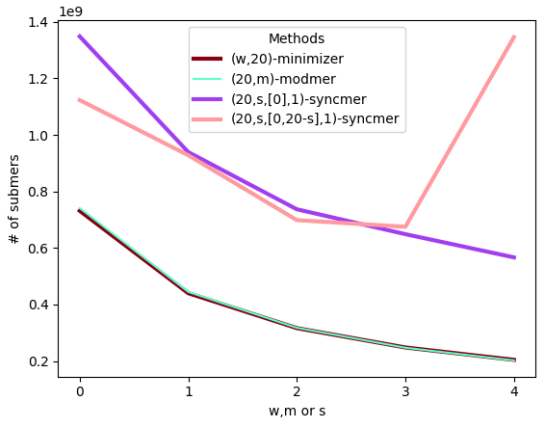
Alternatively, the compression factor might not be a good indicator for the set size. The set size is influenced by the number of different submers selected, while the compression factor is a measurement for how many submers were selected as representative submers. These two values should correlate, because the more submers are selected as representative, the higher are the chances that these selected submers differ from each other and the set size should increase. However, this correlation might only hold when comparing different compression factors of one representative method. Here, different methods are compared for the same compression factor and it seems like syncmers are more likely to pick different submers.

These explanations are not exclusive and both could be the reason for the unexpected results. In summary, the compression factor is not a good predictor of the set size for all methods and only modmers and minimizers are robust methods independent of the underlying submer method when comparing the set size.

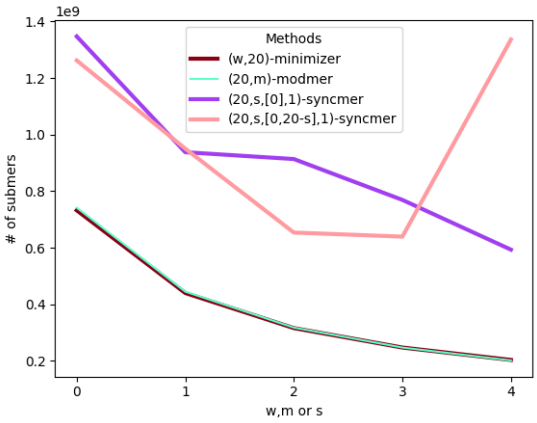


(a) Minstrobemers

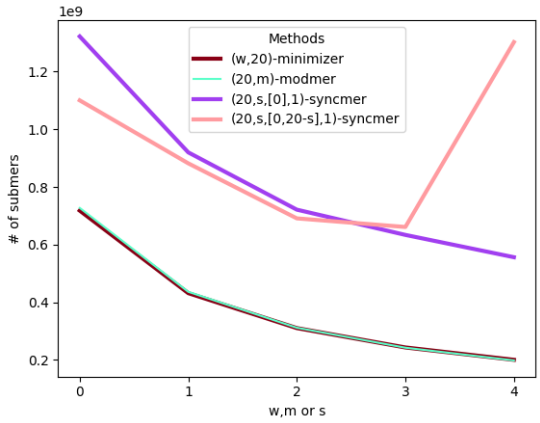
(b) Minstrobemers 8



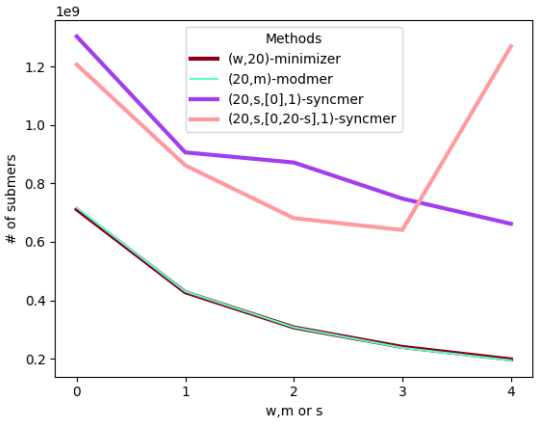
(c) Hybridstrobemers



(d) Hybridstrobemers 8

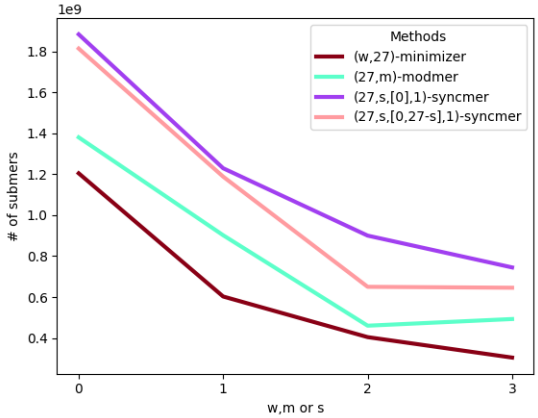


(e) Randstrobemers

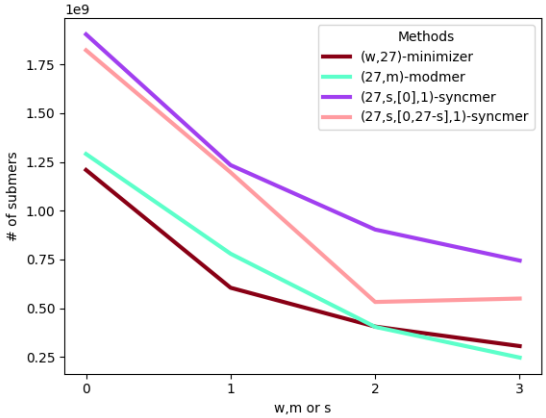


(f) Randstrobemers 8

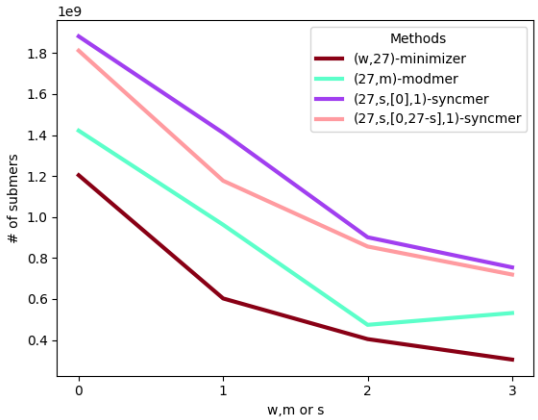
Figure 2.17: **Set size of different representative submer methods based on strobemers of order 2.** The figure shows the number of submers for the different representative methods built on top of different types of strobemers. The graphics on the left are built on top of $(2, 10, 10, 14)$-strobemers. The graphics on the right marked with a 8 behind the name are built on top of $(2, 10, 10, 18)$-strobemers. The window size for minimizer, the modulo for modmers and the s size for syncmers changes for every value on the x-axis. The x-value indicates the position in the following lists: $[24, 28, 32, 36, 40]$ (minimizer), $[3, 5, 7, 9, 11]$ (modmers), $[18, 16, 14, 12, 10]$ ($(20, s, [0], 1)$-syncmers) and $[15, 11, 7, 3, 1]$ ($(20, s, [0, 20 - s], 1)$-syncmers).



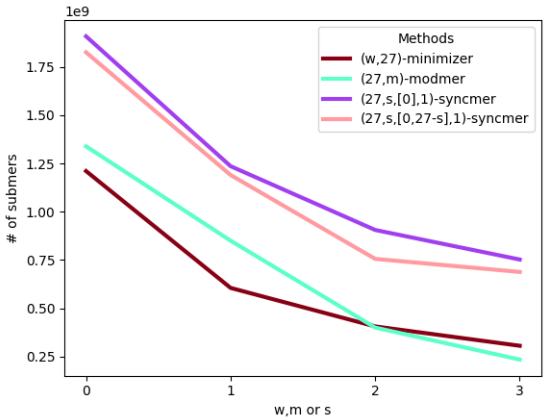
(a) Minstrobemers



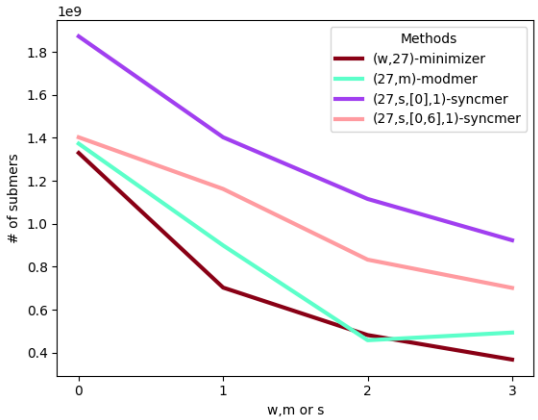
(b) Minstrobemers 8



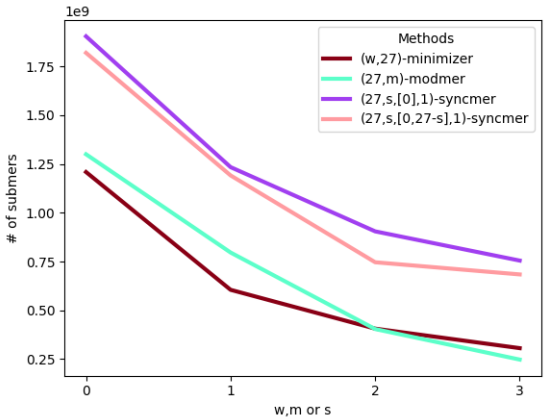
(c) Hybridstrobemers



(d) Hybridstrobemers 8



(e) Randstrobemers



(f) Randstrobemers 8

Figure 2.18: **Set size of different representative submer methods based on strobemers of order 3.** The figure shows the number of submers for the different representative methods built on top of different types of strobemers. The graphics on the left are built on top of $(3, 9, 9, 13)$-strobemers. The graphics on the right marked with a 8 behind the name are built on top of $(3, 9, 9, 17)$-strobemers. The window size for minimizer, the modulo for modmers and the s size for syncmers changes for every value on the x-axis. The x-value indicates the position in the following lists: $[29, 33, 37, 41]$ (minimizer), $[2, 4, 6, 8]$ (modmers), $[26, 24, 22, 20]$ $((20, s, [0], 1)$-syncmers) and $[24, 20, 16, 12]$ $((20, s, [0, 27 - s], 1)$-syncmers).

Uniqueness

The uniqueness was determined with *minions unique* for the same parameters as used in previous analyses for the human genome.

The percentage of unique representative submers did not vary much between the different parameters for one method, therefore, the results were summarized by calculating the average percentage for each representative method. As can be seen in Figure 2.19 (and in Figure A.7 for underlying strobemers of order 3), for k-mers as the underlying submer method minimizers have the highest percentage of unique representative submers, followed by modmers and then the syncmer methods. However, with strobemers as underlying method all representative methods lead to a similar percentage of unique representative submers, although the minimizer methods remains the best performing one.

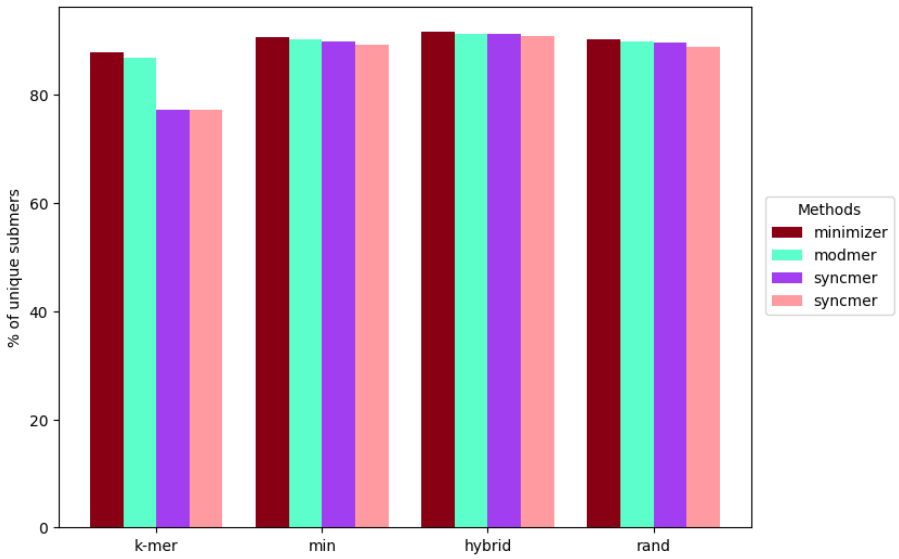
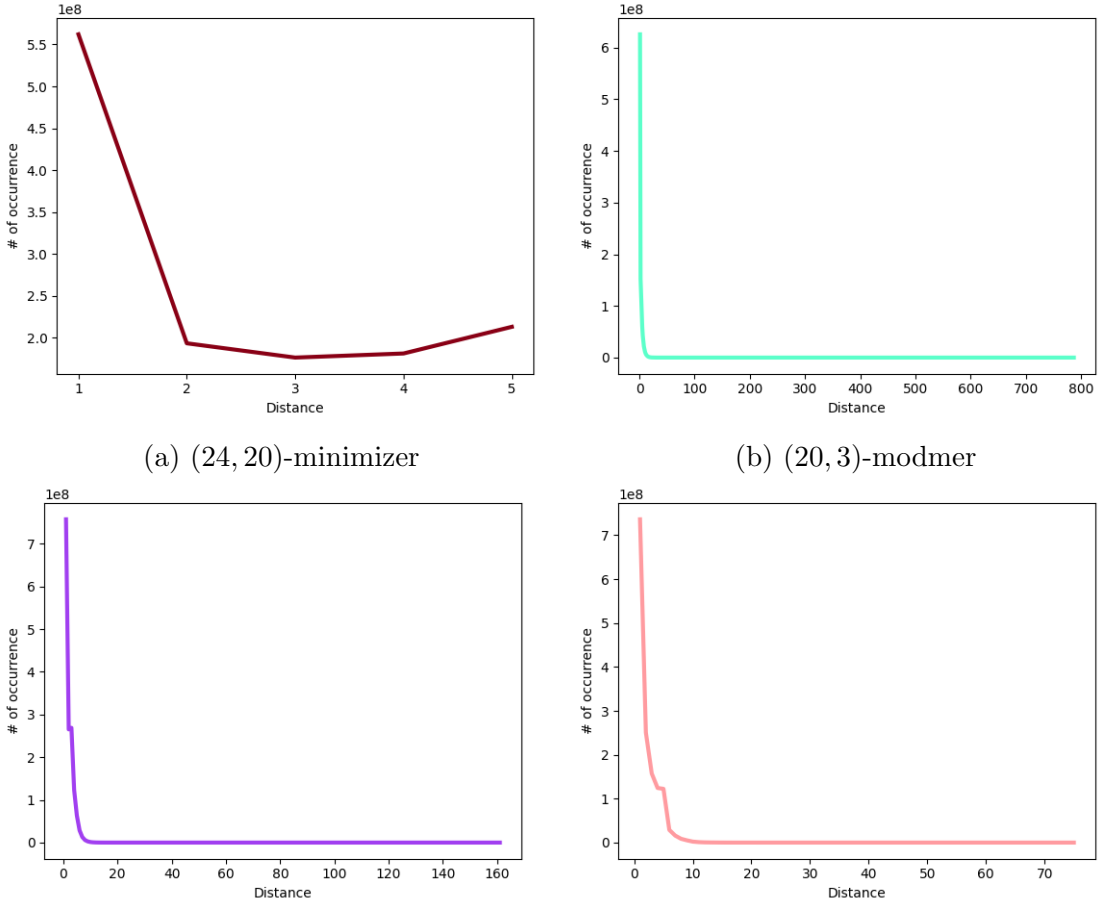


Figure 2.19: **Average percentage of unique submers for different representative methods.** The figure shows the average percentage of unique submers for all representative methods for k-mers and the different strobemers as underlying method. The average is calculated over the different parameters for the window size for minimizer, the modulo for modmers and the s size for syncmers. The following parameters were used: $[24, 28, 32, 36, 40]$ $((w, 20)$-minimizer), $[3, 5, 7, 9, 11]$ $((20, m)$-modmers), $[18, 16, 14, 12, 10]$ $((20, s, [0], 1)$-syncmers) and $[15, 11, 7, 3, 1]$ $((20, s, [0, 20 - s], 1)$-syncmers). As underlying submer method, 20-mer were used for the k-mers and for the different strobemers $(2, 10, 10, 14)$-strobemers were used.

These results align with the findings of Shaw et al., who proved that context free methods like modmers and syncmers have a higher expected number of repetitive k-mers [44] leading to a smaller percentage of unique submers.

Distance

The distance distribution between representative submers was determined with *minions unique* for the same parameters as used in previous analysis for the human genome.



(a) $(24, 20)$-minimizer

(b) $(20, 3)$-modmer

(c) $(20, 18, [0], 1)$-syncmer

(d) $(20, 15, [0, 5], 1)$-syncmer

Figure 2.20: **Distance between representative submers of different representative methods.** The figure shows the occurrence of different distances between adjacent representative submers on the human genome for $(24, 20)$-minimizers, $(20, 3)$-modmers, $(20, 18, [0], 1)$-syncmers and $(20, 15, [0, 5], 1)$-syncmers based on k-mers.

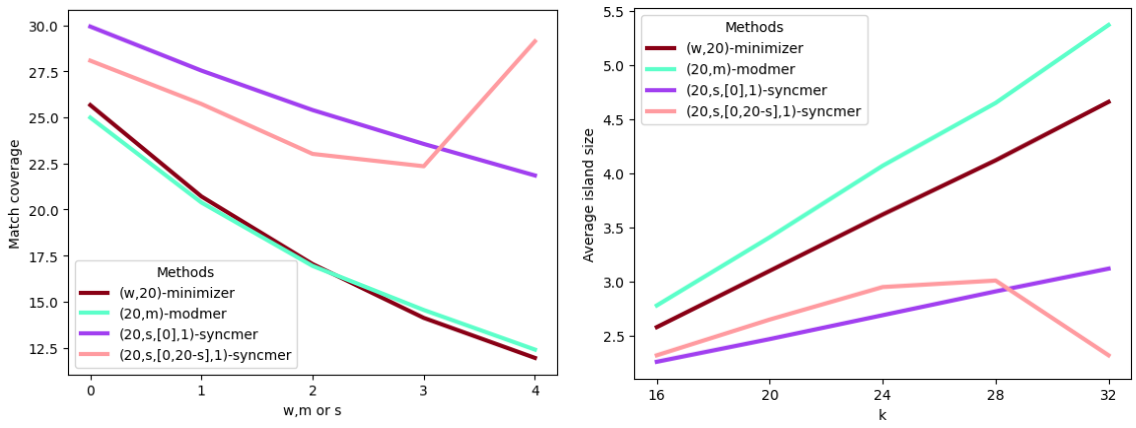
As shown in Figure 2.20 (and in Figure A.8 for more parameters and in Figure A.9 for strobemers as underlying submer method), the greater the distance between the adjacent representative submers, the less likely it is to occur. Modmers have the greatest distance between two adjacent representative submers, open syncmers come in second and closed syncmers third. Unsurprisingly, minimizers span over only a small range of distances due to the window guarantee.

However, as described above closed syncmers have a window guarantee and should have a maximal distance of $k_l - s$, but the closed syncmers exceed this. The

reason for this is that the window guarantee is based on non-canonical k-mers and does not hold for canonical k-mers as it can happen that the k-mer that would guarantee the maximal distance is not the minimal k-mer and therefore not canonical. Consequently, a method to enable the window guarantee is to apply the syncmers for canonical k-mers in the following way: If either a k-mer or its reverse complement is a syncmer, the canonical k-mer is considered a syncmer. But this approach would then diverge from the other representative methods, which apply only for the minimal part of the k-mer pair. Moreover, the window guarantee would still not hold for strobemers as underlying submer method as the proof relies on the smallest s-mer being guaranteed to be part of the resulting submers, which is not necessarily given as the strobemer method could pick a strobe without including the smallest s-mer. Therefore, the window guarantee of syncmers is limited to non-canonical k-mers, while the window guarantee of minimizers is independent of the underlying submer method.

Accuracy

The match coverage and average island size was determined with *minions accuracy* for the same simulated $100,000$ reads of length 100 *bp* as in the comparison between k-mers and strobemers.



(a) Match Coverage

(b) Island Size

Figure 2.21: **Match coverage and island size of different representative submer methods with substitution rate 10.** 4 and 8 k-mer refer to the gapped $k+4$-mer or $k+8$-mer method with 4 or 8 gaps. The strobemers have an order of 2, so the singular strobes have a length of $k/2$ and they have a window length of $k/2+4$, so that a maximal gap of length 4 to the previous strobe is ensured.

As can be seen in Figure 2.21, the match coverages were highest for the open syncmers, followed by the closed syncmers and then the minimizers and modmers, which had quite similar coverages. The average island sizes are in reverse order of this, which is the expected result.

Moreover, the match coverage of the representative methods based on strobemers was smaller than the ones based on k-mers for all methods (see Figure 2.23 and Figure A.10 for strobemers with a maximal gap size of 8). This is unexpected as the strobemer methods compared to the ungapped k-mers had a higher match coverage, which apparently did not improve the performance of the representative methods.

Additionally, the match coverages were smaller for every method than the ones from the underlying submer method, which was to be expected as only a subset of the underlying submers are considered.

In general, these results confirm Shaw et al.'s analysis that context dependency leads to a lower match coverage [44], at least when comparing minimizers to syncmers. However, modmers as the most simplistic context free method did not yield a higher match coverage.

However, these results have to be seen with a grain of salt, as the set sizes of the syncmer methods are greater than the ones from the minimizers and modmers, meaning, they store more submers, which could also explain the higher match coverages. In order to investigate this further, the match coverage was divided by the number of matches a method had to determine the match coverage per match. Figure 2.22 shows a similar graph as the match coverage. The most notable difference is that the modmers have a greater match coverage per match than minimizers and have a similar coverage to the closed syncmers. This indicates that minimizer matches cluster together and overlap thereby covering the same sequence multiple times, which might be partly due to their distance distribution. Therefore, the higher match coverage of the syncmers in Figure 2.21 seems to not be the result of the greater set sizes.

Furthermore, the k-mer lemma could be applied to the complete window size of the minimizers to determine a threshold guaranteeing zero false negatives when searching for a sequence with mutations or sequencing errors, but this would lead to too small a threshold and quite quickly lead to an unusable threshold of zero for an increasing number of mutations or sequencing errors. For methods without a window guarantee, the k-mer lemma is not even applicable. Therefore, like in the comparison of k-mers and strobemers, the highest threshold for finding zero false negatives in a set of simulated sequencing files with 2 to 5 errors was determined practically with *minions accuracy*. As the match coverage was the best for the representative methods based on k-mers, this analysis was only applied to those.

For already 4 errors, no representative method led to zero false negatives and therefore Table 2.3 shows only the result for 2 and 3 errors. Modmers lead to the worst results, having a significantly higher number of false positives than all the other method for 2 errors and not even one threshold with zero false negatives for 3 errors. For 2 errors, the minimizers and both syncmer methods lead to similar

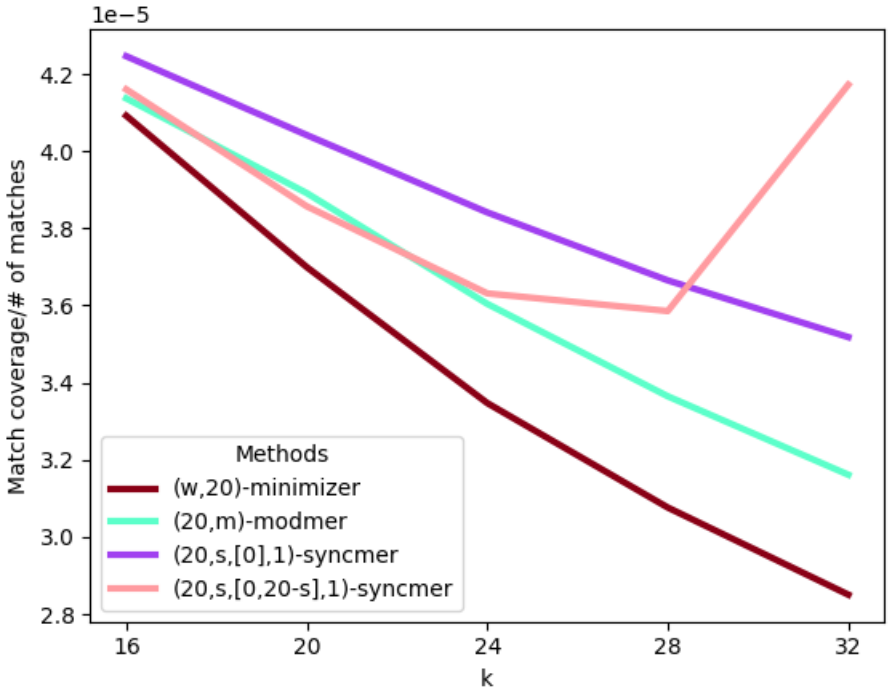


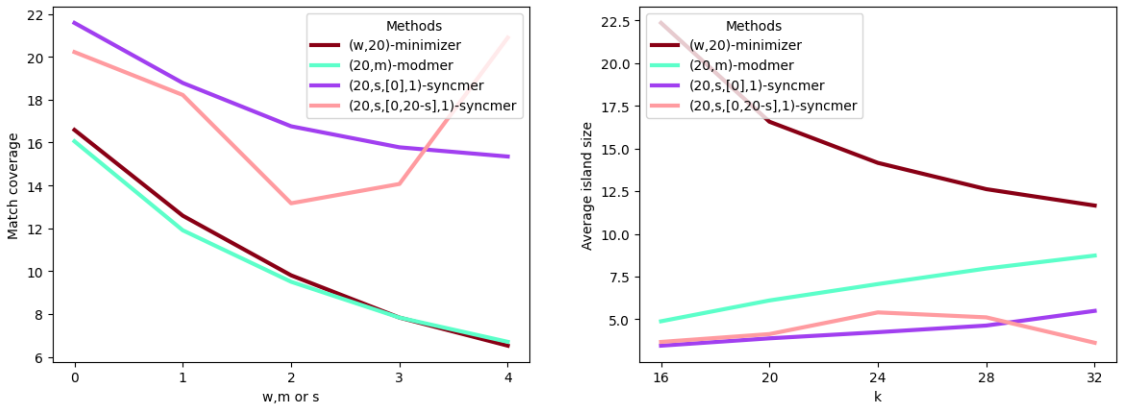
Figure 2.22: **Match coverage divided by the number of matches of different representative submer methods with substitution rate 10.** 4 and 8 k-mer refer to the gapped $k+4$-mer or $k+8$-mer method with 4 or 8 gaps. The strobemers have an order of 2, so the singular strobes have a length of $k/2$ and they have a window length of $k/2+4$, so that a maximal gap of length 4 to the previous strobe is ensured.

	2 Errors		3 Errors	
Method	FP	OT	FP	OT
$(24, 20)$-minimizer	**504**	0.28	876,102	0.08
$(3, 20)$-modmer	354,735	0.09	-	-
$(20, 18, [0], 1)$-syncmer	505	0.24	912,706	0.07
$(20, 15, [0, 5], 1)$-syncmer	**504**	0.38	**551,455**	0.08

Table 2.3: **Number of false positives and optimal threshold for different representative submer methods.** The table shows the number of false positives (FP) for the greatest threshold that still lead to zero false negatives (OT for optimal threshold). Three data sets with 2 and 3 errors were evaluated. The smallest number of false positives per column was marked in bold. If no threshold led to zero false negatives, the FP and OT were marked with a '-'.

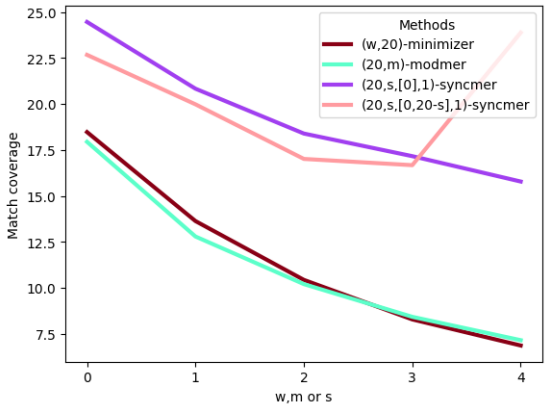
results, while for 3 errors closed syncmers yield the smallest number of false positives. Therefore, it is depending on the number of expected errors, if the threshold analysis should impact the choice of representative submer method.

Comparing the threshold results with the results of the k-mers in Table 2.1, the number of false positives is higher for the representative methods and the thresholds smaller. Therefore, the match coverage analysis as well as the threshold anal-

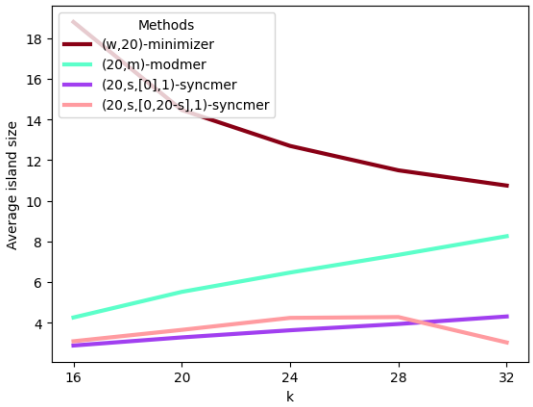


(a) Match Coverage, Minstrobemers

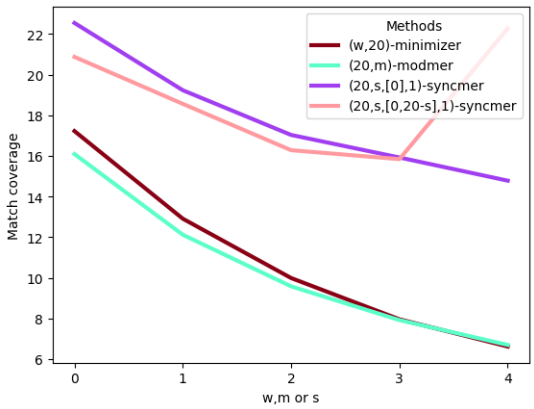
(b) Island Size, Minstrobemers



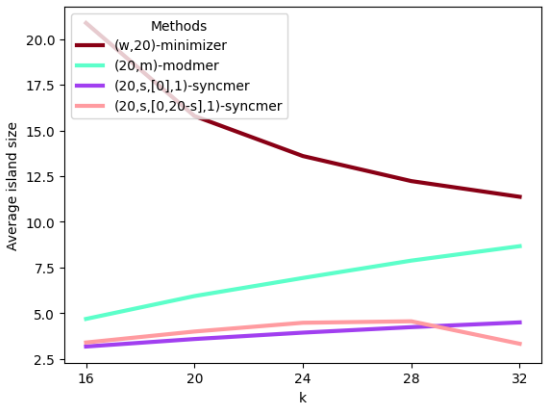
(c) Match Coverage, Hybridstrobemers



(d) Island Size, Hybridstrobemers



(e) Match Coverage, Randstrobemers



(f) Island Size, Randstrobemers

Figure 2.23: **Match coverage and island size of different representative submer methods based on strobemers of order 2.** The figure shows the match coverage and island size of representative submers based on $(2, 10, 10, 14)$-strobemers. The window size for minimizer, the modulo for modmers and the s size for syncmers changes for every value on the x-axis. The x-value indicates the position in the following lists: $[24, 28, 32, 36, 40]$ (minimizer), $[3, 5, 7, 9, 11]$ (modmers), $[18, 16, 14, 12, 10]$ $((20, s, [0], 1)$-syncmers) and $[15, 11, 7, 3, 1]$ $((20, s, [0, 20 - s], 1)$-syncmers).

ysis confirm that the representative methods reduce the accuracy of the underlying

methods.

2.2.11 Conclusion of the comparison

The comparison of the representative methods is hard as their performance is dependent on their parameters and it is not a trivial task to determine comparable parameters. Here, all analyses are based on the same compression factor. However, as is shown above the set size of the different methods differ significantly and therefore do not compress the data similarly making the interpretation of all analyses harder. Therefore, a better method to predict the set size of syncmers would be beneficial.

Moreover, the results are also impacted by the underlying submer method, which sometimes leads to different results for the set size or the number of unique submers for the syncmer methods, while the minimizers have been shown to be quite robust to the underlying method. Considering the fact that the match coverage based on strobemers is lower for all representative methods than the ones based on k-mers and that the k-mer method is the fastest, *Needle count* supports k-mers instead of strobemers.

When comparing the representative submer methods, two characteristics are often discussed: context dependency and the window guarantee.

Context dependency leads to a smaller match coverage as it is influenced by mutations or sequencing errors outside of the chosen submer sequence itself [44].

However, as was shown above the context free methods do not have a window guarantee and therefore greater distances between adjacent submers within one sequence and a greater average island size when comparing two sequences. Edgar questions if the window guarantee is a meaningful metric due to the lower match coverage [40]. While this is a compelling argument, the problem only arises in light of a high mutation rate or a high number of sequencing errors as the difference in the match coverage is only this prominent, if either is high. Therefore, in theses cases where the match coverage is rather similar, the window guarantee might be of interest again as it ensures that every part of the sequence is covered.

For *Needle count*, mutations can be accounted for as they are known beforehand and therefore, the only question is how impactful sequencing errors are. In *Needle count*, the occurrence of each (representative) submer is counted and an expression value is estimated via the median of all occurrences of submers in a queried sequence. As the median is robust to outliers and is rather unlikely that the sequencing error appears always at the same position, the impact of sequencing errors seems rather small. Moreover, with the advances in sequencing technology [68], the number of expected errors is also decreasing. Therefore, for *Needle count*, the match coverage

should be accounted for but only as one metric among others. Because the minimizers are fast with a small set size and a high percentage of unique submers and a window guarantee, *Needle count* supports minimizers instead of syncmers.

Furthermore, it should be mentioned that the analysis here is not complete, for example no method requiring a preprocessing step was considered and further investigations are needed. Preprocessing methods could especially become more of interest, as the completeness of the human transcriptome is only a matter of time [36] and once achieved the preprocessing does not even need to be updated anymore.

In general, determining a good (representative) submer method depends on the task at hand, but is also an active field of research. For this reason having an application like *minion*, which offers analysis of these methods makes comparison easier and could even help by guiding a researcher in the question, which method to use for a given task.

2.3 Transcript quantification

In 2014, *Sailfish* [35] introduced the first alignment free method for transcript quantification [34]. The idea of *Sailfish* and all other methods that followed is to determine the most likely origin of a read instead of doing an base-by-base alignment between the transcriptome and the reads [35–39].

An even simpler approach is presented here with *Needle count*, which does not try to determine the origin of a read but rather assumes that most representative submers of a transcript are unique and uses the count of these submers for quantification. While this assumption does not always hold, especially for transcripts of isoforms, which often share a larger part of their sequence, the approach can lead to good results in cases, where it holds.

2.3.1 Overview of existing transcript quantification application

An overview of existing transcript quantification methods is given here.

Sailfish

Sailfish consists of two operations. The first operation is the creation of an index based on a given transcriptome. This index contains all k-mers in the transcriptome, their counts and the relationship between k-mers and transcripts. The second operation is then the actual transcript quantification, where the k-mers in a RNA-seq file are counted and based on this information an expectation–maximization (EM) algorithm determines the quantification for each transcript [35].

RNA-Skim and Fleximer

Shortly after *Sailfish* and inspired by it, *RNA-Skim* was introduced [36]. The idea of *RNA-Skim* was then taken even further by the application *Fleximer* [38]. *RNA-Skim* and *Fleximer* work similarly to *Sailfish*, but instead of k-mers, they use sigmers or fleximers respectively [36, 38].

kallisto

Kallisto introduced pseudoalignment [37] to quantify transcripts. The index *kallisto* builds over a given transcriptome in form of a coloured de Bruijn graph, where each color symbolizes a different transcript in the transcriptome. Therefore, the *kallsito* index stores additional information for the quantification via an EM algorithm compared to *Sailfish* as the order of the k-mers in the transcriptome is kept. According to the *kallsito* authors this leads to a higher accuracy [37].

The good performance in speed is also a result from the fact that *kallisto* makes use of redundant information. This is achieved by skipping over k-mers in non-branching paths of the coloured de Bruijn graph. The skip is then validated by confirming that the k-mer after the skip is present in the same transcripts as the k-mer before the skip, if not, every k-mer is considered [37].

Salmon

The motivation behind *Salmon* [39] was to increase the accuracy of transcript quantification by correcting GC-content biases. Before these corrections are applied, *Salmon* builds an index based on a given transcriptome and quantifies transcripts of a RNA-seq file by using a method called quasi-mapping [84] to determine the most likely origin of a read in a RNA-seq file [39] (for more information about the quasi-mapping and a direct comparison to *kallisto*'s pseudoalignment, see [84, 85]).

2.3.2 Needle count

Needle count follows an even simpler approach as the methods mentioned above, instead of trying to determine the most likely origin of a read, *Needle count* takes the median of a transcript representative submer counts. This changes the typical metrics to quantify transcript expression, as the other methods try to estimate the number of reads matching with a transcript. Therefore, *Needle count* leads to its own expression value, which is from here on called ME (for median expressed).

The median was chosen instead of other metrics like the mean as the median handles outliers better. This is important as non-unique submers and sequencing errors can lead to outlier count values. If one is interested in the expression of a

singular gene on the other hand, the mean of ME's is taken into account, as the singular transcripts should have similar MEs.

One downside of the ME is that differentiating between transcripts, which share half or more of their submers is not possible because the median will be the same. This is often the case for isoforms.

Similar to the methods mentioned above, *Needle count* consists of two steps: an indexing step and a quantification step.

The index is build by determining all representative submers in a given genome or transcriptome and storing them in a hash table. *Needle count* uses minimizers as representative submers.

For the quantification step, *Needle count* expects as input one or more RNA-Seq files, one genome or transcriptome file and the associated genome or transcriptome index. *Needle count* then counts the occurrences of every minimizer that can be found in the index of a given RNA-seq file and then quantifies a transcript by taking the median of its minimizers.

In theory, any submer method would work with *Needle count*, but at the moment, it only supports gapped and ungapped k-mers and minimizers.

2.3.3 Comparison of transcript quantification applications

The state of the art applications *kallisto* and *Salmon* are compared to *Needle count* in regards to their accuracy and their speed and space consumption. Moreover, an actual differentially expression analysis is repeated with the quantification of *Needle count* to show its applicability in real use cases.

Accuracy

Two different data sets, one simulated and one real data set were used to determine the accuracy of the different applications.

Simulated data The same simulated data set as created by Darvish et al. [32] was used. The data was constructed with the simulation tool Polyester [86]. 256 experiments were simulated with the coverage $20, 40, 60$ and 80, so 64 experiments per coverage. For each of these experiments a second experiment was simulated where ten % of the genes are differentially expressed with fold changes of $\frac{1}{4}, \frac{1}{2}, 2$ and 4 with equal probability. All experiments contain 75-bp paired-end reads and the simulations are based on 100 randomly picked protein-coding transcripts of the human genome [32].

As Figure 2.25 shows *Needle count* performs similar as *kallisto* and *Salmon* for all three types of tested minimizers. The mean and the variance of *Needle count* is

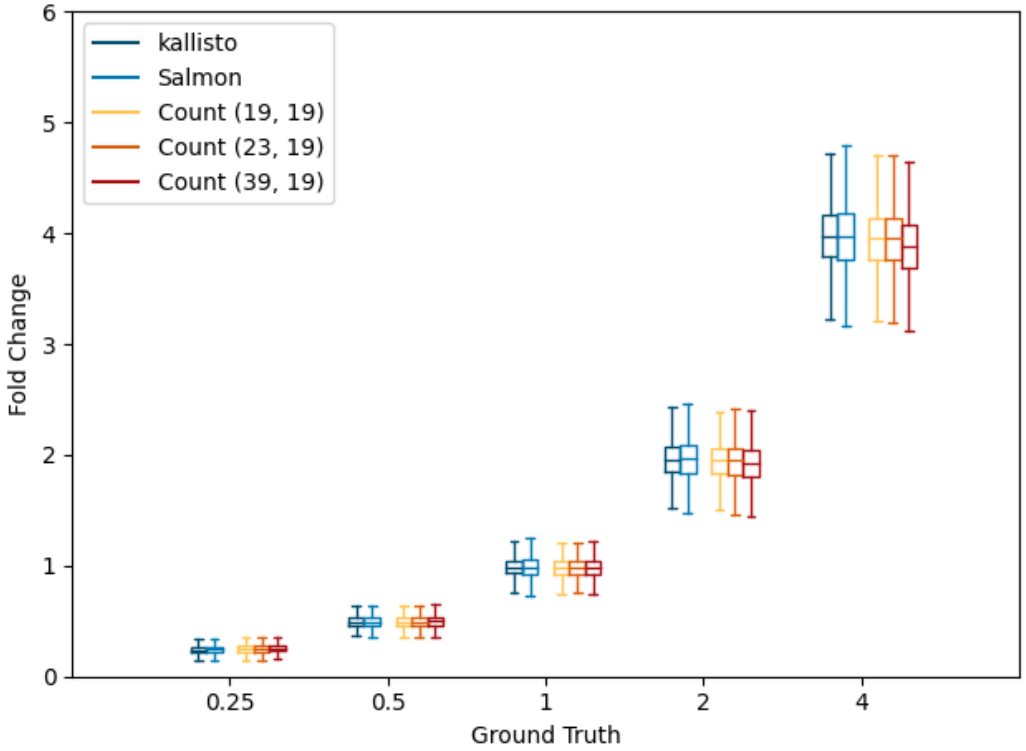


Figure 2.24: **Comparison of Needle count with kallisto and Salmon on differentially expressed genes.** The figure shows the fold change of differentially expressed genes in the simulated data set. For *kallisto* and *Salmon* k was set to 19. *Needle count*, here named just Count, uses minimizers with the window sizes $19, 23$ and 39. The x-axis presents the actual fold change that was set during the simulation, the y-axis shows the measured fold change.

in the same range as *kallisto*'s and *Salmon*'s means and variances. For $(19, 19)$- and $(23, 19)$-minimizers, there is barely any difference visible, while $(39, 19)$-minimizers show especially for the fold changes above 1 a smaller mean. A slightly less accurate performance for such a great window size was to be expected. However, $(39, 19)$-minimizers still perform well enough to be actually used.

A similar performance can be seen for the coverage analysis in Figure 2.25. Moreover, in Figure 2.25 it is more noticeable that *Salmon* has a greater variance than *kallisto* and often also than the *Needle count* applications. Therefore, *Needle count* shows a better performance than *Salmon*, which is surprising as *Salmon* uses a more sophisticated method. However, *Salmon* was built to account for GC-content biases, which were not simulated, so the worse performance could be due to that.

Real data The real data set is from the Sequencing Quality Control (SEQC) project [87] and consists of four samples. Sample A contains Universal Human Reference RNA (UHRR) and Sample B Human Brain Referece RNA (HBRR). Samples

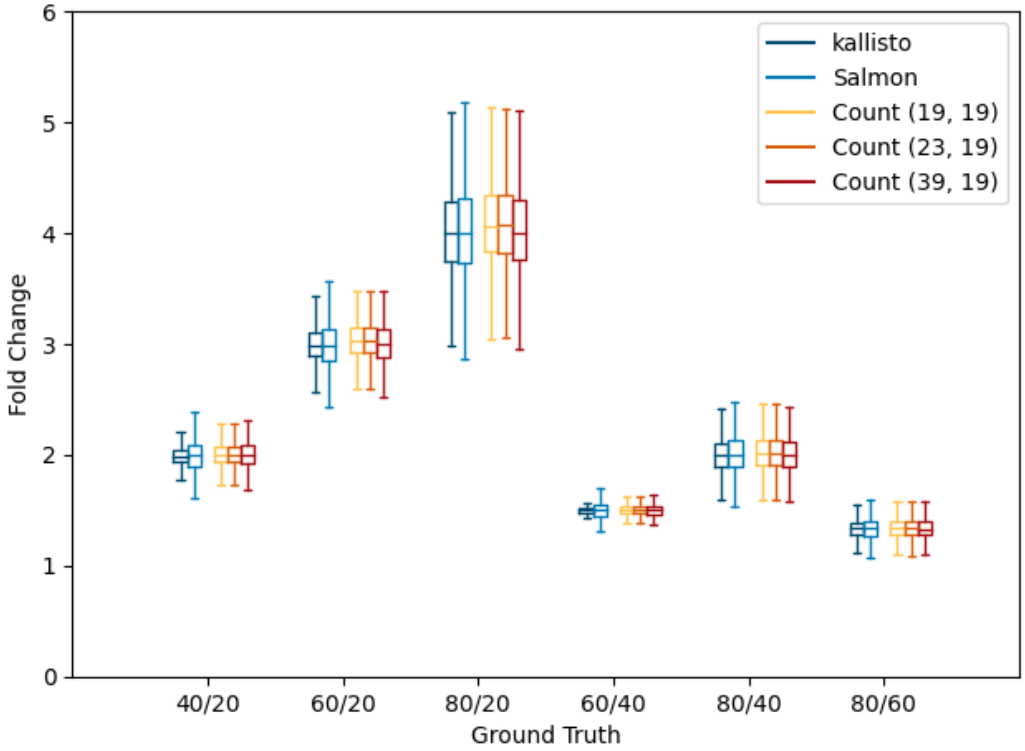


Figure 2.25: **Comparison of Needle count with kallisto and Salmon with different coverages.** The figure shows the fold change of genes expressed within simulated sequencing experiments of different coverages. For *kallisto* and *Salmon* k was set to 19. *Needle count*, here named just Count, uses minimizers with the window sizes $19, 23$ and 39. The x-axis presents the actual expected fold change by dividing the known coverages, for example comparing coverage 40 with 20 should lead to a fold change of $40/20 = 2$. The y-axis shows the measured fold change.

C and D are a mixture of A and B in the ratio 3/1 for C and 1/3 for D. The gene expression of 818 genes was determined by real-time polymerase chain reaction (RT-PCR) and for $15,984$ genes the expression was determined by microarray analysis [32, 87].

This data was analyzed by calculating the Spearman correlation of the number of the TPM values as given by *kallisto* and *Salmon* and the ME as given by *Needle count* with the actual gene expression by either the RT-PCR or the microarray analysis. The TPM value is a normalization method based on the number of matched reads of a transcript and is calculated in the following way for one transcript i with n_i being the number of matched reads to transcript i and l_i being the length of transcript i:

$$TPM_i = \frac{\frac{n_i}{l_i}}{\sum_j \frac{n_j}{l_j}} \cdot 1,000,000. \tag{2.2}$$

The TPM values account for different transcript lengths between different tran-

scripts and different sequencing depths between sequencing experiments [27, 88].

Furthermore, the mean squared error was determined by considering the expected fold change of the ratio between C and D based on A and B and the actual fold change based on C and D. The expected fold change was calculated by the following formula:

$$\frac{D}{C} = \frac{A_i + 3B_i}{3A_i + B_i}, \tag{2.3}$$

where A_i and B_i represent the expression values of a transcript i for experiment A and B [32].

For the mean squared error analysis the *Needle count* values were normalized in the following way: All ME values were summed up and divided by 1 million to obtain the correction factor, then all ME values were corrected by division of the correction factor. This normalization was inspired by the TPM normalization. Unlike the TPM normalization, the ME values do not need to account for the gene length as the median is less prune to be influenced by this. Therefore using the unnormalized or the normalized MEs for the correlation analysis had no impact on the results for *Needle count*.

	SEQC	Microarray	MSE
kallisto	80.2	76.3	0.6
Salmon	80.7	76.7	0.6
Count $(19, 19)$	80.6	77.3	0.6
Count $(23, 19)$	80.6	77.2	0.5
Count $(39, 19)$	80.6	77.2	0.5

Table 2.4: **Comparison of Needle count with kallisto and Salmon on the SEQC data set.** For *kallisto* and *Salmon* k was set to 19. *Needle count*, here named just Count, uses minimizers with the window sizes $19, 23$ and 39. The columns SEQC and mircoarray give the Spearman correlation in percent to the RT-PCR quantification and to the miroarray quantification respectively. The column MSE represent the mean square error of the titration monotonicity transcript-wise.

As Table 2.4 shows *Needle count* performs similarly or even better than the state of the art tools *kallisto* and *Salmon*. However, the differences are so small that it can be concluded that all methods perform the same.

Interestingly, there is barely any difference between the different window sizes, even for $(39, 19)$-minimizers, which have performed slightly worse on the simulated data set than the other tested minimizers. Proving even further that the usage of greater window sizes can lead to accurate results.

Speed and space

For the speed and space analysis, the real data set from the accuracy analysis was used and every proteincoding transcript of the human genome was quantified. As all applications first create an index and then quantify, the speed and space needs to be determined for both steps. All comparisons were performed on a Linux machine (Debian GNU/Linux 11 (bullseye)) with 1TB RAM and an IAMD EPYC 7702P 64-Core Processor CPU with 64 cores and 256MB L3 cache.

	Index			Quantification	
	Time	RAM	Index Size	Time	RAM
kallisto	179	5.0	1.9	259	3.9
Salmon	187	0.6	0.5	993	0.9
Count $(19, 19)$	24	1.7	0.6	631	7.6
Count $(23, 19)$	15	0.5	0.2	313	1.3
Count $(39, 19)$	7	0.2	0.1	202	0.5

Table 2.5: **Speed and space comparison of Needle count with kallisto and Salmon.** Needle count, here named just Count, uses minimizers with the window sizes $19, 23$ and 39. The time is given in seconds and the RAM and index size in GB. The time and memory consumption for the quantification was averaged over all sequencing experiments.

As can be seen in Table 2.5, *Needle count* with $(39, 19)$-minimizers is the fastest and the most space efficient method, which is not surprising as it considers less submers than any other method. For the index building step *Needle count* is considerably faster with any minimizer than *kallisto* and *Salmon*, but has only a smaller memory footprint and a smaller index than *Salmon* with $(23, 19)$- and $(39, 19)$-minimizers. Although the index size of *Needle count* with $(19, 19)$-minimizers is comparable with only a 0.1 GB difference. In terms of the index build, *kallisto* has the biggest memory footprint and the largest index size and is only marginally faster than *Salmon*.

Surprisingly, *kallisto*'s quantification is faster than *Needle count*'s quantification for $(19, 19)$- and $(23, 19)$-minimizers despite the fact that *kallisto* uses an EM-algorithm, while *Needle count* just determines the median. The above explained approach of *kallisto* that some k-mers can be skipped is probably the cause for their good performance. An approach that is dependent on storing a data structure that keeps information about the relationship between submers and therefore is not applicable for the simplistic approach *Needle count* follows. However, as minimizers with greater window sizes can outperform *kallisto*, while still maintaining a good accuracy, *Needle count* is competitive.

Differentially expression analysis

A data set obtained by Hong et al. [89] consisting of ten RNA-Seq files from breast cancer tissue (luminal B subtype) and three files of normal healthy tissue from the same woman were analyzed. One of the breast cancer samples showed a low quality in previous studies and therefore was disregarded [89].

The expression values of all human gene transcripts were determined with *Needle count* for all twelve samples for $(19, 19)$-minimizers. Afterwards, the expression value of every protein coding gene was determined by taking the mean of the expression value of its transcripts. These values were then analyzed with the differential analysis tool *DESeq2*.

Hong et al. pointed out that the data set has 67 differentially expressed genes, which are associated with the lymphocyte activation pathway and the PPAR signaling pathway. The genes of the lymphocyte pathway are upregulated in the breast cancer samples, while the genes in the PPAR signaling pathway are downregulated [89]. As can be seen in Figure 2.26, the same association can be found with *Needle count* and the *DESeq2* analysis considers all 67 genes significantly differentially expressed.

Note that in Figure 2.26, the breast cancer sample named "$C0$" seems like an outlier as its expression profile looks more similar to the healthy samples. However, this can be also seen in the analysis from Hong et al. [89].

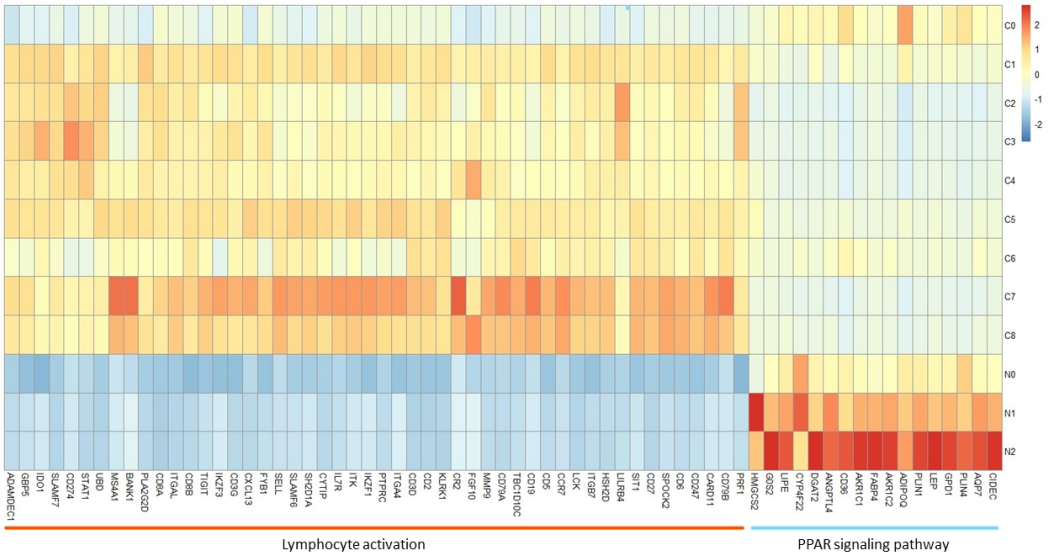


Figure 2.26: **Heatmap of the 67 differentially expressed genes.** The samples starting with the letter "C" indicate a cancer sample, the samples starting with the letter "N" indicate a healthy sample. The heatmap was scaled columnwise.

Furthermore, a breast cancer signature found in a different study [90] on a different data set last year was tested as well. This signature consists of five genes, two

were upregulated in breast cancer (KIF4A and COL11A1) and three were downregulated (SFRP1, SAA1 and RBP4) [90]. All five genes were significantly differentially expressed according to DESeq2 based on the *Needle counts*, confirming the gene signature and the applicability of *Needle count*.

The results for both analysis remained the same using $(23, 19)$- and $(39, 19)$-minimizers.

Conclusion

The here presented approach *Needle count* works well in practice, despite the fact that the order of the submers is not kept and the median approach is more simplistic than maximizing the likelihood a read originated from a specific transcript via an EM algorithm. However, this approach has its limitations, as too similar transcripts can not be differentiated.

Furthermore, the usefulness of representative submers like minimizers was shown here for *Needle count*, which could be integrated in the other approaches as well to boost their memory and run time performance.

Needle count could have a greater flexibility by applying the idea from [91] and building the index over k-mers, thereby allowing queries for (w, k)-minimizers for any $w > k$. In theory, the queries could also support syncmers or any other representative submer method based on k-mers. Such an approach might speed up the query as only the representative k-mers are considered and prevents rebuilding the index, if a different representative submer method is desired. As the index for most sizes of k for one genome is affordable, this seems like a good practice.

Overall, *Needle count* proves that even a simple quantification method can lead to good results and therefore is worth investigating, especially in order to boost speed and space performances.

Chapter 3

Indexing big data

Nowadays, the sequencing databases such as the European Nucleotide Archive [92] or the Sequence Read Archive [5] contain sequences in the range of petabytes [7, 93]. In addition to storing this vast amount of data, searching the data is another essential feature to enable reusability [94, 95]. However, alignment based searches like BLAST [96] are computationally too expensive at this scale. Therefore, searches based solely on the metadata became the default approach. Such an approach is rather limited, though, as metadata can be missing, misclassified or outdated.

In 2016, Solomon and Kingsford had a breakthrough, when they constructed the Sequence Bloom Tree, an index that could store almost $2,600$ RNA-Seq experiments, while supporting queries based on the sequencing information itself instead of metadata [97]. Since then, a wide range of indices were developed improving and outperforming the initial index by Solomon and Kingsford. An overview of the timeline of these indices and their surrounding data structures is given in Figure 3.1. Excluded from this overview are approaches that were developed with pangenomic applications in mind and therefore do not handle diverse data collections well, for example VariMerge [98]. While most indices will benefit from a high similarity between different sequencing experiments, the excluded indices rely heavily on them and are not an actual competitors to Solomon and Kingsford's index.

This fast development and the variety of solutions, spanning only a couple of years, indicates the importance of the underlying issue of the ever growing sequence databases and the need for finding an efficient solution.

At the beginning, all indices were non-quantitative, however in 2020 with the introduction of *Reindeer*, the first quantitative index was published [8]. From then one further indices followed adapting existing non-quantitative versions to support quantification. The focus of this work is on quantitative indices, especially on the current state of the art index *Needle* [32]. However, an overview of the non-quantitative ones are given as well because the approach of *Needle* could be adjusted to these methods.

In other works, the non-quantitative indices were separated in either color ag-

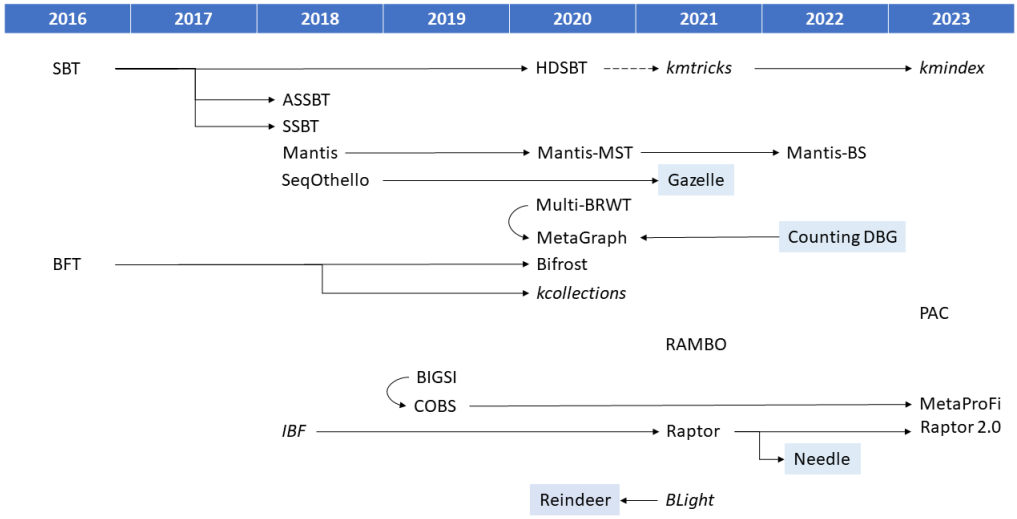


Figure 3.1: **Overview of indices for large collections of sequencing data.** Each index or data structure is listed under the year it was published, either in a peer-reviewed journal or on a pre-print server such as bioRxiv. Closely related works are marked with an arrow. Some arrows seem to go back in time, this is the result of the publication process of papers, which varies a lot. In the case of *kmtricks*, the dotted arrow symbolizes that *kmtricks* in its current implementation is based on the HDSBT, but could be implemented with other indices as well. Indices with a blue background are quantitative. Data structures marked in cursive are not forming a complete independent index. The reference to each mentioned index or data structure can be found in the appendix Table A.1. The figure is an adapted version of the figure found in [99].

gregative and k-mer aggregative methods [100] or in exact methods based on colored de Bruijn graphs and probabilistic methods based on Bloom filters [101]. For almost every index these differentiation describe the same index with color aggregative methods equaling the exact methods and k-mer aggregative methods equaling the probabilistic ones, with the exception of *SeqOthello* [102], which is a color aggregative method [103] and probabilistic, although not based on Bloom filters. However, the distinction based on the two data structures, the Bloom filter and the colored de Bruijn graph, shows how essential the two data structures are. Therefore, both are explained in more detail below.

3.1 Bloom filter

A Bloom filter requires a bit vector of length n, h hash functions and a false positive rate (p_{fpr}). An element is inserted into a Bloom filter by hashing the element with each of the h hash functions. These hash values are then seen as positions in the bit vector and the bits at these positions are flipped to one (see Figure 3.2 for an

example) [104].

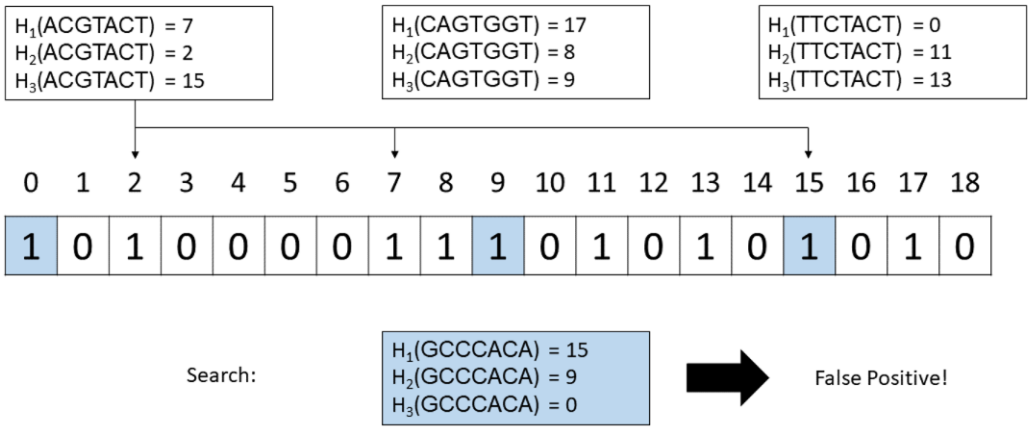


Figure 3.2: **Example of a Bloom filter.** Three elements 'ACGTACT', 'CAGTGGT' and 'TTCTACT' are inserted in an empty Bloom filter, thereby flipping the bits at the positions that their hash functions are indicating. Despite not being inserted, the element 'GCCCACA' is found because its hash values point to positions that have been flipped by other inserted elements.

A query works similarly. Again, the hash values are calculated and their positions in the Bloom filter are looked up and combined with a logical AND. If the logical AND results in one, meaning there was a one at every position, the element is considered present, otherwise the element was not inserted in the Bloom filter [104].

Bloom filters can have false positives, but not false negatives. The false positives are the result of different elements flipping bits to one than the queried element (as shown in the example in Figure 3.2). The probability for a false positive depends on the size of the vector, the number of hash functions and the number of inserted elements m [104]:

$$p_{fpr} = (1 - (1 - \frac{1}{n})^{h \cdot m})^h \approx (1 - e^{\frac{-mh}{n}})^h. \tag{3.1}$$

3.2 Colored de Bruijn graph

A de Bruijn graph (DBG) [105, 106] is a graph that is widely used in bioinformatics, for example in genome assembly. The graph consists of k-mers as nodes and two nodes are connected with a directed edge, if and only if the suffix of the starting node equals the prefix of the ending node [107, 108]. Note, this definition is node-centric as used by Marchet et al. [8] (see Figure 3.3 for an example).

Often there is a path in a DBG, where all nodes have only one incoming and outgoing edge, with the exception of the first and last node, which can have multiple

incoming or outgoing edges respectively, such a path is called an unitig. Storing unitigs instead of the singular nodes is space efficient, therefore this version of a DBG is called a compressed or compact DBG [108].

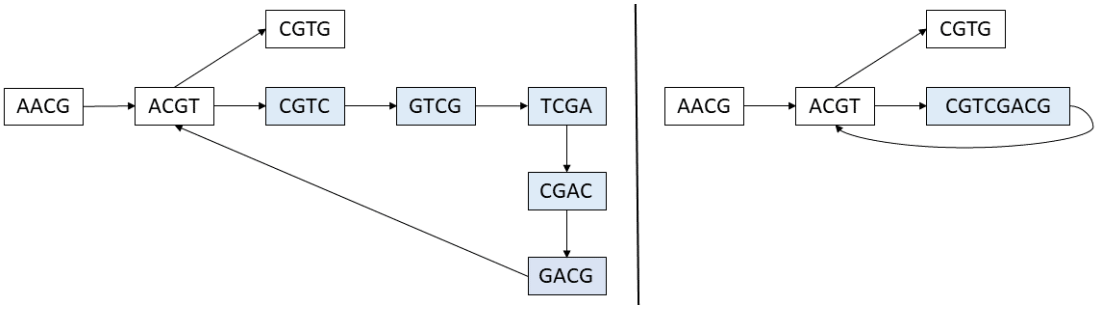


Figure 3.3: **Example of a DBG.** A DBG for the sequence "AACGTCGACGTG" for 4-mers. Marked in blue is the unitig path that can be compressed to one node in a compressed DBG (the graph on the right).

A colored DBG (cDBG) is a DBG built over k-mers from different sequencing experiments, where each node is associated with a color that symbolizes in which experiments the k-mer can be found [109].

3.3 Error handling

All indices have to deal with the fact that sequencing experiments are not error free. Most indices handle errors by applying a cutoff and disregarding all k-mers which appear with a lower frequency in one sequencing experiment than the given cutoff value [65, 97, 110, 111]. Therefore, determining the cutoffs is crucial and most indices rely on the cutoffs introduced by Solomon and Kingsford [97].

These cutoffs are shown in Table 3.1 and are dependent on the file size. Solomon and Kingsford do not clarify, if they meant the uncompressed or gzipped compressed file size [110], but by now it is established to rely on the gezipped file size [65, 110].

Cutoff	File size
1	0-300 MB
3	300-500 MB
10	500 MB - 1 GB
20	1-3 GB
50	$>$ 3 GB

Table 3.1: **Cutoffs based on file size.**

An alternative approach follows *kmtricks*, which keeps low-frequency k-mers, if they appear in multiple experiments because these k-mers are probably not the

result of an sequencing error despite their low frequency [111]. The same approach could be applied to other indices as well.

3.4 Non-quantitative indices

This overview aims to give an insight into the variety of ideas that have been applied to handle a large collection of sequencing data. Some ideas provide supportive data structures or algorithms for existing indices, for example *BLight*, *kcollections* or *kmtricks*, but do not form their own index (marked in cursive in Figure 3.1). These ideas are not explained in detail here, but mentioned wherever needed.

Furthermore, it is worth mentioning that some indices were built with metagenomic applications in mind, for example *COBS* [21], while others target RNA-seq applications, for example SBT [97]. However, while the performance of an index might decrease in the task it was not built for, the underlying ideas are still of interest and some indices are even meant to work in both fields, for example *Raptor* [65].

3.4.1 SeqOthello

SeqOthello is based on a static minimal perfect hashing algorithm called Othello [102, 112, 113]. Unlike other minimal perfect hashing algorithms Othello allows a many-to-one mapping, meaning multiple keys can map to the same value, but multiple values can not be associated with multiple keys. In *SeqOthello*, these keys are k-mers [102].

SeqOthello contains multiple Othellos, one root Othello and multiple bucket Othellos (see Figure 3.4 for an example). The root Othello maps all k-mers found in all given sequencing experiments to the bucket Othello they can be found in. While the bucket Othellos map a k-mer to a bit vector containing the information in which sequencing experiments it is present in. If these bit vectors are sparse or if they contain a high number of ones, which happens when a k-mer appears in either only a few sequencing experiments or many, the bit vectors are stored via a more compact encoding utilizing the sparsity or density. A bucket Othello contains only k-mers with the same encoding and the total size of all bit vectors in one bucket is limited to a given size. In order to optimize the space consumption further, k-mers that appear in only one sequencing experiment are stored only in the root Othello and return that singular sequencing experiment [102].

A k-mer from a queried sequence is then found by checking maximally two Othellos. If a k-mer has been inserted in *SeqOthello* during construction, the correct answer will be returned. However, if a k-mer that has not been seen in any sequencing experiment is searched for, *SeqOthello* may return a false bit vector and

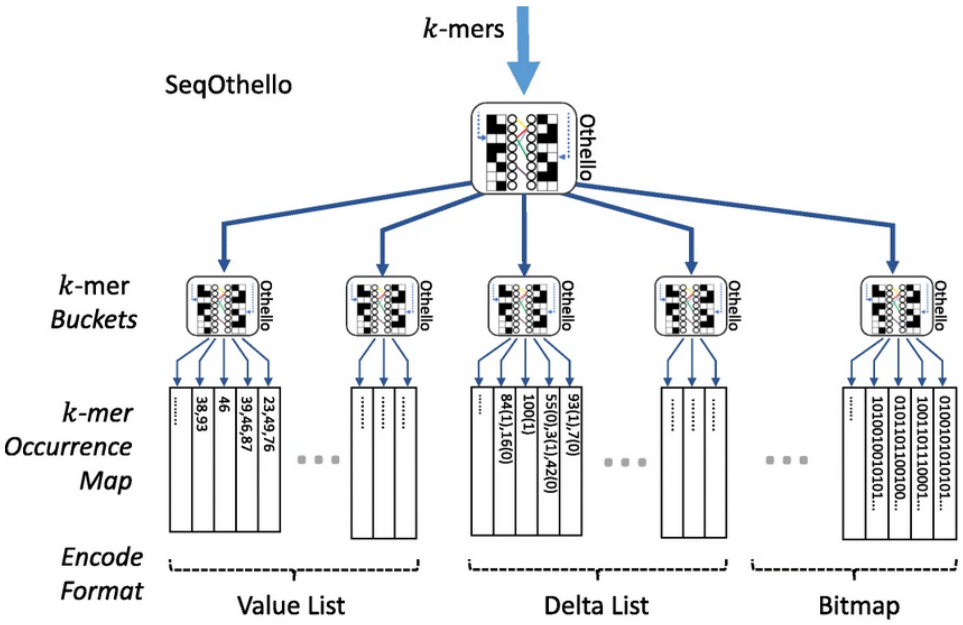


Figure 3.4: **Overview of SeqOthello.** All k-mers from all sequencing experiments are determined and then added in an Othello, which then stores for one k-mer the bucket Othello it is in. These buckets point to bit vectors containing information, in which sequencing experiment a k-mer occurs, here called occurrence map. These bit vectors can be encoded in different formats to save space, which are called Value List, Delta List and Bitmap here. The figure is taken from [102].

therefore lead to false positives [102]. As the number of false positives are dependent on the given input, the probability of a false positive decreases with the diversity of the given sequencing files.

Furthermore, *SeqOthello* has a dynamic update in form of insertions implemented [102].

3.4.2 Mantis

Mantis is based on a counting quotient filter (CQF) and a binary matrix and implements a cDBG [110].

The CQF is at its base a quotient filter, which is a probabilistic data structure that divides a value into a quotient and a remainder. The quotient consists of the first q bits of the value and the remainder of the rest. Inserting an element works by determining the position in the quotient filter based on the quotient and then storing the remainder there. If there is a hash collision, the second remainder is stored in the next free slot on the right, while keeping track of whether the information filled in was due to a hash collision [114].

A CQF adapts this data structure by solving hash collisions in such a way that collided remainders are arranged in an increasing order. Whenever, an element

is inserted more than once, a remainder is used to keep track of the counts, as this remainder does not follow the increasing order, it can be distinguished from element's remainders. Like the BF the CQF is a probabilistic data structure and therefore can return a higher count for an queried element than the actual count value. [114]. However, unlike BFs the CQF can be exact, when using an invertible hash function and thereby preventing hash collisions [110].

The *Mantis* index is built by determining for a k-mer in which sequencing experiment it appears by creating a bit vector with this information. If such a bit vector is not present yet in the binary matrix, the bit vector is added together with a color ID. If it is present on the other hand, the color ID is retrieved. The k-mer itself is then stored in the CQF with the color ID as its count value (see Figure 3.5 for an example). In order to construct a smaller CQF, *Mantis* uses a greedy approach to give the most common bit vectors smaller color IDs [110]. Moreover, the bit vectors can be stored more efficiently by applying a minimal spanning tree structure, where the difference between the bit vectors to their parent node are stored [115].

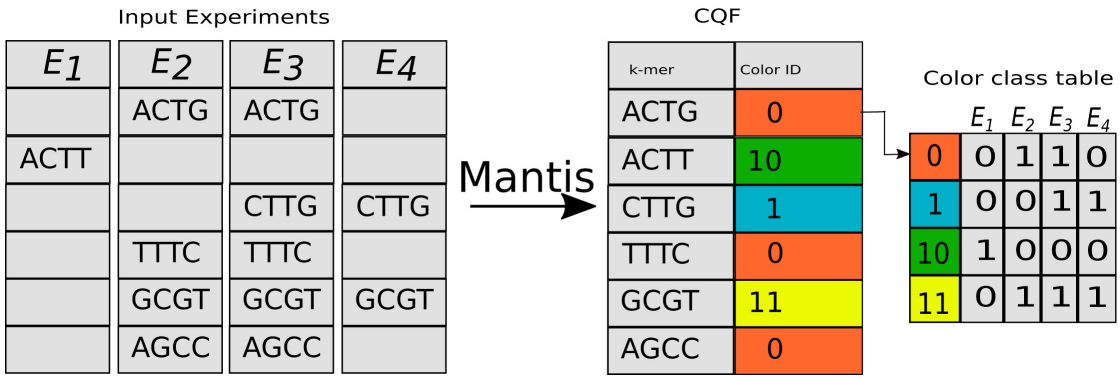


Figure 3.5: **Overview of Mantis.** The k-mers of all four sequencing experiments are determined and inserted into the CQF with a color ID. The color ID references the row in the binary matrix called color class table, which contains bit vectors with information in which experiments a k-mer can be found. For example the k-mer "ACGT" can be found in the sequencing experiments E_2 and E_3. The bit vector in the color class table, which shows presence in these experiments is the bit vector in the first row with the color ID 0 and therefore "ACGT" is stored in the CQF with the color ID 0. The figure is taken from [110].

A k-mer from a queried sequence is then found by looking up its hash value in the CQF and retrieving the color ID, to then check the binary matrix to access the bit vector showing in which experiments the k-mer is present. The bit vectors from all k-mers in a query sequence can be summed up and then based on a given threshold the sequencing experiments containing the query sequence can be reported [110].

As the CQF can be exact, *Mantis* is mostly used in its exact mode. However, using *Mantis* with a false positive rate can lead to false negatives. This is caused

by the fact that false positives in the CQF can lead to the return of higher color ID values than the ones inserted for a k-mer and therefore to a completely different bit vector, where sequencing experiments are missing or added compared to the actual bit vector of that k-mer (for example in Figure 3.5, color ID 1 is returned for "ACGT" instead of 0 and sequencing experiment E_2 would not be found, but E_4). For these false positives, Pandey et al. show that because multiple k-mers are considered these randomly found sequencing experiments have with a high probability no impact on the end result [110]. A similar argument could be made for the false negatives, but false negatives are often seen as more problematic as false positives, because they can not easily be verified in a downstream analysis.

Moreover, *Mantis* in its most recent version reduces the memory consumption by loading the index only partially by applying a partition based on minimizers [116].

Furthermore, *Mantis* allows a dynamic update by applying the Bentley-Saxe transformation, which allows insertions, but not deletions [116].

3.4.3 Bifrost

Bifrost is based on an array, a hash table and an array of binary matrices and implements a cDBG.

The array contains all unitigs extracted from the sequencing experiments. For these unitigs (k, kk)-minimizers, with kk being a user defined value so that $kk < k$, are determined. These minimizers are then stored in a hash table with the information, in which unitig they were found and at which position in the unitig. As a minimizer might appear in different unitigs, multiple values might be stored for one minimizer value. Furthermore, each unitig is associated with a binary matrix in which each of a unitig's k-mer is represented by a binary vector, where a one at position j indicates that the k-mer can be found in sequencing experiment j (see Figure 3.6 for an example). As the binary matrices can take up a considerable amount of space, multiple compressing strategies are applied by *Bifrost* [117].

A query sequence can be found with *Bifrost* by extracting all k-mers from the sequence and then searching for the minimizer of each k-mer in the hash table. If the minimizer can not be found, the respective k-mer is not present in any sequencing experiment. If the minimizer is found, each unitig it is linking to has to be checked and compared to the k-mer, if the k-mer is present in a unitig, the associated binary matrix needs to be checked to retrieve the binary vector with the information in which experiments the k-mer can be found in. The bitvectors of all k-mers can be summed up then and based on a given threshold, it can be determined in which sequencing experiments the query sequence can be found [117].

Unlike all other indices, *Bifrost* allows inexact searches of k-mers with one se-

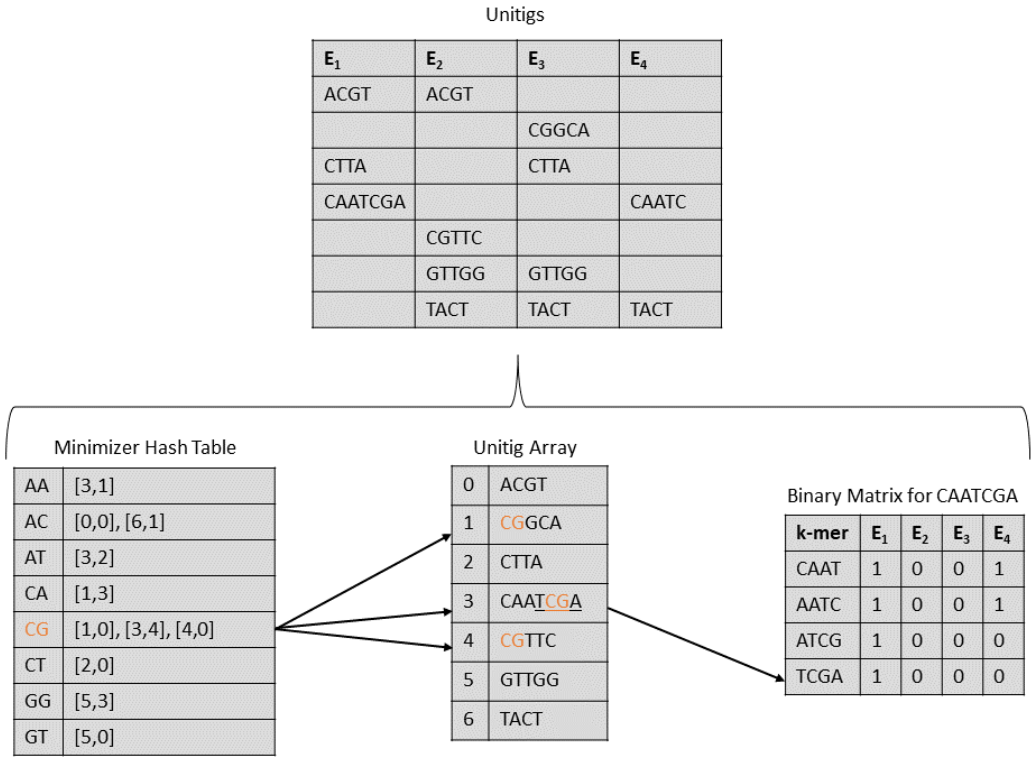


Figure 3.6: **Overview of Bifrost.** *Bifrost* is built over four sequencing experiments with 4-mers. The unitigs of at least length 4 for all given sequencing experiments are determined and stored in an unitig array. In a minimizer hash table, all $(4, 2)$-minimizers are stored with a list of tuples, where each tuple stores a position in the unitig array and a position in the unitig sequence. For example, the minimizer "CG" is a minimizer in three unitig sequences, which can be found in the positions $1, 3$ and 4 in the unitig array. Those minimizer start at different positions in their unitigs, in the unitigs at position 1 and 4 in the unitig array, they start at position 0 of the unitigs, while in unitig at position 3 in the unitig array they start at position 4. Each unitig in the unitig array is associated with a binary matrix that stores which experiments a k-mer in the unitigs can be found. Querying the k-mer "TCGA" would work by first determining the $(4, 2)$-minimizer, which is "CG" assuming lexicographical ordering. Now, all unitigs "CG" is pointing at in the unitig array are considered by matching the unitig sequence to "TCGA". Only in the unitig at position 3 in the unitig array, the k-mer can be found. Then the k-mer is looked up in the binary matrix for that unitig and the bit vector stored for "TCGA" reveales, "TCGA" can only be found in the first sequencing experiments.

quencing error [117]. Furthermore, *Bifrost* supports dynamic updates by an implemented insertion option [117].

3.4.4 MetaGraph

MetaGraph [101] is another cDBG implementation and the first application that actually uses the DBG characteristics in a query.

MetaGraph is constructed from the individual DBGs based on each given sequencing experiment with the additional information how often a k-mer occurs in an experiment. From these singular DBGs the cDBG is constructed. Each k-mer in the cDBG is annotated with which sequencing experiment it occurs (see Figure 3.7 for an example). For these annotations different data structures are supported like a simple hash table or a compressed indicator vector and depending on which one is picked, *MetaGraph* becomes dynamic as certain data structures support insertions of new experiments [101].

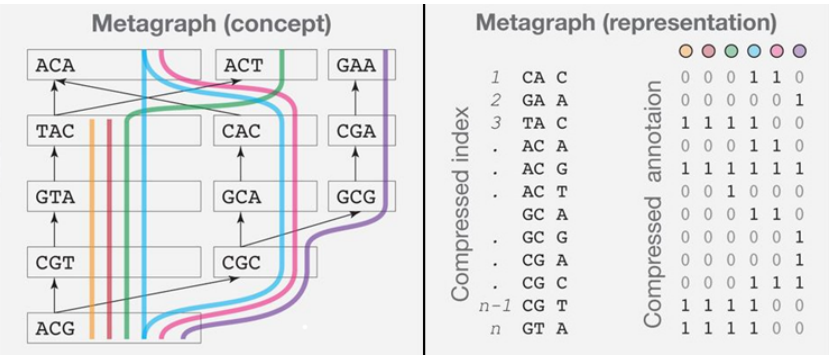


Figure 3.7: **Overview of Metagraph.** *MetaGraph* constructs a cDBG over all given sequencing experiments (shown on the left) and associates to each k-mer represented as a node in the cDBG with information, in which experiments it can be found. Here each experiment is symbolized by a different color and the association is achieved by a binary matrix. The figure is taken from [101] and slightly adapted.

A query sequence can be found by considering all k-mers in the given sequence and checking in the annotation matrix, in which experiments it is present and then returning the percentage of k-mers found per sequencing experiment. If multiple sequences are searched for or a long sequence, the query can be made more efficient by creating a query graph which then can make usage of repetitive k-mers appearing in different query sequences or multiple times in one sequence [101].

Alternatively, a query sequence can be aligned to all sequencing experiments at the same time by finding a path through the cDBG and keeping track, which k-mer has been seen in which sequencing experiment and the allowed distance [101]. Such an approach only makes sense though, if the stored sequencing experiments are genomes.

3.4.5 Sequence Bloom Tree

The idea of a Sequence Bloom Tree (SBT) is to create a binary tree of BFs, where one leaf represents one sequencing experiment and stores all the k-mers of that experiment, while the internal nodes represent the union of their children (see Figure 3.8 for an example) [97].

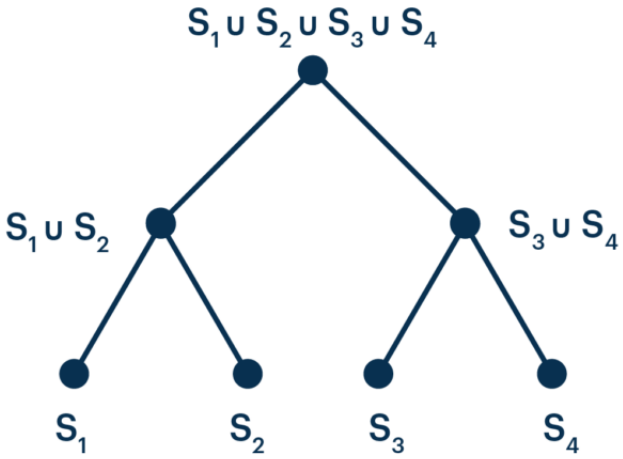


Figure 3.8: **Example of a SBT.** There are four sequencing experiments represented by S_x. Leaves of the tree represent one of these experiments, while the internal nodes are the unions of their children. The figure is taken from [118].

Searching through a SBT for a specific sequence like a gene works in the following way: The k-mers in the query sequence are searched for in the root node, if enough k-mers according to a given threshold are present there, the children are considered. This happens recursively until a leaf node is reached or a node does not contain a certain amount of k-mers from the query sequence. Whenever this happens, the search can end there, as its children will not include more of the query sequence's k-mers [97]. The speed of the search is therefore dependent on the threshold, as a high threshold can exclude pathways in the tree faster.

The AllSome SBT (ASSBT) [119] and the Split SBT (SSBT) [120] apply similar improvements to the SBT. Both store two BFs for internal nodes, where one BF stores the intersection of the information stored in its children and the other one the union minus the intersection (see Figure 3.9 for an example) [119, 120]. Therefore, both approaches benefit from clustering similar sequencing experiments together because the shared information is then the greatest [119]. The difference between ASSBT and SSBT is that the ASSBT does not store the k-mers found in the union of the parent node [119]. Therefore, the SSBT keeps some redundant information. However, in the compressed version of the SSBT this redundancy is removed [120]. As both approaches, the ASSBT and the SSBT, are motivated by the idea to use less redundant information, they achieve a smaller index size than the original SBT [119, 120].

Furthermore, the search of the k-mers from a query sequence is faster as some answers can be found quicker. As soon, as a k-mer is found in the intersection BF of one internal node, it is known that the k-mer is present in all its children [119, 120].

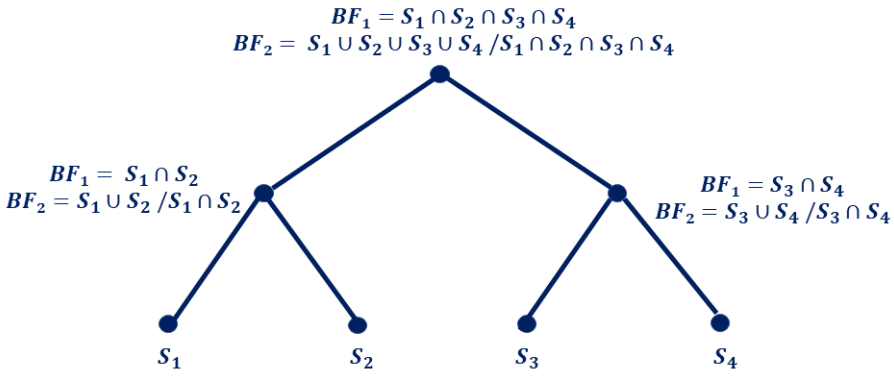


Figure 3.9: **Example of a SSBT.** There are four sequencing experiments represented by S_x. Leaves of the tree represent one of these experiments, while the internal nodes store two BFs: One containing the intersection of its children and one containing the union minus the intersection. The figure is inspired by [118].

The HowDe-SBT (HDSBT) [121] improves the SBT further in terms of space and speed by making even better use of the shared information between sequencing experiments. The HDSBT also stores two BFs at its internal nodes. One BF contains the stored information of the union of the intersection of its children nodes and the complement of the union of its children nodes (from here on first BF), while the other BF contains the information of the intersection of its children (from here on second BF) [121].

For the search of a query sequence two counters, called present and absent, are necessary. They are initialized with zeros. Then the k-mers of a query sequence are processed by checking if the k-mer is found in the first BF of an internal node starting with the root, if it is present, the second BF is checked, if the k-mer is found there as well, the present counter is increased by one, if not the absent counter is increased by one (see Figure 3.10 for an example). Whenever, the present counter reaches the given threshold, all the leaves of the current node contain the query sequence. Whenever, the absent counter reaches $1 - threshold$, the search does not need to investigate the current node's children further, as none of them will contain the query sequence [121].

All SBT implementations can support in theory insertion and deletions. However, so far no SBT implementation actually implemented this feature [122].

Moreover, as BFs are probabilistic data structures with a false positive rate, checking multiple BFs during a query for one k-mer leads to a multiple testing problem. Any k-mer found in an union BF (for ASSBT and SSBT) or any k-mer increasing the presence counter (for HDSBT) could be a false positive and because the query for this k-mer ends there, this cannot be corrected. Therefore, the more internal nodes are checked, the likelihood of a false positive increases and is overall

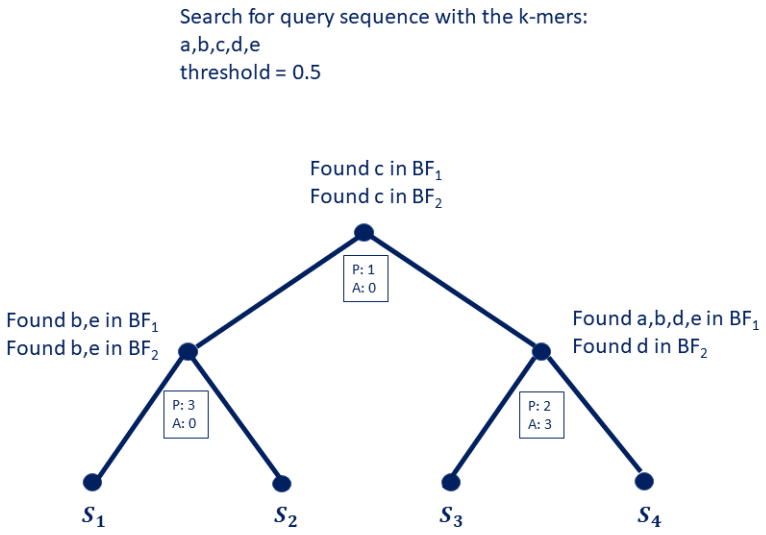


Figure 3.10: **Example of a search in the HDSBT.** There are five k-mers named a, b, c, d and e searched for with a threshold of 0.5. Before looking at any node, the counter present (P) and absent (A) are both zero. At the root node, the k-mer c is found in both BFs, so the present counter is increased by one. At the internal node on the left, the k-mers b and e are found in both BFs and the present counter is set to 3 now. As three exceeds the threshold because $3 >= 0.5 \cdot 5$, it is now clear that the sequencing experiments S_1 and S_2 contain the query sequence. Then the right internal node is checked, there the k-mers a, b, d and e are found in the first BF, but only d is found in the second BF. Therefore, the present counter is increased by one, while the absent counter is increased by three. Now, it is clear that the sequencing experiments S_3 and S_4 do not contain the query sequence.

greater than the false positive rate of the individual BFs. In the original SBT this was accounted for by the redundancy, neither Harris et al., Sun et al. nor Salmon et al. address this issue in their work of the HDSBT, ASSBT and the SSBT though [119–121].

3.4.6 COBS and MetaProFi

COBS, and its direct ancestor *BIGSI*, are in their essence an array of BFs, where each BF stores all k-mers in one sequencing experiment. These BFs are then combined in an array where each BF is a column and a row represents result of a hashed k-mer for each sequencing experiment [21, 123].

Therefore, searching a k-mer from a queried sequence works by finding the rows of a k-mer, one row per hash. The rows of the different hash functions are combined with an binary AND and result in a bit vector representing in which experiments the k-mer can be found. This can be done for all k-mers in a queried sequence and the resulting bit vectors can then be summed up to determine how many k-mers of

a queried sequence are present in each sequencing experiment (see Figure 3.11 for an example) [21].

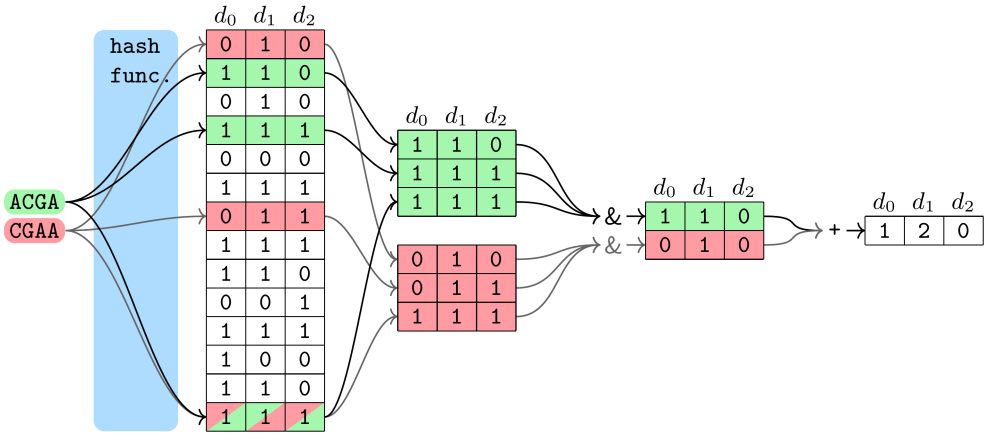


Figure 3.11: **Example of a search in COBS.** Two 4-mers "ACGA" and "CGAA" are searched in a *COBS* index built over three sequencing experiments named here d_0, d_1 and d_2 with three hash functions. Each hash function points to a different row and all three rows per 4-mer are extracted and then combined with a bitwise AND. The resulting bit vectors are then summed up for the final result, which show that the first sequencing experiment d_0 contains one of the searched 4-mers, the second one both and the third one none. The figure is taken from [21].

In order for a k-mer to point to the same positions in all BFs, the BFs need to have the same size and the same hash functions. Sequencing experiments can have a different amount of k-mers and fixing the size of all BFs based on the biggest experiment as done by *BIGSI* leads to a waste of space for smaller experiments, while fixing the size of all BFs based on the smallest experiment would lead to an increase of the false positive rate for the bigger experiments. Therefore, *COBS* offers a compact version of the index, where different sized BFs are allowed. In the compact version, similar sized sequencing experiments are clustered together and still have the same BF size, while different clusters can have different BF sizes. In that case the hash functions for one k-mer can not point to the same positions in the different BFs. However, *COBS* uses still one hash function for all by picking a hash function whose output range is larger and then applying a modulo operator to fit it into the individual BF sizes [21].

The clustering of the compact version makes an insertion harder, as a new sequencing experiment can not simply be added to the array, but needs to be inserted in the best fitting cluster and this cluster needs to have space allocated for additional experiments. At the moment, only a merge operation for non compact indices are offered [21].

MetaProFi is an index from a different research group but the underlying data structure is the same as in *COBS*. The difference to *COBS* lays in the compression

of the data structure, *MetaProFi* divides the array in batches and chunks, which are compressed individually and thereby achieve a better compression ratio. Moreover, *MetaProFi* is the only index so far that is written in Python and Cython for the computationally heavy parts and might be easier to use for non-computer scientists [124].

3.4.7 RAMBO

The idea of *RAMBO* is to not store one BF for each given sequencing experiment, but several BFs storing multiple sequencing experiments and thereby reducing the allocated space. To achieve this, *RAMBO* is at its core an array of BFs [125].

RAMBO is constructed by generating all the k-mers in a sequencing experiment and partitioning the sequencing experiments into B groups. Each group can contain more than one sequencing experiment and is stored in one BF, which is called a BF union. All of these groups represent one column in the array of length R. For the other $R-1$ columns, this procedure is repeated with a different independent partition. Therefore each sequencing experiment is stored R times together with other experiments (for an example see Figure 3.12) [125].

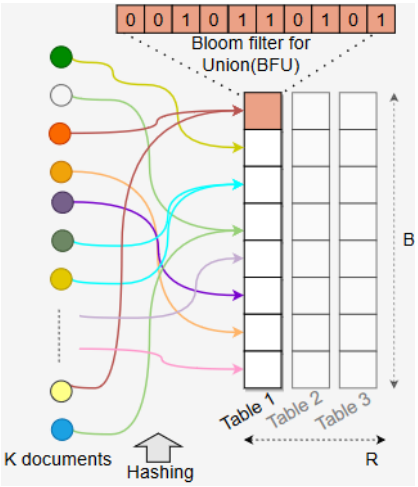


Figure 3.12: **Overview of Rambo.** The K sequencing experiments are partitioned into B groups, for the first entry in the first row (marked in color), the orange and the yellow sequencing experiments are combined into one BF, the Bloom filter union. For the other remaining $R-1$ columns a different partition leads to different sequencing experiments combined in an union. The figure is taken from [125].

Answering for a query sequence then works by checking each k-mer in every $B \cdot R$ BF union. If the k-mer is found in one BF union, the list of sequencing experiments of this union is saved. Therefore, by checking all BF unions, multiple list of sequencing experiments are obtained, these lists are then intersected to determine all

experiments containing the k-mer. As this is repeated for each k-mer, the percentage of k-mers found for each experiment can be determined [125].

Due to the combination of sequencing experiments the false positive rate is not only dependent on the false positive rate of the individual BFs, but also on the partitioning. A k-mer can be considered found in an experiment it is not actually contained, if the k-mer is present in the sequencing experiments this false positive experiment is grouped with over the different columns. However, Gupta et al. argue that the overall false positive rate is still low, if most k-mers appear in only a small number of sequencing experiments [125]. This assumption might hold for metagenomic applications, but not for usage with RNA-seq.

Furthermore, Gupta et al. do not mention the underlying multiple testing problem that can also impact the false positive rate. Addressing it could improve the accuracy.

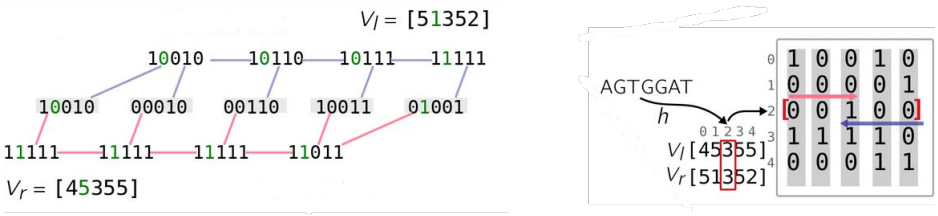
3.4.8 PAC

PAC is at its core a collection of multiple arrays of BFs, where a single array of BFs can be compared to the single array in *COBS*. A k-mer in one of these arrays points to a row, which contains the information, in which sequencing experiment it occurs [103].

PAC is constructed by obtaining all k-mers from all sequencing experiments and partitioning them based on their (k, kk)-minimizers with kk being a user defined value so that $kk < k$. Each minimizer leads to one partition consisting of an array of BFs for k-mers with the same minimizer and two integer arrays. These integer arrays are obtained from leaf BFs, where each leaf BF contains the k-mers of one partition that is present in one sequencing experiment. From these leaf BFs the integer arrays are obtained by first creating the union of adjacent leaf BFs, starting with one from the left and the other from the right, and then counting the number of uninterrupted ones for each position starting from the BF union containing all leaf BFs to a single leaf BF (see Figure 3.13a for an example) [103].

The integer arrays are used to reduce the search space of a query. For a queried sequence, all k-mers with the same minimizer are handled at the same time, as they point to the same partition. By applying the same hash functions as the ones in the leaf BFs, the positions in the integer arrays and the arrays of BFs is determined. The found number in the integer arrays mark a range from left and right of a row in the array of BFs and only at the overlap of these two ranges, the sequencing experiments could contain the queried k-mer. Therefore, only those positions need to be checked (see Figure 3.13b for an example) [103].

Moreover, *PAC* offers an update option by allowing the insertion of multiple



(a) Construction (b) Query

Figure 3.13: **Overview of PAC.** Both figures show one partition. In (a) the leaf BFs are shown marked with a grey background. The top and bottom rows show the adjacent union BFs starting from the left respectively right of the leaf BFs. The two integer arrays are here named V_l and V_r. In V_l the second position marked in green stores a one as there is only one uninterrupted one starting from the last union BF. While V_r stores a five because there is a one in all five union BFs. In (b) the k-mer "AGTGGAT" is searched for, the correct partition was already determined by its minimizer and now the hash functions determine the positions in the integer arrays V_l and V_r as in the array of BFs. In this example, there is only one hash function, so only one position is considered. The number in V_l determines the range starting from the right, while V_r determines the range starting from the left. There is an overlap of these two ranges for only one column, meaning only in the sequencing experiment represented by this column the k-mer might be found. As a one is stored at that spot, it is found. Both figures are taken from [122] with small adaptations.

sequencing experiments, the costs of these insertions is similar to the cost of the build of an independent *PAC* over these experiments [103].

3.4.9 Raptor

Raptor is the only application that also supports minimizers alongside k-mers. The index of *Raptor* was first based on the interleaved BF [65] and has been recently replaced with the hierarchical interleaved BF [126]. Both data structures are explained in detail below in 3.6.2 and 3.6.3. According to the analysis of Seiler et al. and Mehringer et al. *Raptor* is currently the best performing index in terms of construction time, index size and query time [65, 126].

3.5 Quantitative indices

In the last three years, quantitative indices were simultaneously introduced as a way to quantify a query for thousands of sequencing experiments. A quantitative index can always answer a presence or absence query, as any occurrence greater than zero can be interpreted as present. However, as the indices are optimized for the quantitative query, this might be less efficient than using a non-quantitative index.

3.5.1 Reindeer

Reindeer was the first quantitative index and is at its core a cDBG, where each k-mer is associated with a count vector representing the number of occurrences in each sequencing experiment [8].

In order to build *Reindeer*, a preprocessing step is necessary, in which the compact DBG for each sequencing experiment is determined and each unitig in the DBG is associated with a count value. This count value is the average of the number of occurrences of each k-mer contained in the unitig [8].

Given the individual compact DBGs, an union DBG is constructed. This union DBG is built in a compact form, but instead of using unitigs, it is based on monotigs. Monotigs are paths in the union DBG, where each k-mer is associated to the same count vector and has the same (k, kk)-minimizer with $kk < k$ (see the Figure 3.14 for an example). These monotigs are determined by a greedy algorithm and in the end are associated to count vectors [8].

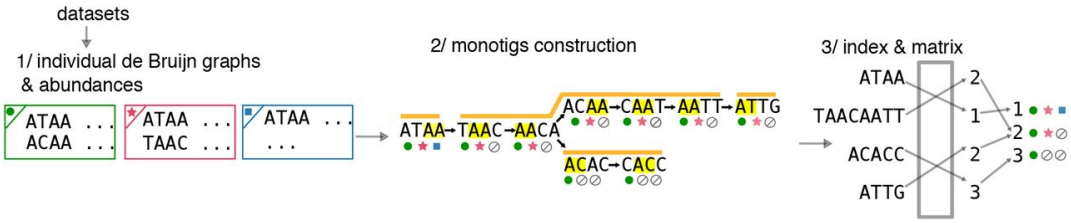


Figure 3.14: **Overview of Reindeer.** From each data set a DBG with the count values is constructed (1), from there the monotigs are determined (2) and then stored in an index, where each monotig is associated to a count vector (3). In (2), the $(4, 2)$-minimizers in the 4-mers are marked in yellow and the singular monotigs are overlined in orange. For each 4-mer the count values are marked with the three symbols circle, star and rectangular. These symbols not only mark, if a 4-mer is present in a given data set, but also the count value, which for simplicity is the same for all the shown 4-mers here for one data set. The figure is taken from [8].

The count vectors can be stored in a more space efficient way by discretization of the count values, for example by log-transforming the counts. Doing so will decrease the accuracy compared to the original counts [8]. However, even when *Reindeer* is used without a discretization it is not exact as the the count values over unitigs in the preprocessing step are averaged.

A query sequence is quantified by first determining in which sequencing experiments the query sequence can be found. This is the case in the default mode, if at least 40 percent of the query sequence's k-mers have count values greater than zero. If a query sequence is found in a sequencing experiment, the count values for each k-mer are determined and returned [8].

Therefore, *Reindeer* does not determine a singular expression value for each

sequencing experiment and it is up to the user, what to do with the received count values. Similar to *Needle count*, the median of these count values could be used and lead to a good result. However, the approach from *kallisto* or *salmon* would also work and Marchet et al. mentions such an approach as future work [8].

3.5.2 Gazelle

Gazelle could be described as the quantitative update to *SeqOthello* and therefore also builds upon the Othello data structure [127].

The first step in the *Gazelle* construction is to build a *SeqOthello* and then construct for each given sequencing experiment a hash table based on minimal perfect hashing, which maps every k-mer contained in an experiment to its number of occurrence [127] (see Figure 3.15 for an example).

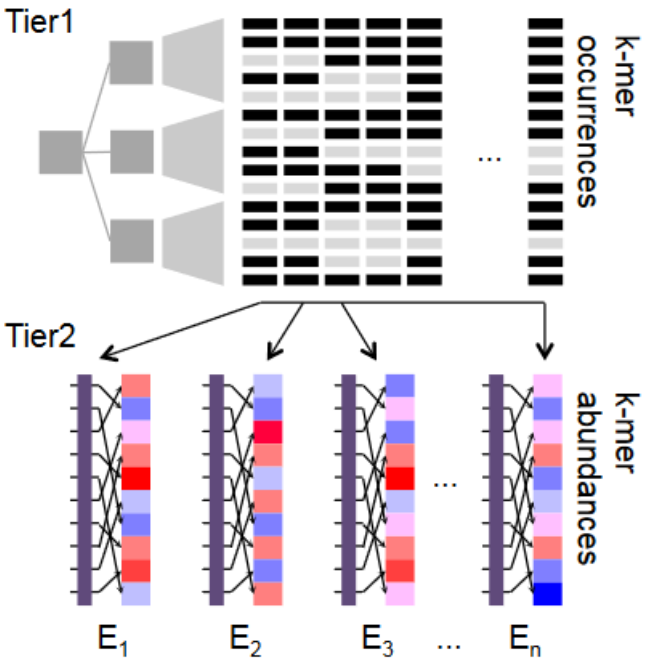


Figure 3.15: **Overview of Gazelle.** The first part named Tier 1 is a *SeqOthello* index, while the Tier 2 consists of n hash tables mapping each k-mer in a sequencing experiment to its number of occurrence in an experiment. The figure is taken from [127].

As the exact number of occurrences often are not necessary, *Gazelle* uses the log-function to discretize the values, similar to *Reindeer*, thereby saving space [127].

A query sequence's expression is determined by searching for all its k-mers in the *SeqOthello* and once a certain percentage of the k-mers has been found in a sequencing experiment, the sequencing experiment's hash table is considered and each k-mer's count value determined. From these k-mer count values the interquartile

mean is calculated as the expression value [127]. The interquartile mean is picked to exclude outliers [127], similar to *Needle count*'s median approach.

3.5.3 Counting DBG

MetaGraph supports associating the k-mers in their cDBG with information about their occurrence, they call this version of the DBG the counting DBG [95, 101].

The association of k-mers to their count values in different experiments works by compressing a integer array, where an entry represents the count value of a k-mer in one given sequencing experiment [95].

The compression of the integer array works by splitting it into two parts, a binary array of the same size as the integer array and a smaller integer array. The binary array has a one at each position where the count values of the integer array are greater than zero. As this is a sparse binary array, common compression strategies can be applied, Karasikov et al. use a Multi-BRWT [128]. The smaller integer array does not store any of the zero values and is thereby smaller. In order to find the correct entry during a k-mer query, the binary array is checked first, if the position the k-mer hashes to is set to one in a column meaning the k-mer is present in a specific sequencing experiment, the rank of that column returns the position in the smaller integer array and the correct count value can be retrieved (see Figure 3.16 for an example) [95].

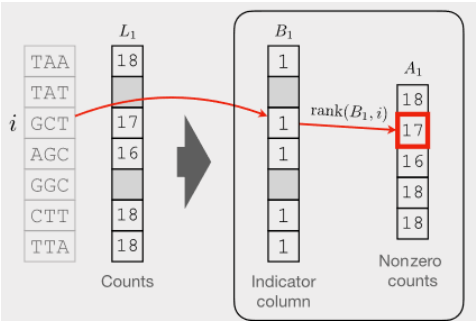


Figure 3.16: **Example of storing the integer array in the Counting DBG.** The example shows one column of each array. In L_1 the counts for each 3-mer are noted, this column is transformed in the binary column B_1 and the column A_1 storing all count values greater than zero. The rank of the value in B_1 gives the correct position in A_1. The figure is taken from [95].

In order to store smaller count values and achieving an even better compression, the counting DBG makes use of the fact that adjacent k-mers in a DBG will have a similar count value. Therefore, the actual count values are only kept for so called anchor nodes and for the other nodes the difference in their count values to their successor nodes are stored (see Figure 3.17 for an example) [95].

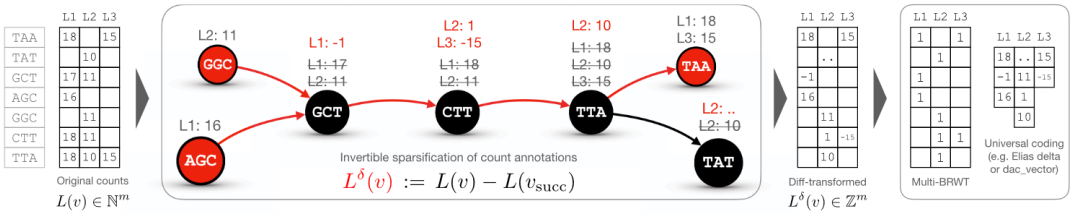


Figure 3.17: **Overview of the counting DBG.** The red nodes are anchor nodes. The node for the 3-mer "TAT" is not an anchor node as this is only a slice of a DBG and the node has a successor not shown. Also marked in red are the difference values stored instead of the actual count values, which capture the difference of a node to its successor. The figure is taken from [95].

Similar to *Reindeer*, the counting DBG does not return singular expression values, but all count values found for the k-mers of a query sequence, thereby leaving it to the user how to interpret these.

3.6 Needle

Needle is the current state of the art of the quantitative indices. The basic idea of *Needle* is to store a non-quantitative index multiple times, where each index stores the submers with a count value in a specified range. Therefore, *Needle* could have been implemented with any non-quantitative index, but was based on *Raptor*'s interleaved Bloom filter (IBF) [32]. First, an overview over *Needle*'s workflow is given. Then an introduction of the IBF and its alternative versions the hierarchical IBF [126] and the counting IBF follows including an discussion about the advantages and disadvantages of the different data structures for *Needle*. Afterwards, an in-depth description of *Needle*'s implementation is presented and lastly, insights into *Needle*'s normalization, dynamization and parallelization are given.

3.6.1 Workflow Needle

Needle constructs q IBFs with q being an input parameter. Each of these IBFs are called a level and each level has a threshold (t_q) associated to it. These associated thresholds increase with the levels, therefore $t_1 < t_2 < ... < t_q$ holds [32].

For each sequencing experiment the canonical k-mers or the (w, k)-minimizers (both from here on denoted minimizer) and their number of occurrence (n_q) in each experiment are determined. The minimizers are then inserted into the level that represents the range that their number of occurrence falls into, so that $t_l <= n_q < t_{l+1}$ holds. For the last level q, n_q only needs to fulfill $t_q <= n_q$ (see Figure 3.18 for an example) [32].

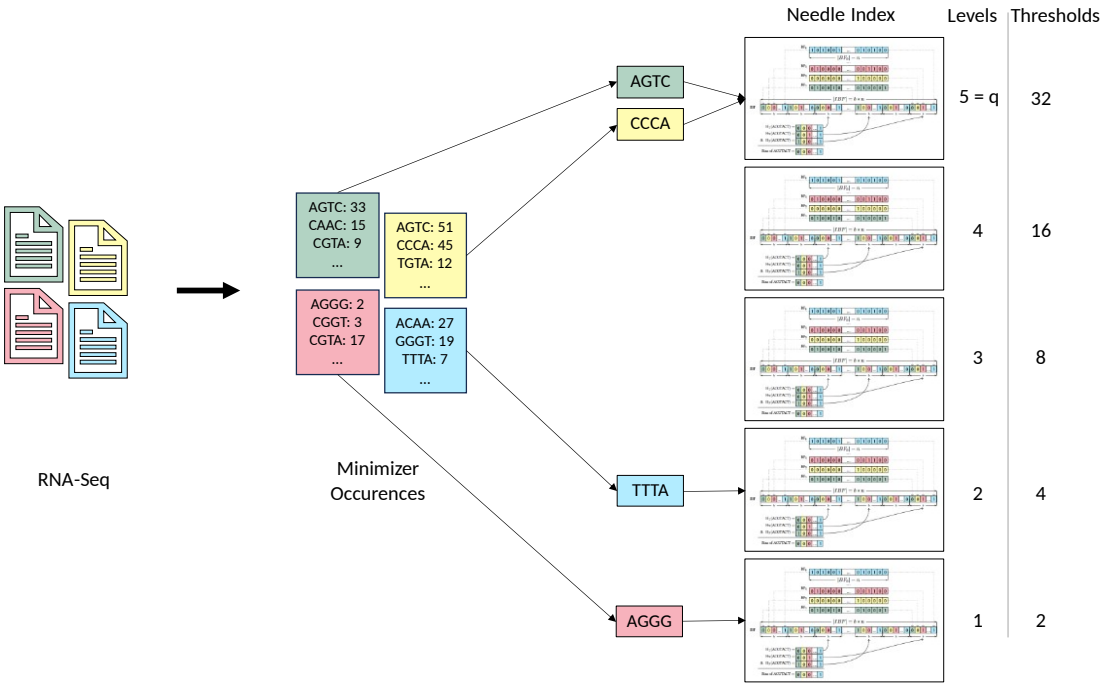


Figure 3.18: **Example of constructing a Needle index for** $(4, 4)$**-minimizers.** The exemplary *Needle* index consists of five IBFs. Each IBF is associated to a level and a threshold and stores all minimizers present for that level for each sequencing experiment. The index is constructed by first determining all minimizers and their respective occurrences for all given sequencing experiments. Then, the minimizers are inserted into the level for which $t_i \leq \text{occurrence} < t_i + 1$ holds. For example, minimizer 'AGGG' from the pink sample occurs two times and is therefore stored in the first level, because $t_1 = 2 \leq 2 < t_2 = 4$.

The thresholds of the level can be either the same for every sequencing experiment and defined by the user (user-defined thresholds) or the thresholds can be automatically determined by *Needle* itself for each sequencing experiment (automatic thresholds). The automatic thresholds are determined by recursively taking the median of the number of occurrences of all minimizers in a sequencing experiment. Thereby, reducing the minimizer content by half for each sequencing experiment at every level [32].

An optimal partitioning is not feasible as the underlying minimizer distribution usually does not contain any local minima that could be used as thresholds [32].

A query sequence is quantified by determining all its minimizers and initializing a integer array as counter that has a zero value for each stored sequencing experiment. Each minimizer is searched for in all levels, starting with the last level q and ending with the first level using the counter array to keep track how many minimizers have been found. Once half or more of a query sequence's minimizers are found for a sequencing experiment, the sequence is quantified for that experiment [32] (see Figure 3.19 for an example).

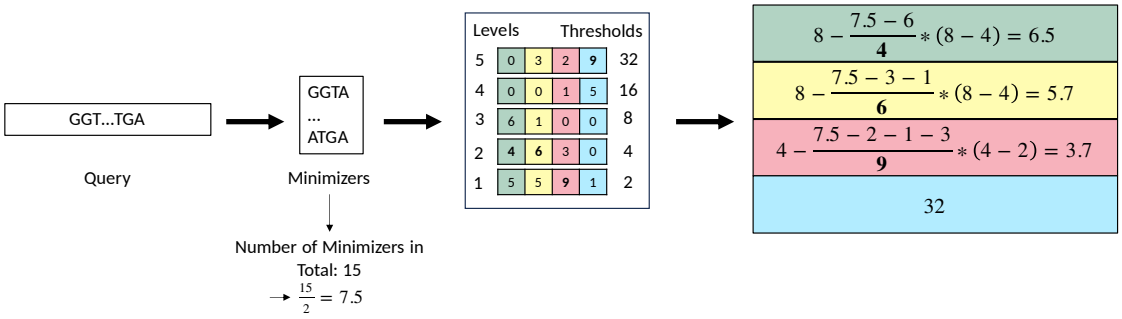


Figure 3.19: **Example of querying a Needle index for $(w, 4)$-minimizers.** The minimizers of a query sequence and the total number of minimizers in the sequence are determined. Afterwards, the minimizers are looked up in each level, starting with the highest level 5. The number of found minimizers per levels is then used to determine, if the expression value should be estimated, which is the case, if half or more of all minimizers in the query are found at that level or all higher levels. Here that level is marked in bold for each color (Level 2 for green and yellow, Level 1 for pink and Level 5 for blue). If the estimation occurs at the highest level, the threshold of that level becomes the expression value. On all other levels, Equation 3.4 is applied.

However, because the IBF is a probabilistic data structure this leads to a multiple testing problem and a correction is necessary. The observed number of found minimizers (n_o) consists of the number of true positives (n_{tp}) and the number of false positives, which can be estimated by the number of minimizers in the query sequence (n_s) multiplied with the false positive rate:

$$n_o = n_{tp} + n_s \cdot fpr. \tag{3.2}$$

Therefore, the number of true positives can be estimated by:

$$n_{tp} = \frac{n_o - n_s \cdot fpr}{1 - fpr}. \tag{3.3}$$

This estimated value n_{tp} is added to the counter at each level [32].

The actual quantification of the query sequence is based on the median of the number of occurrences of all minimizers like in *Needle count*. However, unlike *Needle count* the number of occurrences are not available, as only the level thresholds are known. The median is therefore estimated by linear interpolation of two adjacent levels [32].

Given a query sequence T with m minimizers, let $C_i(T)$ describe the estimated number of true positives n_{tp} found for the sequence T at level i for one sequencing experiment e. Furthermore, let level x_e be the level, where $counter[e]$ becomes equal to or greater than $\frac{m}{2}$ for experiment e and level $y_e = x_e + 1$ be the level after x_e.

Then the expression value for that experiment for the query sequence (μ_e) can be determined by:

$$
\begin{aligned}
a_e &= C_{x_e}(T') \\
b_e &= \sum_{i=y_e}^{q} C_i(T') \\
\mu_e &= t_{y_e} - \frac{\frac{m}{2} - b_e}{a_e} \cdot (t_{y_e} - t_{x_e}).
\end{aligned} \tag{3.4}
$$

In cases, where $counter[e]$ becomes equal to or greater than $\frac{m}{2}$ at level q, which is the first level considered, μ_e is set to t_{q_e} [32].

In theory, not all levels need to be considered, if a query sequence is already quantified before reaching the last level for each sequencing experiment. In practice, this is rather unlikely to happen, especially when considering more than one query sequence, which can be quantified simultaneously with a counter matrix instead of an array [32].

3.6.2 IBF

The IBF combines b BFs into one bit vector by interleaving them. This means that an IBF stores a bit vector of length b at every position instead of a single bit like BFs do (see Figure 3.20 for an example). Each bin, which stands here for one sequencing experiment, is represented by one BF.

Therefore, the size of an IBF is $b \cdot n$. The insertion and query work similar to BFs [66]. As the IBF consists of multiple BFs, the IBF does not have one single false positive rate, but multiple, one for each bin [32, 126].

It should be noted that the IBF is quite similar to *COBS* and shares similar ideas and concepts. Considering the fact that the IBF and *COBS*'s ancestor *BIGSI* were published with just a few months in-between, the similarities are a coincidence of independent research. However, despite their similarities, they differ in their implementation details and according to the analyses performed by Seiler et al. the IBF is the preferable non-quantitative index [65].

As an IBF is at its core a bitvector, known compression algorithms can be applied to decrease the size of the IBF at the cost of increasing the query time.

While the IBF does not allow the deletion of singular elements from a bin, it is possible to delete a complete bin by setting all its bits back to zero. These deleted bins can then be filled by insertions of new bins. If there has been no deletion or not enough for the numbers one wants to insert, the bin size of the IBF can be increased. Therefore, the IBF supports dynamic updates in the sense that whole

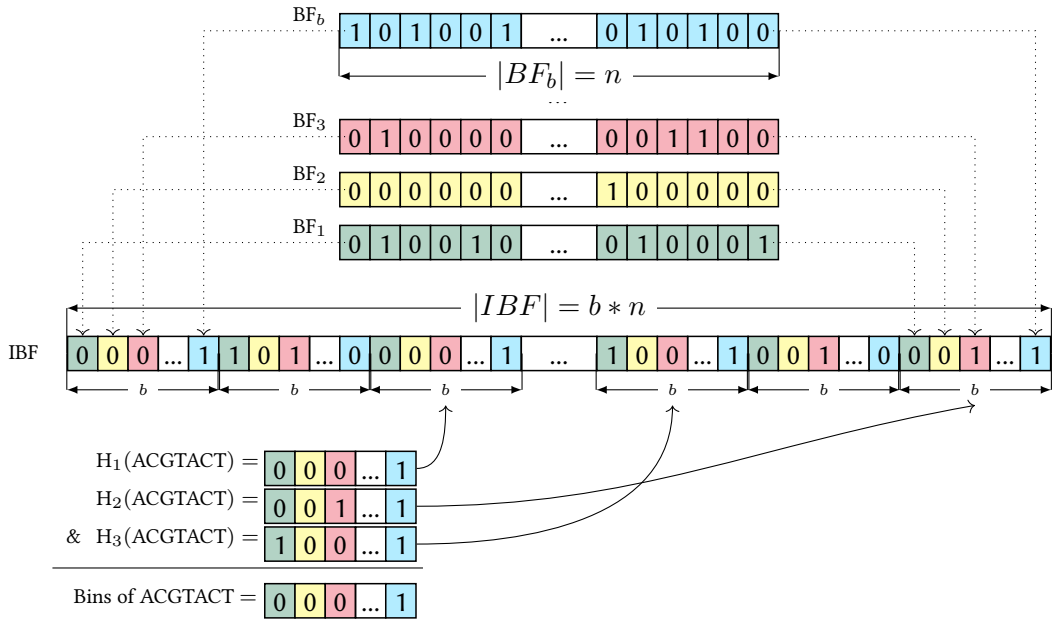


Figure 3.20: **Example of an IBF.** At the top, [b] individual Bloom filters of length n are shown, these were interleaved to create an IBF, of size $b \times n$ using 3 hash functions. When querying a [submer], here ACGTACT, the hash functions return 3 positions in the IBF, so that 3 sub-bitvectors can be retrieved. These sub-bitvectors are combined with a bitwise & to a final resulting bitvector, where a 1 indicates that the [submer] is found in that experiment. Here, ACGTACT can be found in the last (blue) experiment. The figure is taken from [66], the description with small adaptations from [32].

sequencing experiments can be discarded or inserted, however, adapting an existing bin by adding or deleting singular elements is not possible.

3.6.3 HIBF

The HIBF was motivated by the problem that hugely differently sized bins in the IBF would lead to drastically different false positive rates for each bin. If a false positive rate is guaranteed for all bins, this would lead to IBFs with a higher space consumption than necessary, because the smaller bins will then have more space than needed and a much smaller false positive rate. Therefore, the idea of the HIBF is to improve space efficiency of smaller bins by merging them and to reduce the space bigger bins need by splitting them (see Figure 3.21 for an example). For this reason, the term bins is now differentiated into technical and user bins [126].

User bins refer to the bins the user is interested in, for example singular RNA-seq files. As bigger user bins might get split or smaller user bins merged in the HIBF, the actual bins in the data structure are called technical bins [126].

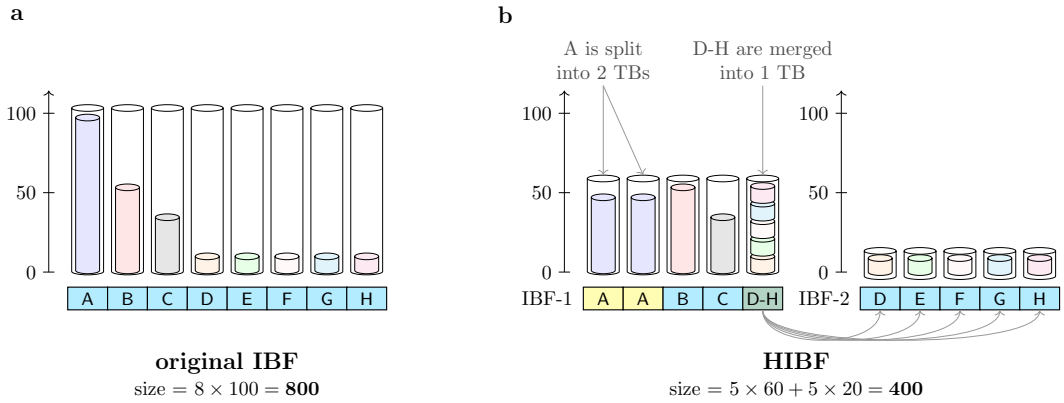


Figure 3.21: **IBF vs. the HIBF** Given an input of eight user bins (UBs), subfigure a displays the layout of a normal IBF storing the content of the UBs, represented by the inner, lightly colored cylinders, in one technical bin (TB) each. The outer cylinders represent the size of the TBs and visualizes the wasted space. The horizontal rectangular bar represents the layout, indicating which UBs, identified by their ID (A-H), are stored in which TB. The same semantics hold for subfigure b which displays the corresponding HIBF with a maximum of 5 TBs on each level. In this example, UB A is split into the first two TBs in IBF-1, while UBs D-H are merged into the last TB. The merged bin requires a second lower-level IBF (IBF-2) storing the content of UB D-H in individual TBs. The size is given in exemplary numbers. The figure and the description are taken from [126].

Splitting a user bin creates two or more technical bins. During a query, the technical bins belonging to one user bin are considered together, so that the counts in either bin are added up. Therefore, if an element is found in either technical bin, the query will return that the element is present [126]. Merging two user bins into one technical bin makes it necessary to store another HIBF, where the merged user bins are stored individually. This allows the differentiation of the different user bins in a query. As this second HIBF might also contain merged user bins, another HIBF might become necessary. Therefore, an HIBF consists basically of multiple levels of IBFs [126].

Before constructing the HIBF, it is necessary to know which bins should be merged and which should be split. To determine this, Mehringer and Seiler et al. introduced a preprocessing step that utilizes a dynamic programming algorithm to determine a good layout, thereby optimizing the space consumption of a single IBF of the HIBF [126].

The construction of the HIBF works then by inserting elements in the different IBFs, while keeping track of which technical bins relate to the different user bins [126].

Unlike the IBF, the HIBF tries to obtain guarantees an upper bound on the false

positive rate for all bins. The splitting and merging operations lead to a similar number of elements stored in the different technical bins in each IBF of an HIBF, meaning the false positive rate of the technical bins are roughly the same. However, splitting a user bin leads to a multiple testing problem, as the query result of all technical bins of a split user bin are summed up. To account for this, more space is allocated for split bins, leading to a smaller false positive rate for the split bin and thereby guarantees the false positive rate for a split user bin as well. However, by doing so, the false positive rate of every other technical bin decreases, meaning technical bins, which represent just a user bin without any split or merge now also have a smaller false positive rate [126].

Moreover, merged user bins have a lower false positive rate, because at least two IBFs are queried for them, meaning a false positive would need to be present in both IBFs to be not detected or a true positive at the higher levels and a false positive at the lower one [126].

Therefore, the HIBF like the IBF has different false positive rates for the different user bins, but can guarantee a certain false positive rate with a smaller space consumption than the IBF [126].

Querying an element for split user bins means checking multiple technical bins and for merged user bins checking multiple IBFs. Despite the need to query multiple IBFs, the HIBF was shown to be faster than a simple IBF due to the fact that the single IBFs in an HIBF are smaller than a simple IBF. Moreover, in most applications a query does not consist of a single element but of a longer query, for example a list of k-mers, and it is only of interest if at least a certain amount of these elements are contained in the HIBF. In these cases, a lookup at a deeper level is not necessary anymore if the threshold has been reached on the upper-level. Thereby speeding up the query even further [126].

Evaluation of the HIBF for the Needle workflow

The input of *Needle* is in most cases differently sized sequencing experiments, therefore, the better handling of the space of the HIBF is desirable for Needle.

As *Needle* stores q different IBFs, it would need to replace each IBF with an HIBF. As a layout needs to be determined for an HIBF beforehand, this leads to the question of if each level would need its own layout. If so, this would drastically increase the construction time, as every file would need to be processed twice: Once for determining how many elements go into which level and once to actually insert the elements in the HIBFs. However, assuming the same layout for each level seems reasonable as the difference between the sequencing experiments should not change between levels at least not for the automatic thresholds. Therefore, the construction time would only increase by calculating one layout. As a faster layout calculation is

currently researched, there is a good chance that this increase in construction time could become insignificant.

Furthermore, *Needle* does not profit from a speed up during a query resulting from reached thresholds, as *Needle* needs the exact number of elements found and therefore, always needs to examine every IBF in a HIBF.

Moreover, *Needle* needs for its correction function during a query the exact false positive rate per bin and would need to know, if a user bin is stored as a split, merged or in one singular technical bin to estimate the false positive rate correctly.

A major drawback of the HIBF is that dynamic updates are currently not supported. However, as with a possible speed up for constructing the layout, this is under active development.

In summary, the HIBF would probably slow down the construction and the search times, while at the same time saving space. As a construction usually only needs to happen once and the query times are still expected to be of the same magnitude, the advantages of the HIBF seem to outweigh the disadvantages and using the HIBF over the IBF seems to be preferable. Nevertheless, the missing dynamic updates are the reason why here the IBF was used instead. But once the HIBF supports the dynamic updates as well, its performance within *Needle* should be tested.

3.6.4 Counting IBF

There is a version of a Bloom filter that allows quantifying an element instead of a mere presence or absence call, this version is called counting Bloom filter (CBF) [129–132]. As Needle's aim is a (semi-)quantification, combining the ideas of a CBF with the ideas of the IBF or HIBF seem like an obvious alternative than storing multiple IBFs or HIBFs as done in the current *Needle* workflow. Therefore, the advantages and disadvantages of such an approach are discussed here, after formally introducing the CBF and then a possible combination of the CBF with the IBF called counting interleaved Bloom filter (CIBF).

CBF

A CBF works similar to a Bloom filter, but instead of being a bit vector, it stores integers. The insertion of an element works by adding the count of an element to the integer at the positions the hash functions of that element are pointing to [132] (see Figure 3.22 for an example). In case the count is inserted to a position that already stores a non-zero value, the maximum value between the stored and the count value of the current element is stored.

Querying an element works by taking the minimum of the values that the hash functions of the element are pointing to. The minimum is picked, because other

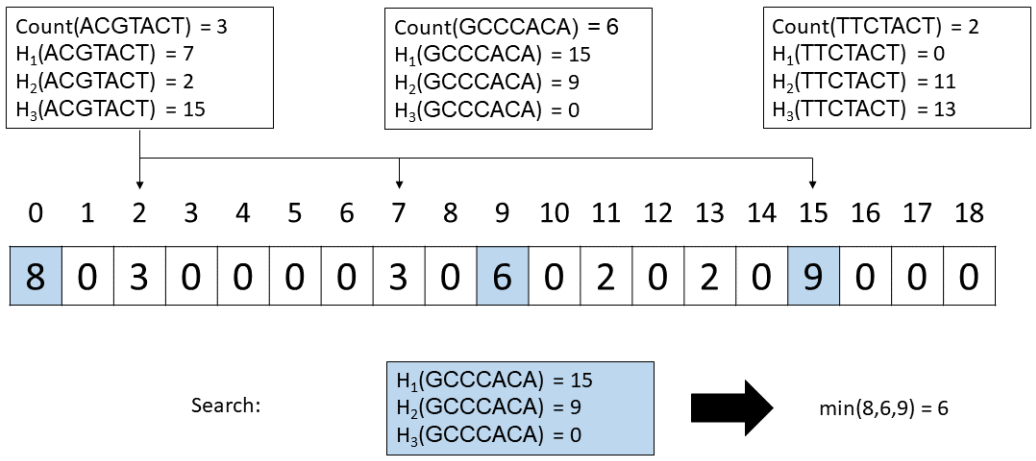


Figure 3.22: **Example of a CBF.** Three elements 'ACGTACT', 'GCCCACA' and 'TTCTACT' are inserted in an empty CBF, thereby adding their counts at the positions that their hash functions are indicating. When then searching for the inserted element 'GCCCACA', the minimum of the found values is then reported as the count value as the higher numbers stem from other elements.

inserted elements might share one or more of the positions in the vector and have increased the count [131].

Similar to Bloom filters these shared hash positions can lead to false positives in CBFs in the sense that the reported count value is higher than the actual one [131].

The size of a CBF compared to a Bloom filter is greater as more than one bit is needed per entry in the vector [129, 132]. The actual size is dependent on the desired integer type one wants to save. Note that there have been some alternative implementation of the CBF to improve further on their space consumption, for example by applying d-Left hashing [130], as these implementation are based on more convoluted ideas, which make it harder or even impossible to combine with the ideas of the IBF, investigating those implementation is out of scope of this work.

Moreover, CBFs are sometimes implemented with a delete function, meaning an element can be removed by decreasing the values at the positions their hash functions are pointing to [130, 133]. This would introduce false negatives [130, 133]. As false negatives should be avoided in the use case covered here, this option is not further taken into account here.

CIBF

The CIBF similar to the IBF combines b CBFs into one vector by interleaving them (see Figure 3.23 for an example). The insertion and query work similar to CBFs. Compressing a CIBF might be harder than compressing an IBF as it does not contain only zeros and ones anymore.

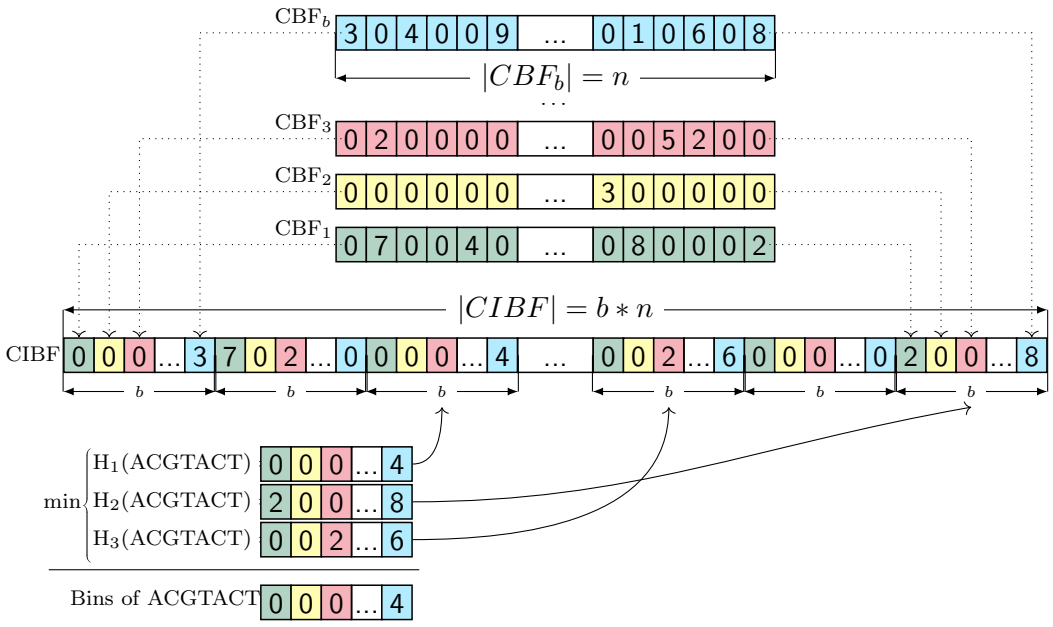


Figure 3.23: **Example of an CIBF.** There are $[b]$ individual CBFs of length n shown, which were interleaved for constructing a CIBF using three hash functions. During a query, here for the example 'ACGTACT', these three hash function will point to positions in the CIBF and to three different sub-vectors. For these sub-vectors for each bin the minimum of all three subvectors is determined for the resulting vector. Here, ACGTACT has a count value of 4 in the last (blue) experiment.

Evaluation of the CIBF for the Needle workflow

The current *Needle* workflow is based on q IBFs storing in each q level elements with a count value in a defined range. 15 levels seem to be sufficient for most use cases [32]. A CIBF replacing these q IBFs could work in two ways.

The first option would be allowing to directly save the counts in the CIBF, thereby abandoning the range approach, which might lead to a higher accuracy. However, by doing so the CIBF would need to save high count values and would need to support an integer type capable of storing at least $2^8 = 256$ numbers. A greater number might be even appropriate, depending on how interested one is in count values above 255. Therefore, at least eight additional bits per entry are needed for a CIBF.

The second option would be to keep the idea of saving only the ranges of count values and not all counts. In this way, each number in a CIBF would relate to a level in the current *Needle* workflow. As 15 levels are a reasonable number, it would be enough, if the CIBF has only four additional bits per entry.

In regards of space consumption, the advantage of the CIBF over q IBFs is that

only one CIBF is necessary. This CIBF however has to store all elements in one CIBF, while the individual IBFs contain only parts of the elements. Given the standard division of the elements in the *Needle* workflow based on the automatic thresholds, the first level will contain half of the elements, while the second level contains a fourth and so on. Therefore, the size of all q IBFs can be estimated by multiplying the size of the first level IBF by two.

The difference of elements stored between the CIBF and the first level IBF is two as well, as the first level stores half of the elements.

Based on equation 3.1, the size of a Bloom filter can be determined by

$$n \approx \frac{-hm}{ln(1 - p_{fpr}^{1/h})}. \tag{3.5}$$

The size of an IBF is this formula multiplied by the number of b bins, while the size of an CIBF is this formula multiplied by b and the number of additional bits for storing the counts (n_{bits}).

$$-h2m/ln(1 - p_{fpr}^{1/h}) \cdot b \cdot n_{bits} < -hm/ln(1 - p_{fpr}^{1/h}) \cdot b \cdot 2$$

$$n_{bits} < 1.$$

Hence, the CIBF consumes more space than the q IBFs and it is not an advantage to use a CIBF.

However, as only one CIBF is necessary, the query could be faster with a CIBF. Moreover, a CIBF would not lead to a multiple testing problem and therefore not need a correction. Also, the first option might be of interest, if a more detailed differentiation between the count values is of interest and might increase the accuracy. In summary, the CIBF has interesting properties and, despite the higher space consumption, might be the appropriate data structure for certain tasks.

3.6.5 Implementation

Needle supports the build directly from sequencing experiment. However, it is recommended to use a preprocessing step.

The preprocessing step consists of determining all minimizers and the number of their occurrence for each sequencing file and storing these information in a temporary file together with the information of how many minimizers are contained in one file. During the preprocessing, the minimizers and their counts are kept in an unordered map and whenever a minimizer is seen the mapped value is increased. However, in most cases only minimizers which have a count value above a certain threshold are of interest. In order to reduce the memory consumption of the preprocessing

step, a second unordered map, which allows only counts up to *uint8* is constructed. Minimizers are kept in this second map until the count value exceeds the threshold, only then they are stored in the first unordered map. Minimizers in the second unordered map are discarded once all minimizers have been seen.

Based on this preprocessing step, the build starts with constructing all q IBFs. The size for each IBF is determined from the average minimizer content for each level and a given false positive rate.

In case of the user-defined thresholds, the minimizer content of each level needs to be determined by going over all minimizers and counting how many to expect per level for each sequencing experiment. In case of the automatic thresholds, the total count of minimizers of one sequencing file is enough to estimate the minimizer content for each level, as it should half with every level due to the construction of the automatic thresholds. Therefore, the construction with automatic thresholds is faster.

If there is no preprocessing step, the size of the IBFs is estimated based on the file size of the sequencing experiments, which is less precise and might lead to too big IBFs.

All constructed IBFs are kept in memory to insert the minimizers in the correct IBF. Alternatively, the IBFs could be loaded one by one to reduce the memory footprint, but this would significantly increase the construction time and therefore was not implemented.

Moreover, the false positive rate is crucial for *Needle*'s query as it is used to correct the number of found minimizers and needs to be known for each sequencing experiment, therefore *Needle* stores a matrix with the false positive rates for each level and each sequencing experiment as the given false positive rate will only hold on average.

3.6.6 Normalization

The results from *Needle* can be normalized by every normalization method that is used for read counts. However, as for *Needle count*, there is no need to correct for read length, because the median is more robust to it than the sum of aligned reads. *Needle* also offers its own normalization method to correct sequencing depth, by dividing each result by the threshold of level 2. This is only applies for the automatic thresholds, as then the thresholds are based on the given sequencing experiments. Level 2 was picked instead of level 1 because level 1 can be influenced by given cutoff values.

3.6.7 Dynamic update

The IBF supports dynamic updates in the form of inserting new elements into specific bins and inserting and deleting complete bins. In *Needle*, the insertion and deletion of complete bins has to happen on each level.

The deletion just resets all bits in a bin to zero, but the bin remains and therefore still takes up space. In theory, one could delete the bin completely by shifting all other bins and then resizing the bitvector. However, the IBF in its current implementation is optimized for 64 bit machines and therefore, the number of bins in an IBF are always a multiple of 64. Thus, such a complete deletion only makes sense, if there are at least 64 empty bins. Furthermore, keeping the deleted bins offers space for new sequencing experiments. For these reasons, the deleted bins are kept.

When new sequencing experiments are added, the IBF is first checked for deleted bins, which are then filled and if there are none or not enough, more bins are added. As the IBF is already created, the size of an individual bin is determined. Therefore, if the new sequencing experiments are significant larger than the ones *Needle* was built on, the false positive rate for these bins will be rather high. There is no option to increase the size of the bins as this would mean all hashes need to be recalculated and for that the original sequencing experiments are needed. In this regard, the HIBF seems promising as then large new sequencing experiments could be split.

3.6.8 Parallelization

The simplest way to parallelize the construction in *Needle* is to go over multiple sequencing experiments at the same time and fill their information into the correct IBFs. If the sequencing experiments are in bins that are stored on a different byte this will not lead to a race condition and therefore, the split of the sequencing experiments is set accordingly.

However, such a simple parallelization for the determination of the minimizers and their counts either in the recommended way by a preprocessing step or during the construction of the IBFs leads to multiple sequencing files loaded into the RAM namely one for each thread, which can easily exceed the available memory due to the size of the sequencing experiments.

Therefore, a more RAM-friendly parallelization option was implemented. In this implementation, the parallelization happens over one sequencing experiment by dividing the sequences over the different threads. Then each thread determines the minimizers and their counts for their sequence batch and after all threads finished their calculation, a single thread merges the result. This merging step slows this parallelization down compared to the simple version, but is less memory intensive.

For this reason both parallelization methods were implemented and the user can decide which option is more suitable.

To show the different performance of these two parallelization methods, the preprocessing step was run for four sequencing experiments with four threads with the simple parallelization and the RAM friendly version on a Linux machine (Debian GNU/Linux 11 (bullseye)) with 1TB RAM and an IAMD EPYC 7702P 64-Core Processor CPU with 64 cores and 256MB L3 cache (see A.3.2 for more details). The four sequencing experiments have a size of approximately 20 GB (gzipped).

	Threads	Time	RAM
simple	1	81	21
	4	18	84
RAM-friendly	4	35	34

Table 3.2: **Comparison of Needle's simple and RAM-friendly parallelization for determining the minimizers and their counts.** The time is given in minutes and the RAM in GB.

As shown in Table 3.2, the simple parallelization method is twice as fast as the RAM-friendly version, but uses more than twice the amount of memory. Both parallelization methods are faster than using only one thread, but have a higher memory consumption. Therefore, both parallelization methods have their merit and its up to the user to decide, which one is the best.

The parallelization for the quantification is implemented in a simplistic way by parallelizing over the query sequences. As a query does not consist of a large collection of sequencing experiments, but rather one file of sequences, the memory footprint of the quantification is small enough to not need a more sophisticated parallelization method.

3.7 Comparison of quantitative indices

The quantitative indices are compared regarding their accuracy and their speed and space performance. As the counting DBG is implemented in *MetaGraph*, *MetaGraph* will from here on refer to the counting DBG not to the non-quantitative version.

Moreover, *Gazelle* is not publicly available and therefore can not be included in the comparison. For all applications, the most recent released version was used, which are *MetaGraph*(v0.3.6), *Needle*(v1.0.2) and *Reindeer*(v1.0.2).

3.7.1 Accuracy

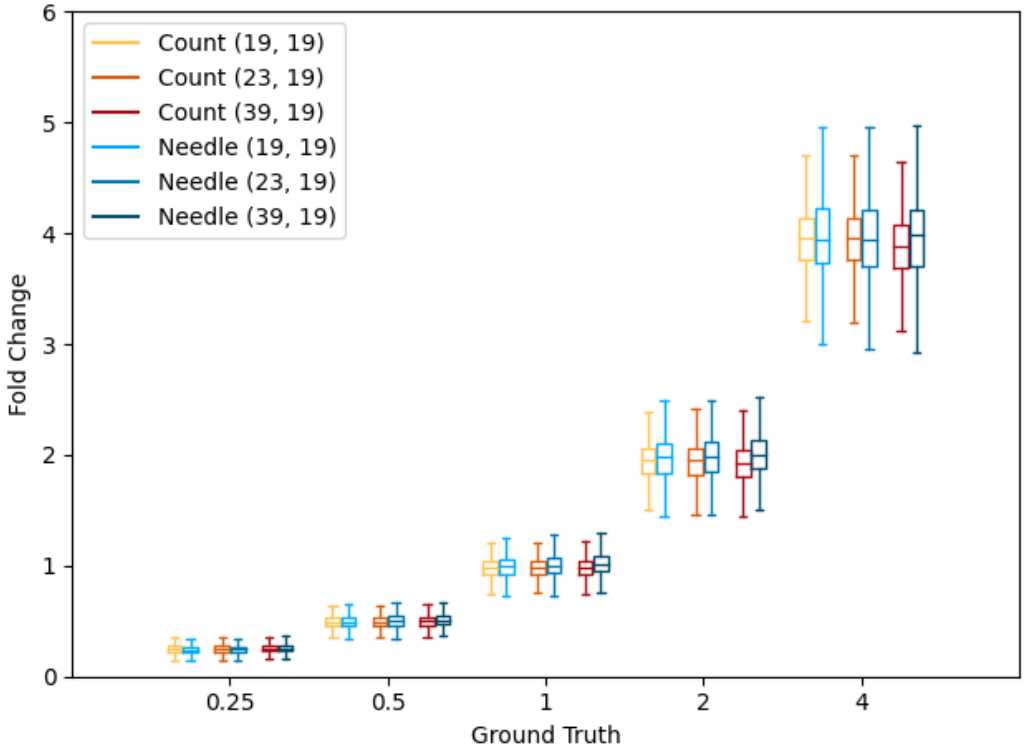
In section 2.3.3, it was shown for *Needle count* that taking the median of the k-mer or minimizer count values has a similar accuracy as more sophisticated approaches. *MetaGraph* and *Reindeer* can determine the exact k-mer count values in a query, therefore, the analysis done for *Needle count* holds for them as well.

However, *Needle* itself only approximates the median. For this reason, the accuracy analyses needs to be repeated for *Needle* for both data sets, the simulated and the real one. *Needle* was tested for two different false positives rates 0.05 and 0.3. The surprisingly high false positive rate of 0.3 was chosen because, as Bingmann et al. pointed out for non-quantitative indices [21], considering multiple minimizers lowers the overall false positive rate and therefore even a false positive rate of 0.3 is reasonable [21].

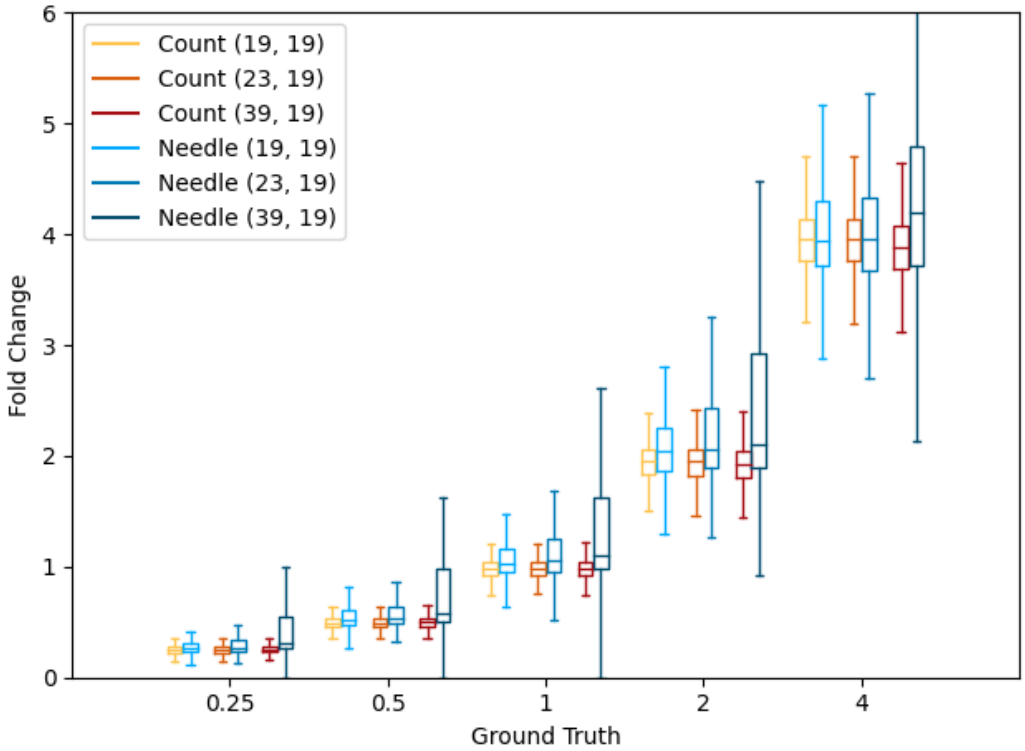
Simulated data

As the simulated data set has coverages ranging between 20 to 80 and differentially expressed genes with fold changes ranging between $\frac{1}{4}$ to 4, *Needle* was built with user-defined thresholds by using 15 thresholds between 5 to 520.

Figures 3.24 and 3.25 show that *Needle* can capture the fold changes for differentially expressed genes and different coverages. There is a difference between *Needle* and *Needle count* that is more prominent with a higher false positive rate. For a false positive rate of 0.05, the differences between the different window sizes seems dismissably small. For a false positive rate of 0.3, the difference between $(19, 19)$- and $(23, 19)$-minimizers is still barely noticeable. Only for a false positive rate of 0.3 and $(39, 19)$-minimizers did the accuracy decrease significantly, which makes sense as the greater the window size the less minimizers are considered and the argument for considering a higher false positive rate weakens. However, even for these parameters, most fold changes are close to the ground truth and therefore, even this combination is a good choice.

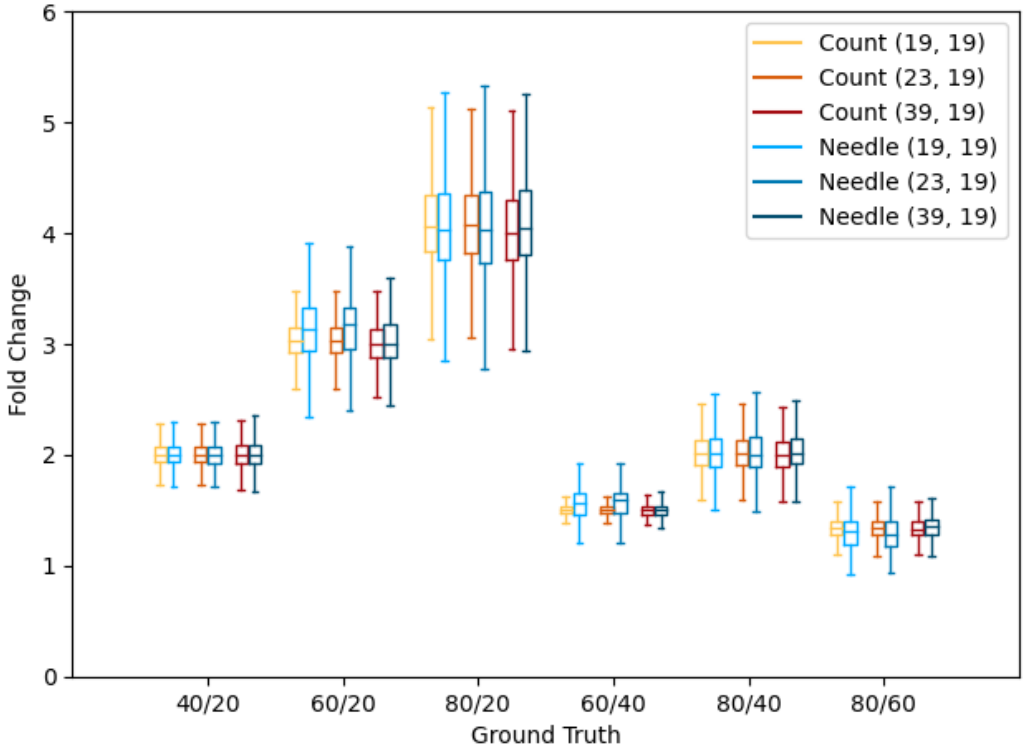


(a) FPR 0.05

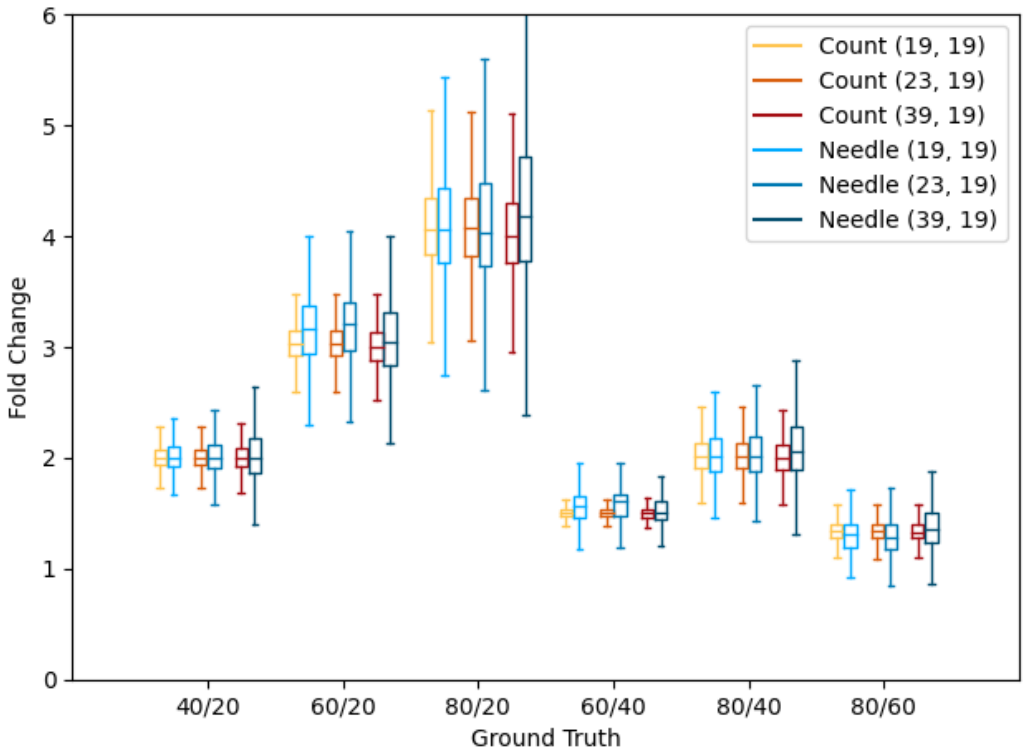


(b) FPR 0.3

Figure 3.24: **Comparison of Needle with Needle Count on differentially expressed genes.** The figure shows the fold change of differentially expressed genes in the simulated data set. *Needle count*, here named Count, and *Needle* use minimizers with the window sizes 19, 23 and 39. *Needle* was used for a false positive rate of 0.05 and 0.3. The x-axis presents the actual fold change that was set during the simulation, the y-axis shows the measured fold change.



(a) FPR 0.05



(b) FPR 0.3

Figure 3.25: **Comparison of Needle with Needle count with different coverages.** The figure shows the fold change of genes expressed within simulated sequencing experiments of different coverages. *Needle count*, here named Count, and *Needle* use minimizers with the window sizes $19, 23$ and 39. *Needle* was used for a false positive rate of 0.05 and 0.3. The x-axis presents the actual expected fold change by dividing the known coverages, for example comparing coverage 40 with 20 should lead to a fold change of $40/20 = 2$. The y-axis shows the measured fold change.

Real data

Needle was built for the SEQC data set with the automatic thresholds for 10 levels for the false positive rates of 0.05 and 0.3. Furthermore, *Needle*'s normalization was tested by using the normalized expression values for the mean square error calculation. For the comparisons with the expression values from the RT-PCR and the microarray, *Needle*'s normalization would not impact the results because the normalization is for comparisons between different sequencing experiments, not for within one.

Table 3.3 shows that the correlations of *Needle* with the expression values from the RT-PCR and the microarray are almost as strong as the ones from *Needle Count*. In most cases, the difference between *Needle* and *Needle Count* is less than one percent. The mean square of errors *Needle* are either the same or almost the same to the ones from *Needle Count* as well.

Similarly to the results for the simulated data set, the combination of $(39, 19)$-minimizers and a false positive rate of 0.3 led to the worst result. But even so, there is still a correlation to the known expression values and therefore, might be the right choice in certain cases.

	SEQC	Microarray	MSE
Count $(19, 19)$	80.6	77.3	0.6
Needle $(19, 19)$	80.6	77.1	0.5
0.3 Needle $(19, 19)$	80.1	76.9	0.5
Count $(23, 19)$	80.6	77.2	0.5
Needle $(23, 19)$	80.5	77.1	0.5
0.3 Needle $(23, 19)$	79.5	75.5	0.6
Count $(39, 19)$	80.6	77.2	0.5
Needle $(39, 19)$	80.4	76.9	0.5
0.3 Needle $(39, 19)$	60.9	58.6	1.3

Table 3.3: **Comparison of Needle with Needle count on the SEQC data set.** *Needle* and *Needle count*, here named just Count, use minimizers with the window sizes $19, 23$ and 39. For *Needle* the false positive rates 0.05 and 0.3 (marked) were considered. The columns SEQC and mircoarray give the Spearman correlation in percent to the RT-PCR quantification and to the miroarray quantification respectively. The column MSE represent the mean square error of the titration monotonicity transcript-wise.

Conclusion

For the simulated data set, *Needle* was tested in a best-case scenario as the expression levels are known during construction, while for the real data set, *Needle* was tested in the more common case, where this information is not known. Both analyses

show that *Needle* can approximate the median quite well and that even greater window sizes or greater false positive rates lead to accurate results. However, not every window size combined with any false positive rate lead to the same accuracy. Therefore, these values should be carefully considered.

Moreover, as *MetaGraph* and *Reindeer* would perform similar to *Needle Count*, they have a higher accuracy than *Needle*, but depending on the chosen parameters *Needle* can be close to their performance and therefore compete with them.

3.7.2 Space and speed

All comparisons were performed on a Linux machine (Debian GNU/Linux 11 (bullseye)) with 1TB RAM and an IAMD EPYC 7702P 64-Core Processor CPU with 64 cores and 256MB L3 cache.

Data

Solomon and Kingsford introduced next to their SBT also a data set that nowadays can be seen as the main benchmark in the field of handling large collections of nucleotide data [8, 97]. The data set consists of $2,568$ human RNA-seq files from blood, breast and brain tissues with different read lengths [97]. Some of the read lengths are quite short for the minimizers that are tested here. Therefore, similar to the benchmark in Raptor [65], a subset of the data set was used for the benchmark by excluding all experiments with an average read length below 50 bp. As such short reads are seldomly relevant. This lead to a data set of $1,742$ sequencing experiments, which takes up around 6 TB of space in gzipped fastq format [32].

Preprocessing

In most benchmarks, the index construction is separated from a preprocessing step (see for example: [32, 97, 110]), in which usually the k-mer or minimizer occurrence is obtained. Lemane et al. criticized this practice as the preprocessing step takes often a long time and thereby slows down construction [111]. Although, Lemane et al. are correct in their assessment, the separation between the index construction and the preprocessing step seems appropriate as often the preprocessing step is calculated by a separated application and not the index application itself. Furthermore, determining k-mer or minimizer occurrences in the fastest and most space efficient way is a separate research topic [134]. Moreover, in theory all indices should be capable of supporting each other's preprocessing step as they do the same. That this is not the case in practice is due to a lack of a standard.

With all this said, the preprocessing is compared for four random sequencing experiments from the $1,742$ to give a rough idea of how the different preprocessing

steps fare.

		Time	RAM
1	KMC3	30	56.3
	MetaGraph	0.4	1.1
	Needle $(21, 21)$	129	84.1
	Needle $(25, 21)$	53	21.5
	Needle $(41, 21)$	27	5.7
4	KMC3	11	71.4
	MetaGraph	0.2	1.1
	bcalm2	260	33.2
	Needle $(21, 21)$	43	223.9
	Needle $(25, 21)$	17	84.3
	Needle $(41, 21)$	8	21.6

Table 3.4: **Comparison of the preprocessing step of different quantitative indices for $k = 21$ on four sequencing experiments.** The numbers on the far left give the number of threads used, for *bcalm2* 2 cores instead of 4 threads are used. *Needle* (w, k) represents *Needle* based on (w, k)-minimizers. Build-Time is given in minutes and RAM is given in GB.

The preprocessing step of the *MetaGraph* is based on *KMC3* [134], *Reindeer*'s is based on *bcalm2* [135] and *Needle*'s preprocessing is part of its implementation. For this comparison, the cutoffs from Table 3.1 have been applied.

The preprocessing step of *Reindeer* is doing more than counting the k-mer occurrences, *bcalm2* determines a DBG for each sequencing experiment. It is questionable to consider this still part of the preprocessing, especially since most other applications, non-quantitative and quantitative, do less in their preprocessing. The first step in a *MetaGraph* construction after the preprocessing is to determine the individual DBGs as well. Therefore, for a fairer comparison this *MetaGraph* build step is also considered here.

Moreover, while *KMC3*, *MetaGraph* and *Needle* allow adjustment in the number of threads to use, *bcalm2* can only adjust the number of cores. Furthermore, *bcalm2* with one core did not manage to process even one sequencing experiment in three hours and was therefore not included in the comparison.

As can be seen in Table 3.4, the *KMC3* has the fastest implementation for counting k-mer occurrences with the smallest memory footprint. This was to be expected as *KMC3* is one of the fastest k-mer counters. However, *Needle* is still reasonable fast and even outperforms *bcalm2*. Moreover, *Needle* is the only preprocessing that allows counting of minimizer occurrences and therefore can not be replaced in *Needle*. Although, the results indicate that the preprocess of *Needle* could be improved for k-mers, if *KMC* files are supported as well.

Construction

For *Reindeer* and *Needle*, the construction is based on one command after preprocessing. For *MetaGraph* multiple commands need to be executed, therefore, the run time of all of these were summed up and the highest memory footprint was reported in Table 3.5. The run time and memory footprint of the individual *MetaGraph* commands can be found in the appendix in Table A.2.

The comparison is based on 21-mers as *MetaGraph* and *Reindeer* used them in their benchmark as well [8, 95]. Both *MetaGraph* and *Reindeer* have an inexact mode named *smooth* and *log* respectively, where the k-mer counts are smoothed over a unitig or log-transformed. These versions were included in the comparison as well.

Needle was built for a false positive rate of 0.05. Moreover, as *Reindeer* has not been updated since the publication of [32] and the analysis here is done on the same machine, the values for *Reindeer* were adopted. The numbers for *Needle* might be slightly different than in [32] as the most recent *Needle* version was applied.

		Time	RAM	Index Size
1	Reindeer log	432	39.4	27.9
	Needle $(21, 21)$	79 (82)	121.1	62.2 (20.1)
	Needle $(25, 21)$	22 (24)	38.4	19.7 (6.9)
	Needle $(41, 21)$	6 (6)	9.4	4.8 (1.7)
4	MetaGraph	916	347.1	20.7
	MetaGraph smooth	791	345.0	9.3
	Reindeer	251	44.1	50.5
	Reindeer log	218	39.4	27.9
	Needle $(21, 21)$	22 (34)	121.1	62.2 (20.1)
	Needle $(25, 21)$	9 (11)	38.4	19.7 (6.9)
	Needle $(41, 21)$	3 (3)	9.4	4.8 (1.7)

Table 3.5: **Comparison of building quantitative indices for $k = 21$ on the large real data set.** *Needle* (w, k) represents *Needle* based on (w, k)-minimizers. The time is given in minutes, RAM and Index size in GB. The values in parentheses are the results when using compressed IBFs. *Needle* was built with a false positive rate of 0.05.

Reindeer in its exact mode led to a library error and is therefore not reported in Table 3.5. However, as it still built with four threads, a comparison with the other applications is still possible [32].

Building *MetaGraph* with one thread took more than 24 hours, after which the build process was stopped, as *MetaGraph* can not compete in its run time with *Needle* or *Reindeer*. However, because the build with four threads was completed, *MetaGraph* can be included in the comparison.

As shown in Table 3.5, *Needle* is the fastest application and creates in its compressed form the smallest index, even when compared to the inexact version of *Reindeer*. Although *MetaGraph*'s inexact version leads to a smaller index than *Needle* for $(21, 21)$-minimizers, *Needle* outperforms that version as well as soon as greater window sizes are considered. *Needle*'s memory footprint is greater than *Reindeer*'s for $(21, 21)$-minimizers, but still at a scale that most modern machines can handle easily.

The index size of *MetaGraph* is surprising as Karasikov et al. [95] reported for the the complete data set of $2,568$ RNA-Seq files index sizes of 21 GB for the exact version and 11 GB for the smoothed version. The index sizes measured here are quite close to these numbers, especially for the exact version, despite considering less sequencing data.

The *MetaGraph* analyses performed here are based on the script that Karasikov et al. [95] provided. After close examination, the only difference that was found between the provided scripts and the commands used here, lies in the usage of *KMC3*. By default, *KMC3* sets all count values above 255 to 255. As this is too restrictive, this value was set to $65,535$ as this is the maximal value *Needle* captures as it is uses the integer type $uint16\_t$. Karasikov et al. [95] on the other hand did not modify the default value of *KMC3*.

Karasikov et al. [95] did not give any reasoning why a maximal count value of 255 is sufficient and considering the sequencing depth of modern sequencing machines this value seems too low. Moreover, *Reindeer* and *Needle* would probably also create smaller indices with such a small maximum. Therefore, the comparison presented here seems to be more accurate.

Furthermore, while *Gazelle* could not be tested due to its unavailability, Zhang et al. analyzed *Gazelle*'s performance on the $2,568$ RNA-Seq files and reported an index size of 45.6 GB for its exact version and 36.7 GB when using less bits for each count value [127]. As here only a subset is used, these values do not easily compare. If the values are multiplied with $\lfloor \frac{1,742}{2,568} = 0.6 \rfloor$, the expected index sizes for the here used data set would be 27.4 GB and 22.0 GB, which would make *Needle* in its compressed form still the smallest index. However, this estimation is rather loose as the sequencing experiments excluded to get the $1,742$ were experiments with a short read length. Comparing the *Reindeer* index sizes reported here with the one Marchet et al. reported for the $2,568$ set leads to a difference of more than 0.8 [32].

According to Zhang et al. *Gazelle* takes more than nine hours to build [127] for all $2,568$ RNA-Seq files. Even if it would take only half the time to process the $1,742$, *Needle* would be still considerable faster. Therefore, it can be concluded that *Needle* outperforms *Gazelle* as well.

On top of *Needle* already outperforming *Gazelle*, *MetaGraph* and *Reindeer* for

$(21, 21)$-minimizers, its real advantage can be seen when applying greater window sizes as the index size can shrink to less than 2 GB and the construction can happen in less than ten minutes. This kind of speed up and compression is something the other indices can not offer. And as shown above, using greater window sizes impacts the accuracy with a false positive rate of 0.05 only slightly.

However, even a false positive rate of 0.3 can lead to meaningful results. Therefore, the index sizes between the false positive rates 0.05 and 0.3 were compared in Table 3.6. The construction time and memory footprint were omitted because the difference was rather small.

		Index Size
0.05	Needle $(21, 21)$	62.2 (20.1)
	Needle $(25, 21)$	19.7 (6.9)
	Needle $(41, 21)$	4.8 (1.7)
0.3	Needle $(21, 21)$	8.9 (11.7)
	Needle $(25, 21)$	2.8 (3.5)
	Needle $(41, 21)$	0.7 (0.8)

Table 3.6: **Index sizes for Needle constructed with different minimizers and false positive rates.** *Needle* (w, k) represents *Needle* based on (w, k)-minimizers. The index size is given in GB. The values in parentheses are the results when using compressed IBFs. *Needle* was built with a false positive rate of 0.05 and 0.3 as marked with the values on the leftmost column.

As shown in Table 3.6, a false positive rate of 0.3 can reduce the size to less than a 15 percent of the size with a false positive rate of 0.05. With $(41, 21)$-minimizers even an index size of less than one GB can be achieved. Interestingly, the compressed versions take up more space than the uncompressed versions, which is probably due to the fact that IBFs with higher false positive rates have more ones and the compression loses its benefit.

In summary, *Needle* can create small indices fast and because the false positive rate as well as the concrete minimizers, especially the window size, are user defined parameters, *Needle* can be optimized for any given task.

Query

The query performance was tested for 1, 100 and $1,000$ randomly picked transcripts from the human genome. Similar to the comparison for the construction, the values for *Reindeer* were adopted from [32].

Reindeer led to a memory allocation error when run in their log-transformed version, and with four threads to a segmentation fault error. Even after consulting the developers, the issues could not be solved and have been open issues in the

		1	100	1000	RAM
1	MetaGraph	10	69	430	20.9/21.2/21.8
	MetaGraph smooth	11	27	219	9.4/9.7/10.2
	Reindeer	813	897	1,710	80.7
	Reindeer log	>559	>559	>559	67.3
	Needle $(21, 21)$	77 (30)	41 (34)	147 (214)	30.4 (9.7)
	Needle $(25, 21)$	6 (4)	10 (10)	45 (67)	9.6 (2.8)
	Needle $(41, 21)$	4 (1)	3 (3)	13 (20)	2.4 (0.7)
4	MetaGraph	9	59	248	20.9/21.5/21.9
	MetaGraph smooth	5	12	86	9.4/9.9/10.4
	Reindeer	>746	>746	>746	80.7
	Reindeer log	>559	>559	>559	67.3
	Needle $(21, 21)$	30 (15)	33 (21)	54 (67)	30.4 (9.7)
	Needle $(25, 21)$	9 (4)	9 (6)	19 (21)	9.6 (2.8)
	Needle $(41, 21)$	2 (1)	3 (2)	5 (6)	2.4 (0.7)

Table 3.7: **Comparison of querying quantitative indices for $k = 21$ on the large real data set.** *Needle* (w, k) represents *Needle* based on (w, k)-minimizers. Time is given in seconds, RAM and Index size in GB. The values in parentheses are the results when using compressed IBFs. The RAM for *MetaGraph* is given for the queries 1/100/1000, for *Reindeer* and *Needle* the RAM usage was the same for all three queries. In the instances where *Reindeer* resulted in an error, only the index loading times are reported and marked with >.

Reindeer github repository for two years by now[1]. However, in both cases, the query worked in so far as the indices were loaded successfully, as this is always the first step in a *Reindeer* query, therefore the times for the index loading were reported in Table 3.7.

As shown in Table 3.7, *Needle*'s memory footprint is smaller than the complete *Needle* index as only one IBF at a time is loaded into memory. For $(21, 21)$-minimizers *Needle* uses a similar amount of memory in its compressed form as *MetaGraph*, for greater window sizes it has the smallest memory footprint compared to the other applications.

Moreover, *Needle* and *MetaGraph* are a magnitude faster than *Reindeer*. *MetaGraph* is faster than *Needle* with $(21, 21)$-minimizers when querying one transcript. For 100 and $1,000$ transcripts the speed between both applications is rather similar, especially if *MetaGrap smooth* is considered. However, as soon as a greater window size than 21 is taken into account, *Needle* clearly outperforms both *MetaGraph* versions.

The query performance of *Needle* based on a false positive rate of 0.3 is not considered here, as the false positive rate should have no impact on the speed of a

[1]Malloc: `https://github.com/kamimrcht/REINDEER/issues/13`, Segfault: `https://github.com/kamimrcht/REINDEER/issues/14` (both accessed: 14. August 2023).

query, outside of creating smaller indices, which are faster to load.

Insertion and deletion

Needle is the only quantitative index that offers dynamic updates by inserting or deleting sequencing experiments. In order to show the performance of both features, four sequencing experiments are deleted from and inserted into the indices built above for a false positive rate of 0.05.

		Time	RAM	Index Size
Delete	Needle $(21, 21)$	173	60.7	62.2
	Needle $(25, 21)$	43	19.2	19.7
	Needle $(41, 21)$	11	4.7	4.8
Insert	Needle $(21, 21)$	425	121.2	62.2
	Needle $(25, 21)$	243	38.4	19.7
	Needle $(41, 21)$	31	9.4	4.8

Table 3.8: **Performance of inserting and deleting into a Needle index.** *Needle* (w, k) represents *Needle* based on (w, k)-minimizers. The time is given in seconds, RAM and Index size in GB.

As can be seen in Table 3.8 the index size was not changed through the insertion or deletion. Deletions can never change the index size as the space is kept for future insertions. The insertions did not change the index size as there were empty bins left as $1,742$ is not a multiple of 64.

Both deletions and insertions, are reasonably fast. Therefore, *Needle*'s dynamic update options are practicable.

3.7.3 Conclusion

In contrast to other quantitative indices, *Needle* does not store the minimizer occurrence directly, instead it stores in which occurrence range the minimizer falls. Therefore, the median of minimizer occurrences as expression value can only be approximated. However, despite the approximation *Needle* is capable of capturing fold changes due to different coverages or differential expression and correlates with expression values obtained by microarray and RT-PCR.

Moreover, *Needle* outperforms the other quantitative indices in speed and size. Due to the application of minimizers, *Needle* can reduce the index size drastically while still being accurate. Alternatively, the index size can be minimized by increasing the false positive rate, which for some minimizers barely has an impact on the accuracy. Because of this *Needle* is more flexible than other applications as its

setting can be adapted for a given task. However, more research is necessary to understand the relationship between the window size and the false positive rate.

Furthermore, the idea of *Needle* is not dependent on the IBF. There is the potential to replace it with the HIBF as soon as dynamic updates are possible, but any other non-quantitative index would work with the idea here presented. Therefore, as the research of non-quantitative indices advances, so can *Needle*.

Chapter 4

Utilizing large collections of sequencing data

Indexing large collections of sequencing data opens up new possibilities for research. In order to show and to prove the advantages of actually utilizing large amounts of sequencing data, three different data sets are investigated. Firstly, *Needle* is applied to find tissue-specific genes in a data set consisting of three different tissue types. Secondly, a newly annotated mouse transcript is searched for in a set of randomly choosen mouse sequencing experiments and lastly, *Needle* will be used to find cancer signatures. As all three analyses have a different goal and topic, this shows that *Needle* can be applied for a wide range of research questions.

4.1 Determining genes of interest

Studying genes and their functions takes up a huge part of the biological and medical research field. As genes are often involved in numerous phenotypes and seldomly act alone, it is often of interest to study a network of genes [136]. Determining genes of interest is thereby a tedious task and often based on a small set of sequencing experiments. With *Needle*, interesting genes can be determined within in minutes for a wide range of phenotypes.

As a proof of concept the large data set from the previous chapter consisting of $1,742$ RNA-Seq experiments is analyzed further. The data set contains 809 RNA-Seq experiments from blood, 336 from brain tissue and 597 from breast tissue. As the *Needle* index was already built, the index was used as is without a change of parameters. All $103,155$ protein-coding transcripts in the human genome were quantified. With one thread the quantification took less than twenty minutes and with 4 threads less than six minutes [32].

Then the gene expression of each human gene was estimated by taking the mean

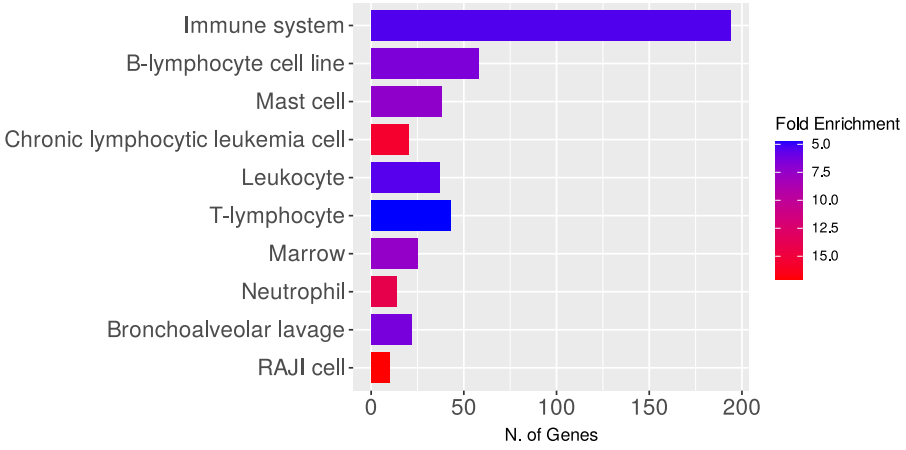
of all its transcripts expression values. Based on these values a differential expression analysis was performed with *DESeq2*. All over-expressed genes between one tissue and the other two tissue types with at least a fold change of 4 were determined. Note that this analysis is slightly different than the one in [32], where the differential gene expression was based on simple t-test instead of *DESeq2*.

For the blood category $1,052$ genes were found overexpressed, for the brain tissue $1,085$ genes and for the breast tissue 270 genes. Figure 4.1 shows the gene ontology for each gene set obtained with *ShinyGo* [137] for the *TISSUES 2.0* database [138].

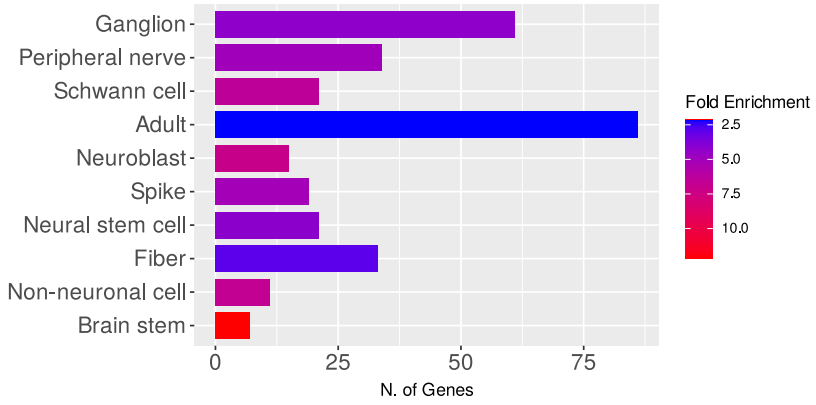
For the blood category, all found types are related to blood. Almost all tissue types for the brain are brain related. The large numbers of genes for *Adult* might be surprising, but adult brains differ from children brains [139] and the large numbers associated with adulthood could indicate that the samples in the data set stem from adult brains.

For the breast tissue, at a first glance there are some surprising associations, such as the association with *Mouth*, *Bladder* and several epithelium types. However, these tissues have been shown to be associated to breast tissue. For example, some breast cancer genes were shown to be involved in an oral cancer [140], bladder cancer subtypes showed similarities to breast cancer subtypes at a molecular level [141] and the epithelium in breast tissue is a target of multiple research projects due to its unique characteristic of developing after child birth [142]. Specifically, *Myoepithelium* is known for its role in breast tissue [143].

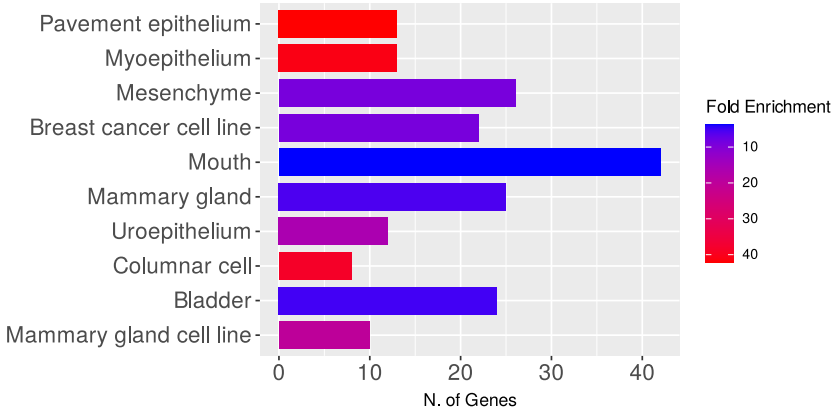
In summary, with a rather simple differential expression analysis on three tissue types that were only considered together because the data set already existed, genes of interest could be determined and shown to be actually meaningful as they described the underlying tissue. Thereby proving that analyses with *Needle* on large collections of sequencing experiments are sufficient.



(a) Blood



(b) Brain



(c) Breast

Figure 4.1: **Gene ontology analysis for genes found differentially expressed in blood, brain and breast tissue of 1742 RNA-seq experiments.** The colors show the fold enrichment, the tissues are sorted by their false discovery rate.

4.2 Finding a newly annotated mouse transcript

One major benefit of utilizing large collections of sequencing data is that new findings can be either validated quickly or further researched. Gjaltema et al. discovered a

previously unannotated transcript in the mouse genome called *Xert*, which regulates and enhances the developmental gene *Xist* [144]. Due to the lack of a database allowing searches for the transcript, it is not trivial to determine whether *Xert* can be found under different conditions as well.

4.2.1 Data and building the Needle index

In order to prove that *Needle* could be applied for such a task, $1,024$ mouse RNA-Seq files were used to build a *Needle* index. These $1,024$ RNA-Seq files included twelve RNA-Seq files from the original study by Gjaltema et al., which were from different time points for two female embryonic mouse stem cells, where for one line one X chromosome was lost (XO). There are two replicates for each time point. All other sequencing experiments were randomly selected from all publicly available mouse RNA-Seq experiments on the Sequence Read Archive, for which information about the gender was given and the average read length was at least 150 *bp*. The gender information was of interest because *Xert* is expected to be differential expressed between female and male mice as there is a difference between having two X chromosomes or one [144].

These $1,024$ RNA-Seq files took roughly 3 TB of space in gzipped form. A compressed *Needle* index was built with $(23, 19)$-minimizer, a cutoff value of 1 and a false positive rate of 0.05. This led to an index size of 75.2 GB, so roughly 2.5 percent of the original size. As mentioned above for the cancer signatures, a smaller index size could be achieved by using a greater window size, a higher false positive rate or a greater cutoff. The cutoff of 1 was used to avoid parameter tuning and to account for low counts because *Xert* appeared in the original study in a low quantity, therefore these might be informative.

The expression values from *Needle* were normalized in a similar way as in the original study by summing up the expression of all transcripts and then dividing the expression values of the *Xert* transcripts by the sum and multiplying the result with 10^7. This is similar to the TPM normalization of the original study without considering the transcript length as *Needle's* expression value are less prone to be influenced by the transcript length due to the usage of the median. The multiplication with 10^7 instead of 10^6 was done to achieve a nicer scaling.

4.2.2 Finding Xert in the original data

Figure 4.2 shows the normalized expression based on *Needle* of *Xert* for the sequencing experiments of the original study. In four sequencing experiments from the original study, there can be no traces found of *Xert* (Day 0 in Figure 4.2), which also can not be found with Needle. As in the original study, there is an expression

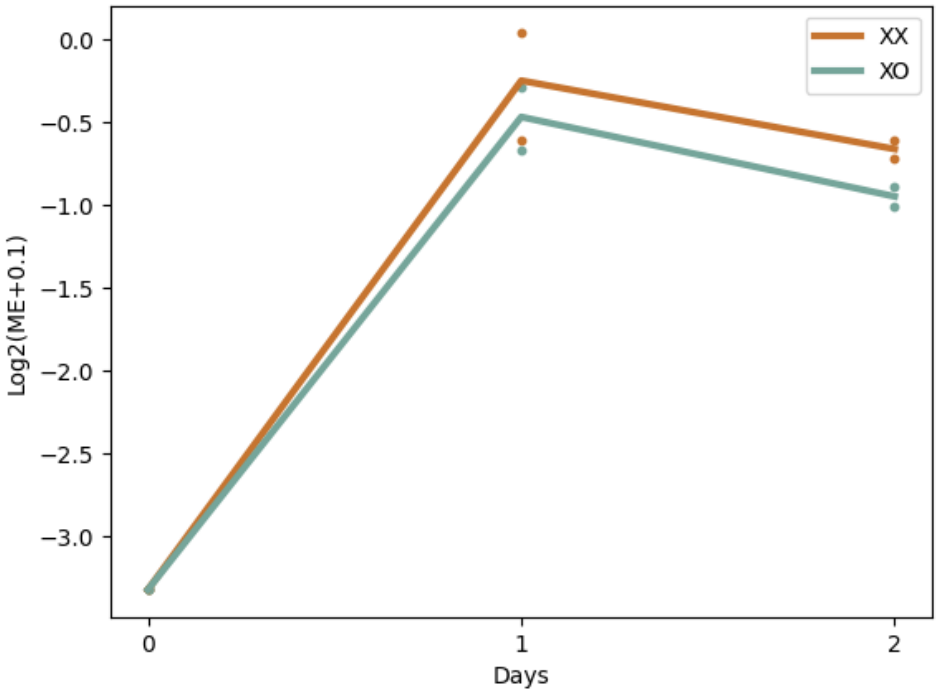


Figure 4.2: **Expression of Xert for different days of differentiation for XX and XO cell line.** The expression of *Xert* according to the ME from *Needle* for the XX cell line (both X chromosomes present) and the XO cell line for different time points after differentiation. The ME was normalized similar to the original study. 0.1 was added to avoid taking the logarithm of 0. The line is based on the mean between the different replicates. The dots mark the expression of the replicates.

peak at day 2 and then a slow decline [144].

Moreover, the difference in Figure 4.2 between the two cell lines is rather small and does not seem significant. Even this is in accordance with the original study (see Figure 3B in [144]), where the difference between the two cell lines was not significant for the RNA-Seq experiments, but for other sequencing techniques like TT-Seq [144].

4.2.3 Finding Xert in mouse randomly picked RNA-Seqs

In 855 of the 1,024 sequencing experiment, there was no trace of *Xert* found, leaving 169 sequencing experiments potentially containing *Xert*. 16 of these 169 experiments are the ones from the original study as discussed above. The 10 sequencing experiments with the highest normalized ME values were determined excluding the sequencing experiments from the original study, because these sequencing experiments are already known to contain *Xert*. Then these top ten sequencing experiments were aligned to the mouse genome including *Xert* with the tool *STAR* [145].

Experiment	# Reads	Project Topics
SRR1205854	51	Cornea
SRR851653	7	de novo DNA methylation in germ cells
SRR5279965	4	cancer secreted exosomes during cancer progression
SRR11241390	2	ADAR1 bound to DNA in neurons
SRR12170201	2	FXR1 during spermiogenesis
SRR8589508	51	cerebral cortex during postnatal development
SRR12179942	10	anxiety and depression during aging
SRR4015305	1	immunized mouse spleen

Table 4.1: **Top ten sequencing Experiments found by Needle to contain Xert.** The number of matched reads was determined with *STAR*. The project topics are based on the abstract given in the sequence read archive and are supposed to give the most informative description.

In 8 out of the 10 sequencing experiments *Xert* was found (see Table 4.1). For some of these found experiments, the number of matched reads to *Xert* is rather small and probably not worth further investigations. The sequencing experiment SRR4015305 with only one matched read has to be seen as a false positive as well as the used cutoff of 1 should have prevented finding it. Therefore, the false positive rate for these 10 sequencing experiments is 30 %.

The sequencing experiment SRR1205854 is from sequencing cornea tissue. *Xist* has been found to play a role in the cornea related disease keratoconus [146]. Therefore, finding *Xist* in presumably healthy cornea tissue indicates that studying the impact of *Xert* in keratoconus or even in general in cornea tissue might lead to new insights.

Furthermore, the X chromosome has a strong impact on the development of the brain [147]. Therefore, the findings of *Xert* in brain tissues in SRR8589508, SRR12179942 and SRR11241390 are of interest and the associated projects could be used for further research.

Moreover, *Xist* is also known to be expressed in germ cells [148] and *Xist* is an oncogene in certain cancer types [149, 150]. Therefore, finding *Xert* in germ cells (SRR851653 and SRR12170201) and cancer tissue (SRR5279965) is not surprising.

For 162 of the $1,024$ sequencing experiments, there is a sequencing experiment from the same project of the opposite gender. As mentioned above, one would expect a stronger expression of *Xert* in female samples than male samples, therefore all projects were determined with a stronger *Xert* expression for the female sample than the male sample. Afterwards, these experiments were validated by aligning them to the mouse genome.

11 projects were determined, so 22 sequencing experiments. For 5 projects *Xert* could be found with a higher number of matched reads for the female sample than

the male one (see Table 4.2). Similar to the sequencing experiment in Table 4.1 with only one matched read, SRR13859593 should be probably seen as a false positive as well. Therefore, there are 7 false positives and the false positive rate is around 64 %.

Experiment	# Reads	Project Topics
SRR10513178	12	melanomas after UVB
SRR10513179	7	treatment
SRR13615756	3	mice hearts infected
SRR13615766	2	with T1L reovirus
SRR13859593	1	Intestinal tissue
SRR13859586	0	following Pichia colonization
SRR15722532	3	islet b cell dysfunction
SRR15722533	0	
SRR9127111	21	aging in mice, skin tissue
SRR9126727	1	aging in mice, liver tissue

Table 4.2: **Sequencing Experiments with a higher gene expression of Xert for the female sample than the male sample according to Needle.** The number of matched reads was determined with *STAR*. The project topics are based on the abstract given in the sequence read archive and are supposed to give the most informative description. The female sample is given first for each project.

As already mentioned, *Xist* is known as a oncogene in certain cancers, but it also acts as a regulator in melanoma [151]. Therefore, the findings of *Xert* in SRR10513178 and SRR10513179 seem promising that *Xert* plays a similar role. Moreover, it is interesting that *Xert* can be found in presumably healthy skin tissue (SRR9127111) as well.

Furthermore, pancreatic islet biology is influenced by sex [152] the different numbers of matched reads for SRR15722532 and SRR15722533 could indicate that this is partly due to the involvement of *Xert*.

4.2.4 Conclusion

In general, the numbers of matched reads found here is rather low and while there might be some connection to the X chromosome or even *Xist*, that does not necessarily mean *Xert* is a contributing factor. However, these findings are still interesting as they give a reason to look further into the mentioned topics. Further research could validate *Xert*'s actual role in these conditions by either reusing existing experiment or starting new ones. As *Xert* can be better detected with TT-Seq [144], new sequencing experiments might help in clarifying *Xert*'s role.

The high number of false positives, especially in the second analysis, might seem concerning at the first glance. However, the used parameters were rather loose.

A cutoff of only 1 is small and allowed identification of sequencing experiments with small numbers of matched reads. Depending on whether one is interested in sequencing experiments with such a low number of matched reads or not, an increase of the cutoff could have resulted in a smaller false positive rate. Also, the results from *Needle* could have undergone a post processing, excluding experiments with a low quantification. The fact that the number of false positives was lower for the top ten results compared to the analysis based on gender shows that the difference in *Needle*'s expression values matters.

However, even if there were no possibilities to improve the false positive rate, the shown results are helpful. *Needle* was capable of reducing $1,024$ potentially interesting sequencing experiments to only 169 sequencing experiments. This is an enormous achievement, especially considering the fact that the parameters were set rather loose.

4.3 Cancer signatures

Cancer is a widespread, heterogeneous disease that is the cause of nearly ten million deaths worldwide and therefore has been a research focus for decades [153]. One research aim is to find significant characteristics of cancer that can in the best case help to develop a treatment. These characteristics are often dependent on a specific cancer type due to its heterogeneity and can be described with the broad term cancer signatures. Cancer signatures thereby include mutations from substitutions to rearrangements [154], biallelic mutations [155] or gene signatures [156]. Gene signatures consists of at least one gene with an unique expression profile for a certain condition that should be the result of an altered biological or biochemical processes [156].

A wide range of gene signatures exist for different kind of cancers. Breast cancer as the most common cancer type [153] has been the focus of many research projects, which is why there are multiple breast cancer gene signatures [157–159]. Some of them are even used to guide medical treatment by giving a prediction whether a patient will profit from a chemotherapy or not [157, 159–161].

If gene signatures can be detected in a *Needle* index, this would open up new research possibilities. If a *Needle* index over large collections of sequencing data exists and a new gene signature is determined based on one data set, it could be easily verified with *Needle* for a larger data set. Moreover, the *Needle* index can be used in the beginning, to find a new gene signature by determining differentially expressed genes between different samples.

Furthermore, even a use case in direct patient care could be applicable by comparing the sequencing data to all stored data in a *Needle* index and thereby clas-

sifying the patient in either a certain risk group or detecting the specific cancer type.

In order for any of these applications to work, the *Needle* index needs to be capable of detecting gene signatures. The analyses above showed that the quantification of *Needle* is meaningful and make the recognition of gene signatures with *Needle* look promising. However, to further validate the capabilities of the *Needle* index, two different breast cancer gene signatures were analyzed on a data set consisting of different breast cancer samples. The aim of these analyses is to show that first of all, the *Needle* quantification verifies these known gene signatures and secondly, that a clustering based on these genes is possible as these would prove that classifications with *Needle* are attainable.

4.3.1 Data and building the Needle index

A publicly accessible data set (GEO accession number: GSE58135) consisting of 168 sequencing experiments in total was used to search for gene signatures with Needle. All sequencing experiments were obtained with Illumina HiSeq 2000 and consist of paired-end reads of length 50 *bp*. 28 experiments originate from breast cancer cell lines, 42 from triple negative breast cancer primary tumors (TNBC), another 42 from estrogen receptor positive (ER+) breast cancer primary tumors (ERBC), 21 from non-malignant breast tissue next to the TNBC, 30 from non-malignant breast tissue next to the ERBC and 5 from non-cancer tissue obtained from a reduction of mammoplasty [162]. The sequencing files in fastq format take up roughly 1 TB of disk space.

A compressed *Needle* index was built with $(24, 20)$-minimizer, a cutoff value of 1 and a false positive rate of 0.05. This led to an index of size 16.7 GB, so a size of less than 2 percent of the original size. Moreover, a smaller index could be achieved by using a greater window size, a higher false positive rate or a greater cutoff. The chosen cutoff is particularly small, as only minimizers that occur once are disregarded. The cutoff was set to 1 to avoid parameter tuning.

For quantifying genes, all their transcript sequences were quantified with *Needle* and then the mean of all the transcript values was picked as the gene expression value.

4.3.2 Finding TNBC signature

The first gene signature analyzed is a gene signature distinguishing TNBC from ERBC and consists of 40 genes. Nine of these genes were found overexpressed in TNBC and the other 31 genes were underexpressed in TNBC compared to ERBC [163].

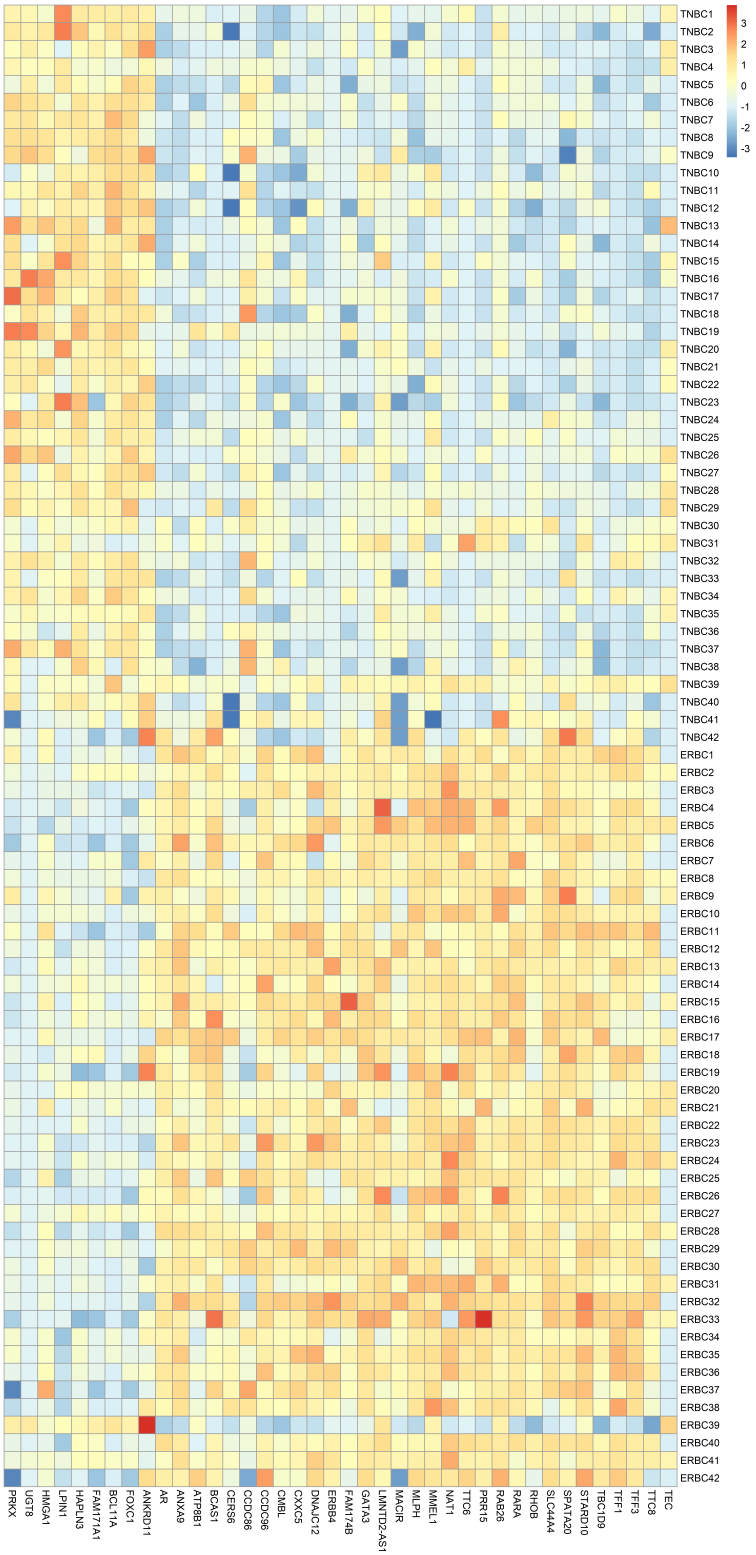


Figure 4.3: **Heatmap of 40 gene signature for ERBC and TNBC samples in data set.** The heatmap was built on top of the Z-Scores. The first nine genes are the ones that should be overexpressed in TNBC and the other 31 genes are the underexpressed genes.

A differential expression analysis was performed with *DESeq2* and found 38 of the 40 genes differential expressed between the TNBC and ERBC samples of the GSE58135 data set. As can be seen in Figure 4.3, generally the overexpressed genes are in the TNBC samples and the underexpressed genes are in the TNBC samples. One sample of the ERBC with the name "ERBC39" does not fit well as its gene expression shows more similarity to the TNBC samples. There might be a chance that this specific sample was mislabeled.

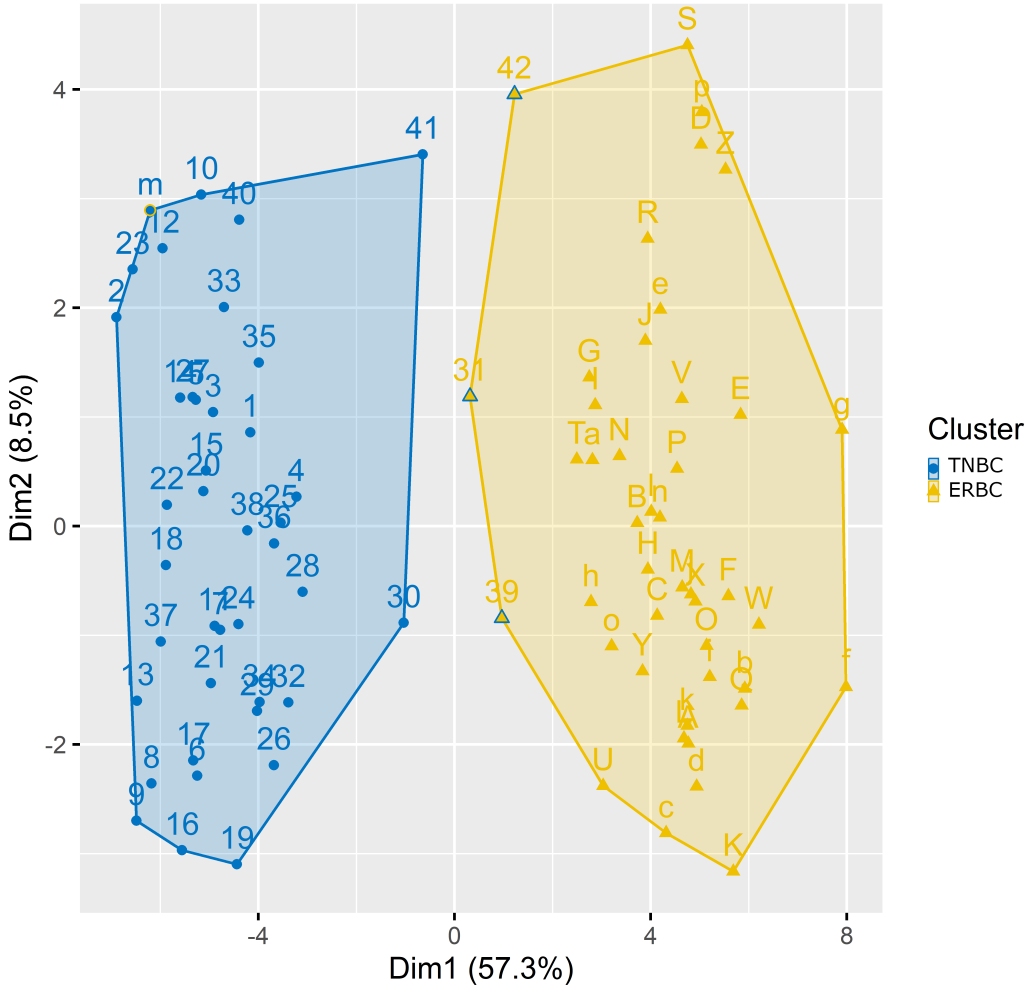


Figure 4.4: **PCA of the clustering of the TNBC and ERBC samples in the data set for the gene signature.** The TNBC samples are named from 1 to 42, the ERBC samples are named with letters. Samples that were misclassified by the clustering algorithm (31,39,42 and m) are marked by being circled in the color of the cluster they are supposed to be in.

Furthermore, the TNBC and ERBC samples were clustered with a K-Medoids algorithm based on the Z-Score values of the gene expression values given by Needle. As can be seen in Figure 4.4, the two sample groups are distinguishable from another.

Only four of the 84 samples are considered to be in the wrong cluster. One of these samples (the one named "m" in Figure 4.4) is the "ERBC39" sample already pointed out above, which might be a mislabeled sample.

4.3.3 Finding NAT signature

The second gene signature analyzed is a gene signature consisting of 18 genes that was found to distinguish normal tissue adjacent to tumor tissue (NAT) from tumor tissue and normal healthy tissue. Aran et al. identified 18 genes overexpressed in NATs compared to tumor and healthy tissue [164].

In the here used data set there are two different types of NATs, one adjacent to ERBC and one adjacent to TNBC. Here, a differential expression analysis was performed with *DESeq2* between the NATs and their respective tumor tissue and between the NATs and the healthy samples.

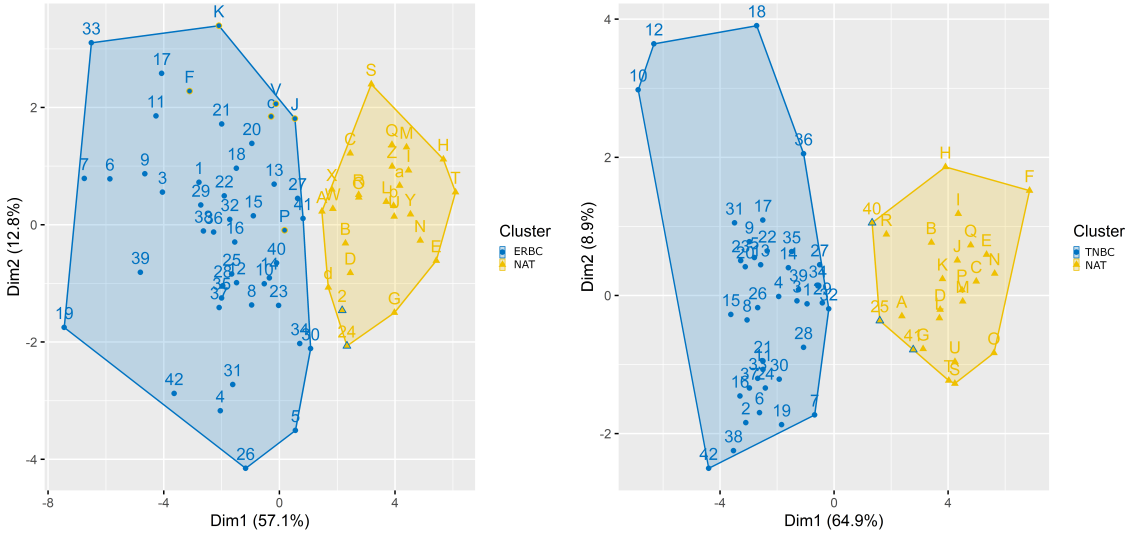
For the analysis with the healthy samples none of the genes were significantly differentially expressed. As the healthy samples consist of only five samples, this sample size might be to small for an analysis. Therefore, the focus of this analysis was on the comparison between the tumor samples and the NAT samples.

For the comparison between the ERBC samples and its NAT samples, 15 out of the 18 genes were significantly differentially expressed and for the comparison between the TNBC and its NAT samples all 18 were differentially expressed.

As with the TNBC gene signature, a clustering based on K-Medoids was performed. The gene signature led to a good clustering for both examples as can be seen in Figure 4.5. In the clustering of the ERBC tissues, eight samples out of 72 were misclassified and on the clustering of the TNBC tissues, three samples out of 63 were misclassified.

4.3.4 Conclusion

The two analyses show that the *Needle* index can be used to verify existing gene signatures, which also indicates the index could be used to determine new ones. Moreover, while the clustering is not perfect, it is promising enough to investigate further the possibility of a clinical use of the *Needle* index.



(a) ERBC (b) TNBC

Figure 4.5: **PCA of the clustering of the NAT samples with their respective cancer samples in the data set for the gene signature.** The cancer samples are named from 1 to 42, the NAT samples are named with letters. Samples that were misclassified by the clustering algorithm (2,24,F,J,K,P,V,c for ERBC and 25,40,41 for TNBC) are marked by being circled in the color of the cluster they are supposed to be in.

Chapter 5

Conclusion

In light of the ever increasing amount of sequencing data, new methods to store and search through this data became necessary because established and already well-known strategies can not handle the sheer amount of data. Therefore, alignment-free methods gained the attention of the research community.

As was shown here, a direct approach of determining gene expression solely on the median of a gene's k-mer occurrences, works surprisingly well in capturing the underlying information. The same holds for a representative set of k-mers like minimizers. Alternative approaches to k-mers and minimizers would also work in theory, but based on the analyses done here, the choice of (representative) submer method is expected to have a rather small impact on the gene expression analysis.

However, in order to actually manage large collections of sequencing experiments, an index that stores this data needs to be built. Here, *Needle* was presented, which outperforms existing alternatives and offers more features like dynamic updates and a better handle on the trade-off between space and accuracy.

There are multiple possibilities to utilize the existing sequencing data, three different possibilities were shown here, proving the usefulness of storing sequencing data to begin with. Furthermore, as the research of utilizing large collections of sequencing data is rather new, it can be expected that new analyses will be developed to learn as much as possible from the data. One possible development could be the usage of machine learning based directly on quantitative indices like *Needle*. For example, the IBFs *Needle* is based on could be used as the input to a machine learning algorithm, thereby allowing analyses directly on the stored sequencing data.

In summary, it can be stated that quantitative indices like *Needle* resolve one major bottleneck of bioinformatics by utilizing the large amount of existing sequencing data and it remains to be seen, how this can advance biological and biomedical research concretely.

References

[1] J. C. Venter *et al.*, "The sequence of the human genome," *Science (New York, N.Y.)*, vol. 291, no. 5507, pp. 1304–1351, 2001. DOI: 10.1126/science.1058040.

[2] L. Hood and L. Rowen, "The human genome project: Big science transforms biology and medicine," *Genome Medicine*, vol. 5, no. 9, p. 79, 2013. DOI: 10.1186/gm483.

[3] K. A. Wetterstrand, *The cost of sequencing a human genome*, 2019. [Online]. Available: https://www.genome.gov/about-genomics/fact-sheets/Sequencing-Human-Genome-cost, (Accessed: 3. July 2023).

[4] H. Armitage, *Fastest dna sequencing technique helps undiagnosed patients find answers in mere hours*, 2022. [Online]. Available: https://med.stanford.edu/news/all-news/2022/01/dna-sequencing-technique.html, (Accessed: 3. July 2023).

[5] D. L. Wheeler *et al.*, "Database resources of the national center for biotechnology information," *Nucleic acids research*, vol. 36, no. Database issue, pp. D13–21, 2008. DOI: 10.1093/nar/gkm1000.

[6] E. W. Sayers *et al.*, "Genbank 2023 update," *Nucleic acids research*, vol. 51, no. D1, pp. D141–D144, 2023. DOI: 10.1093/nar/gkac1012.

[7] K. Katz, O. Shutov, R. Lapoint, M. Kimelman, J. R. Brister, and C. O'Sullivan, "The Sequence Read Archive: a decade more of explosive growth," *Nucleic acids research*, vol. 50, no. D1, pp. D387–D390, 2022. DOI: 10.1093/nar/gkab1053.

[8] C. Marchet, Z. Iqbal, D. Gautheret, M. Salson, and R. Chikhi, "REINDEER: efficient indexing of k-mer presence and abundance in sequencing datasets," *Bioinformatics*, vol. 36, pp. i177–i185, 2020. DOI: 10.1093/bioinformatics/btaa487.

[9] GenomeAsia100K Consortium, "The genomeasia 100k project enables genetic discoveries across asia," *Nature*, vol. 576, no. 7785, pp. 106–111, 2019. DOI: 10.1038/s41586-019-1793-z.

[10] J. Kaiser, *Nih's huge all of us genes and health study releases first 100,000 genomes*, 2022. [Online]. Available: https://www.science.org/content/article/nih-s-huge-all-us-genes-and-health-study-releases-first-100-000-genomes, (Accessed: 4. July 2023).

[11] K. Levi, M. Rynge, E. Abeysinghe, and R. A. Edwards, "Searching the sequence read archive using jetstream and wrangler," in *Proceedings of the Practice and Experience on Advanced Research Computing*, S. Sanielevici, Ed., Pittsburgh PA USA: ACM, 2018, pp. 1–7. DOI: 10.1145/3219104.3229278.

[12] P. Suwinski, C. Ong, M. H. T. Ling, Y. M. Poh, A. M. Khan, and H. S. Ong, "Advancing personalized medicine through the application of whole exome sequencing and big data analytics," *Frontiers in Genetics*, vol. 10, p. 49, 2019. DOI: 10.3389/fgene.2019.00049.

[13] T. Ching *et al.*, "Opportunities and obstacles for deep learning in biology and medicine," *Journal of the Royal Society, Interface*, vol. 15, no. 141, 2018. DOI: 10.1098/rsif.2017.0387.

[14] G. Kucherov, "Evolution of biosequence search algorithms: A brief survey," *Bioinformatics (Oxford, England)*, vol. 35, no. 19, pp. 3547–3552, 2019. DOI: 10.1093/bioinformatics/btz272.

[15] The Cancer Genome Atlas (TCGA) Research Network, "Comprehensive genomic characterization defines human glioblastoma genes and core pathways," *Nature*, vol. 455, no. 7216, pp. 1061–1068, 2008. DOI: 10.1038/nature07385.

[16] K. Tomczak, P. Czerwińska, and M. Wiznerowicz, "The cancer genome atlas (tcga): An immeasurable source of knowledge," *Contemporary oncology (Poznan, Poland)*, vol. 19, no. 1A, A68–77, 2015. DOI: 10.5114/wo.2014.47136.

[17] G. F. Gao *et al.*, "Before and after: Comparison of legacy and harmonized tcga genomic data commons' data," *Cell Systems*, vol. 9, no. 1, 24–34.e10, 2019. DOI: 10.1016/j.cels.2019.06.006.

[18] M. Burrows and D. L. Wheeler, "A block sorting lossless data compression algorithm," *Technical Report*, no. 124, 1994.

[19] P. Ferragina and G. Manzini, "Indexing compressed text," *Journal of the ACM*, vol. 52, no. 4, pp. 552–581, 2005. DOI: 10.1145/1082036.1082039.

[20] E. Ukkonen, "On-line construction of suffix trees," *Algorithmica*, vol. 14, no. 3, pp. 249–260, 1995. DOI: 10.1007/BF01206331.

[21] T. Bingmann, P. Bradley, F. Gauger, and Z. Iqbal, "COBS: a Compact Bit-Sliced Signature Index," in *26th International Conference on String Processing and Information Retrieval (SPIRE)*, ser. LNCS, Springer, 2019, pp. 285–303.

[22] N. D. Magar *et al.*, "Gene expression and transcriptome sequencing: Basics, analysis, advances," in *Gene Expression*, F. Uchiumi, Ed., IntechOpen, 2022. DOI: 10.5772/intechopen.105929.

[23] S. A. Byron, K. R. van Keuren-Jensen, D. M. Engelthaler, J. D. Carpten, and D. W. Craig, "Translating rna sequencing into clinical diagnostics: Opportunities and challenges," *Nature Reviews Genetics*, vol. 17, no. 5, pp. 257–271, 2016. DOI: 10.1038/nrg.2016.10.

[24] A. Negi, A. Shukla, A. Jaiswar, J. Shrinet, and R. S. Jasrotia, "Chapter 6 - applications and challenges of microarray and rna-sequencing," in *Bioinformatics*, D. B. Singh and R. K. Pathak, Eds., Academic Press, 2021, pp. 91–103. DOI: 10.1016/B978-0-323-89775-4.00016-X.

[25] R. Stark, M. Grzelak, and J. Hadfield, "Rna sequencing: The teenage years," *Nature Reviews Genetics*, vol. 20, no. 11, pp. 631–656, 2019. DOI: 10.1038/s41576-019-0150-2.

[26] B. A. Adewale, "Will long-read sequencing technologies replace short-read sequencing technologies in the next 10 years?" *African Journal of Laboratory Medicine*, vol. 9, no. 1, p. 1340, 2020. DOI: 10.4102/ajlm.v9i1.1340.

[27] A. Conesa *et al.*, "A survey of best practices for RNA-seq data analysis," *Genome Biology*, vol. 17, p. 13, 2016. DOI: 10.1186/s13059-016-0881-8.

[28] C. Evans, J. Hardin, and D. M. Stoebel, "Selecting between-sample rna-seq normalization methods from the perspective of their assumptions," *Briefings in Bioinformatics*, vol. 19, no. 5, pp. 776–792, 2018. DOI: 10.1093/bib/bbx008.

[29] S. Vinga and J. Almeida, "Alignment-free sequence comparison-a review," *Bioinformatics*, vol. 19, no. 4, pp. 513–523, 2003. DOI: 10.1093/bioinformatics/btg005.

[30] G. Holley, R. Wittler, and J. Stoye, "Bloom Filter Trie: an alignment-free and reference-free data structure for pan-genome storage," *Algorithms for molecular biology : AMB*, vol. 11, no. 1, p. 3, 2016. DOI: 10.1186/s13015-016-0066-8.

[31] B. Morgenstern, B. Zhu, S. Horwege, and C. A. Leimeister, "Estimating evolutionary distances between genomic sequences from spaced-word matches," *Algorithms for Molecular Biology*, vol. 10, no. 1, p. 5, 2015. DOI: 10.1186/s13015-015-0032-x.

[32] M. Darvish, E. Seiler, S. Mehringer, R. Rahn, and K. Reinert, "Needle: A fast and space-efficient prefilter for estimating the quantification of very large collections of expression experiments," *Bioinformatics (Oxford, England)*, 2022. DOI: 10.1093/bioinformatics/btac492.

[33] J. Ren *et al.*, "Alignment-Free Sequence Analysis and Applications," *Annual review of biomedical data science*, vol. 1, pp. 93–114, 2018. DOI: 10.1146/annurev-biodatasci-080917-013431.

[34] A. Zielezinski, S. Vinga, J. Almeida, and W. M. Karlowski, "Alignment-free sequence comparison: Benefits, applications, and tools," *Genome Biology*, vol. 18, no. 1, p. 186, 2017. DOI: 10.1186/s13059-017-1319-7.

[35] R. Patro, S. M. Mount, and C. Kingsford, "Sailfish enables alignment-free isoform quantification from RNA-seq reads using lightweight algorithms," *Nature Biotechnology*, vol. 32, no. 5, pp. 462–464, 2014. DOI: 10.1038/nbt.2862.

[36] Z. Zhang and W. Wang, "Rna-Skim: A rapid method for RNA-Seq quantification at transcript level," *Bioinformatics (Oxford, England)*, vol. 30, no. 12, pp. i283–i292, 2014. DOI: 10.1093/bioinformatics/btu288.

[37] N. L. Bray, H. Pimentel, P. Melsted, and L. Pachter, "Near-optimal probabilistic RNA-seq quantification," *Nature Biotechnology*, vol. 34, no. 5, pp. 525–527, 2016. DOI: 10.1038/nbt.3519.

[38] C. J.-T. Ju, R. Li, Z. Wu, J.-Y. Jiang, Z. Yang, and W. Wang, "Fleximer: Accurate Quantification of RNA-Seq via Variable-Length k-mers," in *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics (ACM-BCB '17)*, New York, NY, USA: Association for Computing Machinery, 2017, pp. 263–272. DOI: 10.1145/3107411.3107444.

[39] R. Patro, G. Duggal, M. I. Love, R. A. Irizarry, and C. Kingsford, "Salmon provides fast and bias-aware quantification of transcript expression," *Nature Methods*, vol. 14, no. 4, pp. 417–419, 2017. DOI: 10.1038/nmeth.4197.

[40] R. Edgar, "Syncmers are more sensitive than minimizers for selecting conserved k-mers in biological sequences," *PeerJ*, vol. 9, e10805, 2021. DOI: 10.7717/peerj.10805.

[41] K. Sahlin, "Effective sequence similarity detection with strobemers," *Genome Research*, vol. 31, no. 11, pp. 2080–2094, 2021. DOI: 10.1101/gr.275648.121.

[42] K. Sahlin, T. Baudeau, B. Cazaux, and C. Marchet, "A survey of mapping algorithms in the long-reads era," *bioRxiv*, 2022. DOI: 10.1101/2022.05.21.492932.

[43] M. S. Fujimoto, C. A. Lyman, and M. J. Clement, "Kcollections: A Fast and Efficient Library for K-mers," in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2020, pp. 193–198. DOI: 10.1109/IPDPSW50202.2020.00041.

[44] J. Shaw and Y. W. Yu, "Theory of local k-mer selection with applications to long-read alignment," *Bioinformatics (Oxford, England)*, vol. 38, no. 20, pp. 4659–4669, 2022. DOI: 10.1093/bioinformatics/btab790.

[45] K. Břinda, M. Sykulski, and G. Kucherov, "Spaced seeds improve k-mer-based metagenomic classification," *Bioinformatics (Oxford, England)*, vol. 31, no. 22, pp. 3584–3592, 2015. DOI: 10.1093/bioinformatics/btv419.

[46] D. E. Wood, J. Lu, and B. Langmead, "Improved metagenomic analysis with Kraken 2," *Genome Biology*, vol. 20, no. 1, p. 257, 2019. DOI: 10.1186/s13059-019-1891-0.

[47] H. Mohamadi, J. Chu, B. P. Vandervalk, and I. Birol, "ntHash: recursive nucleotide hashing," *Bioinformatics (Oxford, England)*, vol. 32, no. 22, pp. 3492–3494, 2016. DOI: 10.1093/bioinformatics/btw397.

[48] S. Burkhardt and J. Kärkkäinen, "Better Filtering with Gapped q-Grams," *Fundamenta Informaticae*, vol. 56, no. 1–2, pp. 51–70, 2003.

[49] B. D. Maier and K. Sahlin, "Entropy predicts sensitivity of pseudorandom seeds," *Genome Research*, 2023. DOI: 10.1101/gr.277645.123.

[50] K. Sahlin, *Evaluation of methods for constructing randstrobes*, 2022. [Online]. Available: https://github.com/ksahlin/strobemers/tree/main/randstrobe_implementations, (Accessed: 23. May 2023).

[51] M. Darvish, *Comparison between different methods to simplify sequence data*, 2023. [Online]. Available: https://github.com/seqan/minions, (Accessed: 2. June 2023).

[52] K. Reinert *et al.*, "The SeqAn C++ template library for efficient sequence analysis: A resource for programmers," *Journal of Biotechnology*, vol. 261, pp. 157–168, 2017. DOI: 10.1016/j.jbiotec.2017.07.017.

[53] F. Mölder *et al.*, "Sustainable data analysis with Snakemake," *F1000Research*, vol. 10, p. 33, 2021. DOI: 10.12688/f1000research.29032.2.

[54] K. Sahlin, "Strobealign: Flexible seed size enables ultra-fast and accurate read alignment," *Genome Biology*, vol. 23, no. 1, p. 260, 2022. DOI: 10.1186/s13059-022-02831-7.

[55] L. Noé, "Best hits of 11110110111: Model-free selection and parameter-free sensitivity calculation of spaced seeds," *Algorithms for molecular biology : AMB*, vol. 12, p. 1, 2017. DOI: 10.1186/s13015-017-0092-1.

[56] L. Zhang, "Superiority and Complexity of the Spaced Seeds," in *Encyclopedia of Algorithms*, Springer, Berlin, Heidelberg, 2015, pp. 1–5. DOI: 10.1007/978-3-642-27848-8_803-1.

[57] J. Chu *et al.*, "Mismatch-tolerant, alignment-free sequence classification using multiple spaced seeds and multiindex Bloom filters," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 117, no. 29, pp. 16 961–16 968, 2020. DOI: 10.1073/pnas.1903436117.

[58] M. Holtgrewe, "Mason? A Read Simulator for Second Generation Sequencing Data," *Technical Report FU Berlin*, 2010.

[59] A. B. Carvalho, E. G. Dupim, and G. Goldstein, "Improved assembly of noisy long reads by k-mer validation," *Genome Research*, vol. 26, no. 12, pp. 1710–1720, 2016. DOI: 10.1101/gr.209247.116.

[60] W. Kaisers, H. Schwender, and H. Schaal, "Hierarchical clustering of dna k-mer counts in rnaseq fastq files identifies sample heterogeneities," *International Journal of Molecular Sciences*, vol. 19, no. 11, 2018. DOI: 10.3390/ijms19113687.

[61] C. Pockrandt, M. Alzamel, C. S. Iliopoulos, and K. Reinert, "Genmap: Ultra-fast computation of genome mappability," *Bioinformatics (Oxford, England)*, vol. 36, no. 12, pp. 3687–3692, 2020. DOI: 10.1093/bioinformatics/btaa222.

[62] A. Blanca, R. S. Harris, D. Koslicki, and P. Medvedev, "The Statistics of k-mers from a Sequence Undergoing a Simple Mutation Process Without Spurious Matches," *Journal of Computational Biology*, vol. 29, no. 2, pp. 155–168, 2022. DOI: 10.1089/cmb.2021.0431.

[63] N. Stoler and A. Nekrutenko, "Sequencing error profiles of Illumina sequencing instruments," *NAR Genomics and Bioinformatics*, vol. 3, no. 1, lqab019, 2021. DOI: 10.1093/nargab/lqab019.

[64] M. Schirmer, R. D'Amore, U. Z. Ijaz, N. Hall, and C. Quince, "Illumina error profiles: Resolving fine-scale variation in metagenomic sequencing data," *BMC Bioinformatics*, vol. 17, no. 1, p. 125, 2016. DOI: 10.1186/s12859-016-0976-y.

[65] E. Seiler, S. Mehringer, M. Darvish, E. Turc, and K. Reinert, "Raptor: A fast and space-efficient pre-filter for querying very large collections of nucleotide sequences," *iScience*, vol. 24, no. 7, 2021. DOI: 10.1016/j.isci.2021.102782.

[66] T. H. Dadi *et al.*, "DREAM-Yara: an exact read mapper for very large databases with short update time," *Bioinformatics (Oxford, England)*, vol. 34, no. 17, pp. i766–i772, 2018. DOI: 10.1093/bioinformatics/bty567.

[67] V. C. Piro, T. H. Dadi, E. Seiler, K. Reinert, and B. Y. Renard, "ganon: precise metagenomics classification against large and up-to-date sets of reference sequences," *Bioinformatics (Oxford, England)*, vol. 36, no. Suppl_1, pp. i12–i20, 2020. DOI: 10.1093/bioinformatics/btaa458.

[68] S. L. Amarasinghe, S. Su, X. Dong, L. Zappia, M. E. Ritchie, and Q. Gouil, "Opportunities and challenges in long-read sequencing data analysis," *Genome Biology*, vol. 21, no. 1, p. 30, 2020. DOI: 10.1186/s13059-020-1935-5.

[69] A. Dutta, D. Pellow, and R. Shamir, "Parameterized syncmer schemes improve long-read mapping," *PLOS Computational Biology*, vol. 18, no. 10, e1010638, 2022. DOI: 10.1371/journal.pcbi.1010638.

[70] G. Marçais, D. Pellow, D. Bork, Y. Orenstein, R. Shamir, and C. Kingsford, "Improving the performance of minimizers and winnowing schemes," *Bioinformatics*, vol. 33, no. 14, pp. i110–i117, 2017. DOI: 10.1093/bioinformatics/btx235.

[71] M. Roberts, W. Hayes, B. R. Hunt, S. M. Mount, and J. A. Yorke, "Reducing storage requirements for biological sequence comparison," *Bioinformatics*, vol. 20, no. 18, pp. 3363–3369, 2004. DOI: 10.1093/bioinformatics/bth408.

[72] S. Schleimer, D. Wilkerson, and A. Aiken, "Winnowing: Local algorithms for document fingerprinting," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA: Association for Computing Machinery, 2003, pp. 76–85.

[73] H. Zheng, C. Kingsford, and G. Marçais, "Improved design and analysis of practical minimizers," *Bioinformatics (Oxford, England)*, vol. 36, no. Suppl_1, pp. i119–i127, 2020. DOI: 10.1093/bioinformatics/btaa472.

[74] M. Hoang, H. Zheng, and C. Kingsford, "DeepMinimizer: A Differentiable Framework for Optimizing Sequence-Specific Minimizer Schemes," in *Research in Computational Molecular Biology*, I. Pe'er, Ed., Cham: Springer International Publishing, 2022, pp. 52–69.

[75] C.-S. Chin and A. Khalak, "Human genome assembly in 100 minutes," *bioRxiv*, p. 705 616, 2019. DOI: 10.1101/705616.

[76] K. Sahlin and P. Medvedev, "Error correction enables use of oxford nanopore technology for reference-free transcriptome analysis," *Nature Communications*, vol. 12, no. 1, p. 2, 2021. DOI: 10.1038/s41467-020-20340-8.

[77] C. Jain *et al.*, "Weighted minimizer sampling improves long read mapping," *Bioinformatics (Oxford, England)*, vol. 36, no. Suppl_1, pp. i111–i118, 2020. DOI: 10.1093/bioinformatics/btaa435.

[78] B. Ekim, B. Berger, and Y. Orenstein, "A Randomized Parallel Algorithm for Efficiently Finding Near-Optimal Universal Hitting Sets," in *Research in Computational Molecular Biology*, R. Schwartz, Ed., Cham: Springer International Publishing, 2020, pp. 37–53.

[79] Y. Orenstein, D. Pellow, G. Marçais, R. Shamir, and C. Kingsford, "Designing small universal k-mer hitting sets for improved analysis of high-throughput sequencing," *PLOS Computational Biology*, vol. 13, no. 10, e1005777, 2017. DOI: 10.1371/journal.pcbi.1005777.

[80] D. Pellow *et al.*, "Efficient minimizer orders for large values of k using minimum decycling sets," *bioRxiv*, 2022. DOI: 10.1101/2022.10.18.512682.

[81] H. Zheng, C. Kingsford, and G. Marçais, "Sequence-specific minimizers via polar sets," *Bioinformatics (Oxford, England)*, vol. 37, no. Suppl_1, pp. i187–i195, 2021. DOI: 10.1093/bioinformatics/btab313.

[82] M. C. Frith, L. Noé, and G. Kucherov, "Minimally-overlapping words for sequence similarity search," *Bioinformatics (Oxford, England)*, vol. 36, no. 22-23, pp. 5344–5350, 2020. DOI: 10.1093/bioinformatics/btaa1054.

[83] M. C. Frith, J. Shaw, and J. L. Spouge, "How to optimally sample a sequence for rapid analysis," *Bioinformatics (Oxford, England)*, vol. 39, no. 2, 2023. DOI: 10.1093/bioinformatics/btad057.

[84] A. Srivastava, H. Sarkar, N. Gupta, and R. Patro, "RapMap: a rapid, sensitive and accurate tool for mapping RNA-seq reads to transcriptomes," *Bioinformatics*, vol. 32, no. 12, pp. i192–i200, 2016. DOI: 10.1093/bioinformatics/btw277.

[85] L. Pachter, *How not to perform a differential expression analysis (or science)*, 2017. [Online]. Available: https://liorpachter.wordpress.com/tag/pseudoalignment/, (Accessed: 9. January 2022).

[86] A. C. Frazee, A. E. Jaffe, B. Langmead, and J. T. Leek, "Polyester: Simulating rna-seq datasets with differential transcript expression," *Bioinformatics*, vol. 31, no. 17, pp. 2778–2784, 2015. DOI: 10.1093/bioinformatics/btv272.

[87] SEQC/MAQC-III Consortium, "A comprehensive assessment of rna-seq accuracy, reproducibility and information content by the sequencing quality control consortium," *Nature Biotechnology*, vol. 32, no. 9, pp. 903–914, 2014. DOI: 10.1038/nbt.2957.

[88] Y. Zhao *et al.*, "TPM, FPKM, or Normalized Counts? A Comparative Study of Quantification Measures for the Analysis of RNA-seq Data from the NCI Patient-Derived Models Repository," *Journal of Translational Medicine*, vol. 19, no. 1, p. 269, 2021. DOI: 10.1186/s12967-021-02936-w.

[89] J. H. Hong, Y. H. Ko, and K. Kang, "Rna-seq data of invasive ductal carcinoma and adjacent normal tissues from a korean patient with breast cancer," *Data in Brief*, vol. 18, pp. 736–739, 2018. DOI: 10.1016/j.dib.2018.03.079.

[90] Y. Yang, H.-L. Liu, and Y.-J. Liu, "A Novel Five-Gene Signature Related to Clinical Outcome and Immune Microenvironment in Breast Cancer," *Frontiers in Genetics*, vol. 13, p. 912 125, 2022. DOI: 10.3389/fgene.2022.912125.

[91] M. Almutairy and E. Torng, "Comparing fixed sampling with minimizer sampling when using k-mer indexes to find maximal exact matches," *PLOS ONE*, vol. 13, no. 2, e0189960, 2018. DOI: 10.1371/journal.pone.0189960.

[92] J. Burgin *et al.*, "The european nucleotide archive in 2022," *Nucleic acids research*, vol. 51, no. D1, pp. D121–D125, 2023. DOI: 10.1093/nar/gkac1051.

[93] M. Arita, I. Karsch-Mizrachi, and G. Cochrane, "The international nucleotide sequence database collaboration," *Nucleic acids research*, vol. 49, no. D1, pp. D121–D124, 2021. DOI: 10.1093/nar/gkaa967.

[94] X. Su, G. Jing, Y. Zhang, and S. Wu, "Method development for cross-study microbiome data mining: Challenges and opportunities," *Computational and Structural Biotechnology Journal*, vol. 18, pp. 2075–2080, 2020. DOI: 10.1016/j.csbj.2020.07.020.

[95] M. Karasikov, H. Mustafa, G. Rätsch, and A. Kahles, "Lossless indexing with counting de Bruijn graphs," *Genome Research*, vol. 32, no. 9, pp. 1754–1764, 2022. DOI: 10.1101/gr.276607.122.

[96] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of molecular biology*, vol. 215, no. 3, pp. 403–410, 1990. DOI: 10.1016/S0022-2836(05)80360-2.

[97] B. Solomon and C. Kingsford, "Fast search of thousands of short-read sequencing experiments," *Nature Biotechnology*, vol. 34, no. 3, pp. 300–302, 2016. DOI: 10.1038/nbt.3442.

[98] M. D. Muggli, B. Alipanahi, and C. Boucher, "Building large updatable colored de Bruijn graphs via merging," *Bioinformatics (Oxford, England)*, vol. 35, no. 14, pp. i51–i60, 2019. DOI: 10.1093/bioinformatics/btz350.

[99] C. Marchet, *Data-structures for sets of k-mer sets: What's new since 2020*, 2023. [Online]. Available: https://kamimrcht.github.io/webpage/sets_kmer_sets.html, (Accessed: 18. July 2023).

[100] C. Marchet, C. Boucher, S. J. Puglisi, P. Medvedev, M. Salson, and R. Chikhi, "Data structures based on k-mers for querying large collections of sequencing data sets," *Genome Research*, vol. 31, no. 1, pp. 1–12, 2021. DOI: 10.1101/gr.260604.119.

[101] M. Karasikov *et al.*, "MetaGraph: Indexing and Analysing Nucleotide Archives at Petabase-scale," *bioRxiv*, 2020. DOI: 10.1101/2020.10.01.322164.

[102] Y. Yu *et al.*, "SeqOthello: querying RNA-seq experiments at scale," *Genome Biology*, vol. 19, no. 1, p. 167, 2018. DOI: 10.1186/s13059-018-1535-9.

[103] C. Marchet and A. Limasset, "Scalable sequence database search using partitioned aggregated bloom comb trees," *Bioinformatics (Oxford, England)*, vol. 39, no. 39 Suppl 1, pp. i252–i259, 2023. DOI: 10.1093/bioinformatics/btad225.

[104] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970. DOI: 10.1145/362686.362692.

[105] N. de Bruijn, "A combinatorial problem," *Indagationes Mathematicae*, no. 49, pp. 758–764, 1946.

[106] I. J. Good, "Normal Recurring Decimals," *Journal of the London Mathematical Society*, vol. s1-21, no. 3, pp. 167–169, 1946. DOI: 10.1112/jlms/s1-21.3.167.

[107] J. T. Simpson and M. Pop, "The theory and practice of genome sequence assembly," *Annual review of genomics and human genetics*, vol. 16, pp. 153–172, 2015. DOI: 10.1146/annurev-genom-090314-050032.

[108] R. Chikhi, J. Holub, and P. Medvedev, "Data Structures to Represent a Set of k -long DNA Sequences," *ACM Computing Surveys*, vol. 54, no. 1, pp. 1–22, 2022. DOI: 10.1145/3445967.

[109] Z. Iqbal, M. Caccamo, I. Turner, P. Flicek, and G. McVean, "De novo assembly and genotyping of variants using colored de Bruijn graphs," *Nature Genetics*, vol. 44, no. 2, pp. 226–232, 2012. DOI: 10.1038/ng.1028.

[110] P. Pandey, F. Almodaresi, M. A. Bender, M. Ferdman, R. Johnson, and R. Patro, "Mantis: A Fast, Small, and Exact Large-Scale Sequence-Search Index," *Cell Systems*, vol. 7, no. 2, 201–207.e4, 2018. DOI: 10.1016/j.cels.2018.05.021.

[111] T. Lemane, P. Medvedev, R. Chikhi, and P. Peterlongo, "kmtricks: efficient and flexible construction of Bloom filters for large sequencing data collections," *Bioinformatics Advances*, vol. 2, no. 1, vbac029, 2022. DOI: 10.1093/bioadv/vbac029.

[112] Y. Yu, D. Belazzougui, C. Qian, and Q. Zhang, "Memory-Efficient and Ultra-Fast Network Lookup and Forwarding Using Othello Hashing," *IEEE/ACM Transactions on Networking*, vol. 26, no. 3, pp. 1151–1164, 2018. DOI: 10.1109/TNET.2018.2820067.

[113] X. Liu, Y. Yu, J. Liu, C. F. Elliott, C. Qian, and J. Liu, "A novel data structure to support ultra-fast taxonomic classification of metagenomic sequences with k-mer signatures," *Bioinformatics (Oxford, England)*, vol. 34, no. 1, pp. 171–178, 2018. DOI: 10.1093/bioinformatics/btx432.

[114] P. Pandey, M. A. Bender, R. Johnson, and R. Patro, "A general-purpose counting filter," in *Proceedings of the 2017 ACM International Conference on Management of Data*, R. Chirkova, Ed., ser. ACM Digital Library, New York, NY: ACM, 2017, pp. 775–787. DOI: 10.1145/3035918.3035963.

[115] F. Almodaresi, P. Pandey, M. Ferdman, R. Johnson, and R. Patro, "An Efficient, Scalable, and Exact Representation of High-Dimensional Color Information Enabled Using de Bruijn Graph Search," *Journal of Computational Biology*, vol. 27, no. 4, pp. 485–499, 2020. DOI: 10.1089/cmb.2019.0322.

[116] F. Almodaresi *et al.*, "An incrementally updatable and scalable system for large-scale sequence search using the Bentley-Saxe transformation," *Bioinformatics (Oxford, England)*, vol. 38, no. 12, pp. 3155–3163, 2022. DOI: 10.1093/bioinformatics/btac142.

[117] G. Holley and P. Melsted, "Bifrost: highly parallel construction and indexing of colored and compacted de Bruijn graphs," *Genome Biology*, vol. 21, no. 1, p. 249, 2020. DOI: 10.1186/s13059-020-02135-8.

[118] Nate, "Sequence bloom trees, part i: Motivation and principles," *Seven Bridges Genomics Inc*, 2017. [Online]. Available: https://www.sevenbridges.com/sequence-bloom-trees-principles/, (Accessed: 9. August 2023).

[119] C. Sun, R. S. Harris, R. Chikhi, and P. Medvedev, "AllSome Sequence Bloom Trees," *Journal of Computational Biology*, vol. 25, no. 5, pp. 467–479, 2018. DOI: 10.1089/cmb.2017.0258.

[120] B. Solomon and C. Kingsford, "Improved Search of Large Transcriptomic Sequencing Databases Using Split Sequence Bloom Trees," *Journal of Computational Biology*, vol. 25, no. 7, pp. 755–765, 2018. DOI: 10.1089/cmb.2017.0265.

[121] R. S. Harris and P. Medvedev, "Improved representation of sequence bloom trees," *Bioinformatics*, vol. 36, no. 3, pp. 721–727, 2020. DOI: 10.1093/bioinformatics/btz662.

[122] C. Marchet and A. Limasset, "Scalable sequence database search using Partitioned Aggregated Bloom Comb-Trees," *bioRxiv*, 2022. DOI: 10.1101/2022.02.11.480089.

[123] P. Bradley, H. C. den Bakker, E. P. C. Rocha, G. McVean, and Z. Iqbal, "Ultrafast search of all deposited bacterial and viral genomic data," *Nature Biotechnology*, vol. 37, no. 2, pp. 152–159, 2019. DOI: 10.1038/s41587-018-0010-1.

[124] S. K. Srikakulam, S. Keller, F. Dabbaghie, R. Bals, and O. V. Kalinina, "MetaProFi: an ultrafast chunked Bloom filter for storing and querying protein and nucleotide sequence data for accurate identification of functionally relevant genetic variants," *Bioinformatics (Oxford, England)*, vol. 39, no. 3, 2023. DOI: 10.1093/bioinformatics/btad101.

[125] G. Gupta *et al.*, "Fast Processing and Querying of 170TB of Genomics Data via a Repeated And Merged BloOm Filter (RAMBO)," in *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21)*, New York, NY, USA: Association for Computing Machinery, 2021, pp. 2226–2234. DOI: 10.1145/3448016.3457333.

[126] S. Mehringer *et al.*, "Hierarchical interleaved bloom filter: Enabling ultrafast, approximate sequence queries," *Genome Biology*, vol. 24, no. 1, p. 131, 2023. DOI: 10.1186/s13059-023-02971-4.

[127] X. Zhang, Y. Yu, C. H. Mok, J. N. MacLeod, and J. Liu, "Gazelle: Transcript abundance query against large-scale rna-seq experiments," in *Proceedings of the 12th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*, ser. BCB '21, New York, NY, USA: Association for Computing Machinery, 2021. DOI: 10.1145/3459930.3469548.

[128] M. Karasikov, H. Mustafa, A. Joudaki, S. Javadzadeh-no, G. Rätsch, and A. Kahles, "Sparse Binary Relation Representations for Genome Graph Annotation," *Journal of Computational Biology*, vol. 27, no. 4, pp. 626–639, 2020. DOI: 10.1089/cmb.2019.0324.

[129] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: A scalable wide-area web cache sharing protocol," *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, pp. 281–293, 2000. DOI: 10.1109/90.851975.

[130] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, "An Improved Construction for Counting Bloom Filters," Springer, Berlin, Heidelberg, 2006, pp. 684–695. DOI: 10.1007/11841036_61.

[131] K. Kim, Y. Jeong, Y. Lee, and S. Lee, "Analysis of Counting Bloom Filters Used for Count Thresholding," *Electronics*, vol. 8, no. 7, p. 779, 2019. DOI: 10.3390/electronics8070779.

[132] M. Mitzenmacher, "Compressed bloom filters," in *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, A. Kshemkalyani, Ed., ser. ACM Conferences, New York, NY: ACM, 2001, pp. 144–150. DOI: 10.1145/383962.384004.

[133] D. Guo, Y. Liu, X. Li, and P. Yang, "False negative problem of counting bloom filter," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 5, pp. 651–664, 2010. DOI: 10.1109/TKDE.2009.209.

[134] M. Kokot, M. Dlugosz, and S. Deorowicz, "Kmc 3: Counting and manipulating k-mer statistics," *Bioinformatics (Oxford, England)*, vol. 33, no. 17, pp. 2759–2761, 2017. DOI: 10.1093/bioinformatics/btx304.

[135] R. Chikhi, A. Limasset, and P. Medvedev, "Compacting de bruijn graphs from sequencing data quickly and in low memory," *Bioinformatics (Oxford, England)*, vol. 32, no. 12, pp. i201–i208, 2016. DOI: 10.1093/bioinformatics/btw279.

[136] The Gene Ontology Consortium, "The gene ontology resource: 20 years and still going strong," *Nucleic acids research*, vol. 47, no. D1, pp. D330–D338, 2019. DOI: 10.1093/nar/gky1055.

[137] S. X. Ge, D. Jung, and R. Yao, "Shinygo: A graphical gene-set enrichment tool for animals and plants," *Bioinformatics (Oxford, England)*, vol. 36, no. 8, pp. 2628–2629, 2020. DOI: 10.1093/bioinformatics/btz931.

[138] O. Palasca, A. Santos, C. Stolte, J. Gorodkin, and L. J. Jensen, "Tissues 2.0: An integrative web resource on mammalian tissue expression," *Database*, vol. 2018, 2018. DOI: 10.1093/database/bay003.

[139] J. Pujol *et al.*, "Differences between the child and adult brain in the local functional structure of the cerebral cortex," *NeuroImage*, vol. 237, p. 118 150, 2021. DOI: 10.1016/j.neuroimage.2021.118150.

[140] T. Z. Parris *et al.*, "Clinical relevance of breast cancer-related genes as potential biomarkers for oral squamous cell carcinoma," *BMC Cancer*, vol. 14, p. 324, 2014. DOI: 10.1186/1471-2407-14-324.

[141] L. Hutchinson, "Genetics: What do breast and bladder cancer have in common?" *Nature Reviews Clinical Oncology*, vol. 11, no. 4, p. 179, 2014. DOI: 10.1038/nrclinonc.2014.33.

[142] P.-A. Vidi, M. J. Bissell, and S. A. Lelièvre, "Three-dimensional culture of human breast epithelial cells: The how and the why," *Methods in Molecular Biology (Clifton, N.J.)*, vol. 945, pp. 193–219, 2013. DOI: 10.1007/978-1-62703-125-7_13.

[143] T. Gudjonsson, M. C. Adriance, M. D. Sternlicht, O. W. Petersen, and M. J. Bissell, "Myoepithelial cells: Their origin and function in breast morphogenesis and neoplasia," *Journal of mammary gland biology and neoplasia*, vol. 10, no. 3, pp. 261–272, 2005. DOI: 10.1007/s10911-005-9586-4.

[144] R. A. F. Gjaltema *et al.*, "Distal and proximal cis-regulatory elements sense x chromosome dosage and developmental state at the xist locus," *Molecular Cell*, vol. 82, no. 1, 190–208.e17, 2022. DOI: 10.1016/j.molcel.2021.11.023.

[145] A. Dobin *et al.*, "Star: Ultrafast universal rna-seq aligner," *Bioinformatics*, vol. 29, no. 1, pp. 15–21, 2013. DOI: 10.1093/bioinformatics/bts635.

[146] R. Tian, L. Wang, H. Zou, M. Song, L. Liu, and H. Zhang, "Role of the xist-mir-181a-col4a1 axis in the development and progression of keratoconus," *Molecular Vision*, vol. 26, pp. 1–13, 2020.

[147] T. T. Mallard *et al.*, "X-chromosome influences on neuroanatomical variation in humans," *Nature Neuroscience*, vol. 24, no. 9, pp. 1216–1224, 2021. DOI: 10.1038/s41593-021-00890-w.

[148] J. R. McCarrey and D. D. Dilworth, "Expression of xist in mouse germ cells correlates with x-chromosome inactivation," *Nature Genetics*, vol. 2, no. 3, pp. 200–203, 1992. DOI: 10.1038/ng1192-200.

[149] J. Yang, M. Qi, X. Fei, X. Wang, and K. Wang, "Long non-coding rna xist: A novel oncogene in multiple cancers," *Molecular Medicine*, vol. 27, no. 1, p. 159, 2021. DOI: 10.1186/s10020-021-00421-0.

[150] W. Han *et al.*, "Pan-cancer analysis of lncrna xist and its potential mechanisms in human cancers," *Heliyon*, vol. 8, no. 10, e10786, 2022. DOI: 10.1016/j.heliyon.2022.e10786.

[151] B. Pan, X. Lin, L. Zhang, W. Hong, and Y. Zhang, "Long noncoding rna x-inactive specific transcript promotes malignant melanoma progression and oxaliplatin resistance," *Melanoma Research*, vol. 29, no. 3, pp. 254–262, 2019. DOI: 10.1097/CMR.0000000000000560.

[152] M. Gannon, R. N. Kulkarni, H. M. Tse, and F. Mauvais-Jarvis, "Sex differences underlying pancreatic islet biology and its dysfunction," *Molecular Metabolism*, vol. 15, pp. 82–91, 2018. DOI: 10.1016/j.molmet.2018.05.017.

[153] H. Sung *et al.*, "Global Cancer Statistics 2020: GLOBOCAN Estimates of Incidence and Mortality Worldwide for 36 Cancers in 185 Countries," *CA: A Cancer Journal for Clinicians*, vol. 71, no. 3, pp. 209–249, 2021. DOI: 10.3322/caac.21660.

[154] L. B. Alexandrov *et al.*, "Signatures of mutational processes in human cancer," *Nature*, vol. 500, no. 7463, pp. 415–421, 2013. DOI: 10.1038/nature12477.

[155] J. Demeulemeester, S. C. Dentro, M. Gerstung, and P. van Loo, "Biallelic mutations in cancer genomes reveal local mutational determinants," *Nature Genetics*, vol. 54, no. 2, pp. 128–133, 2022. DOI: 10.1038/s41588-021-01005-8.

[156] S. Mallik and Z. Zhao, "Identification of gene signatures from RNA-seq data using Pareto-optimal cluster algorithm," *BMC Systems Biology*, vol. 12, no. Suppl 8, p. 126, 2018. DOI: 10.1186/s12918-018-0650-2.

[157] L. Mittempergher *et al.*, "MammaPrint and BluePrint Molecular Diagnostics Using Targeted RNA Next-Generation Sequencing Technology," *The Journal of molecular diagnostics*, vol. 21, no. 5, pp. 808–823, 2019. DOI: 10.1016/j.jmoldx.2019.04.007.

[158] K. Manjang, S. Tripathi, O. Yli-Harja, M. Dehmer, G. Glazko, and F. Emmert-Streib, "Prognostic gene expression signatures of breast cancer are lacking a sensible biological meaning," *Scientific Reports*, vol. 11, no. 1, p. 156, 2021. DOI: 10.1038/s41598-020-79375-y.

[159] X. Zhao, H. Yan, X. Yan, Z. Chen, and R. Zhuo, "A novel prognostic four-gene signature of breast cancer identified by integrated bioinformatics analysis," *Disease Markers*, vol. 2022, p. 5 925 982, 2022. DOI: 10.1155/2022/5925982.

[160] F. Cardoso *et al.*, "70-Gene Signature as an Aid to Treatment Decisions in Early-Stage Breast Cancer," *The New England journal of medicine*, vol. 375, no. 8, pp. 717–729, 2016. DOI: 10.1056/NEJMoa1602253.

[161] I. Krop *et al.*, "Use of Biomarkers to Guide Decisions on Adjuvant Systemic Therapy for Women With Early-Stage Invasive Breast Cancer: American Society of Clinical Oncology Clinical Practice Guideline Focused Update," *Journal of Clinical Oncology*, vol. 35, no. 24, pp. 2838–2847, 2017. DOI: 10.1200/JCO.2017.74.0472.

[162] K. E. Varley *et al.*, "Recurrent read-through fusion transcripts in breast cancer," *Breast Cancer Research and Treatment*, vol. 146, no. 2, pp. 287–297, 2014. DOI: 10.1007/s10549-014-3019-2.

[163] S. K. Santuario-Facio *et al.*, "A New Gene Expression Signature for Triple Negative Breast Cancer Using Frozen Fresh Tissue before Neoadjuvant Chemotherapy," *Molecular medicine (Cambridge, Mass.)*, vol. 23, pp. 101–111, 2017. DOI: 10.2119/molmed.2016.00257.

[164] D. Aran *et al.*, "Comprehensive analysis of normal adjacent to tumor transcriptomes," *Nature Communications*, vol. 8, no. 1, p. 1077, 2017. DOI: 10.1038/s41467-017-01027-z.

[165] C. Marchet, M. Kerbiriou, and A. Limasset, "BLight: efficient exact associative structure for k-mers," *Bioinformatics (Oxford, England)*, vol. 37, no. 18, pp. 2858–2865, 2021. DOI: 10.1093/bioinformatics/btab217.

[166] T. Lemane *et al.*, "Kmindex and ora: Indexing and real-time user-friendly queries in terabytes-sized complex genomic datasets," *bioRxiv*, p. 2023.05.31.543043, 2023. DOI: 10.1101/2023.05.31.543043.

Appendix

A.1 Repositories

The here presented and developed applications *minions* and *Needle* can be found under the following links: (`https://github.com/seqan/minions`) and (`https://github.com/seqan/needle`).

A.2 Alignment-free quantification

A.2.1 Submers

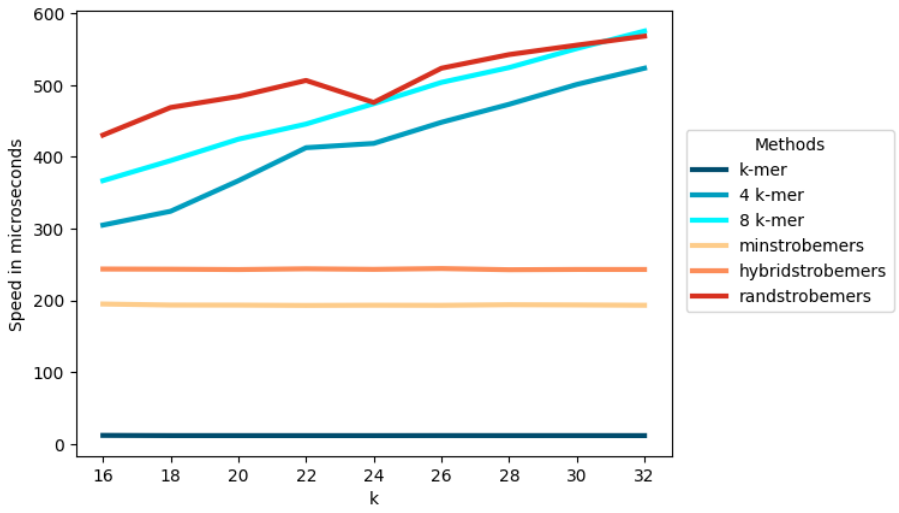
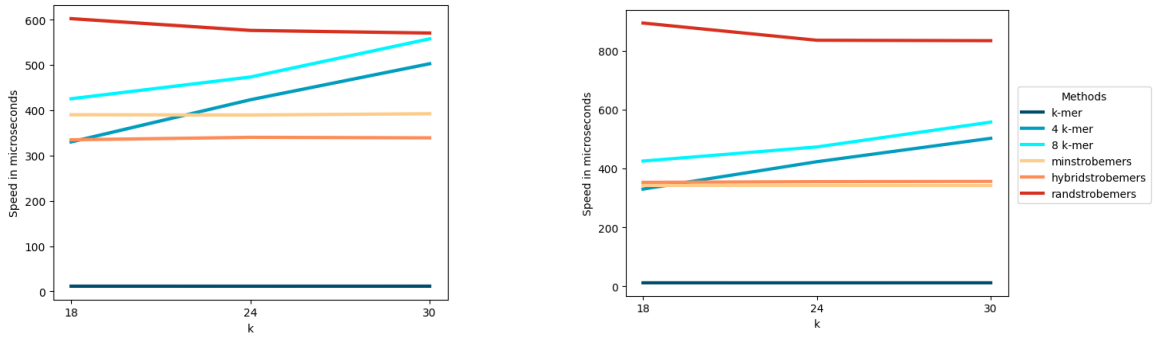


Figure A.1: **Average speed of different submer methods.** 4 and 8 k-mer refer to the gapped $k+4$-mer or $k+8$-mer method with 4 or 8 gaps. The strobemers have an order of 2, so the singular strobes have a length of $k/2$ and they have a window length of $k/2+8$ to ensure a maximal gap of length 8 to the previous strobe.

A.2.2 Representative submers



(a) Strobemers 4

(b) Strobemers 8

Figure A.2: **Average speed of different methods with strobemers of order 3.** 4 and 8 k-mer refer to the gapped $k+4$-mer or $k+8$-mer method with 4 or 8 gaps. The strobemers have an order of 3, so the singular strobes have a length of $k/3$ and they have a window length of $k/2+4$, so that a maximal gap of length 4 to the previous strobe is ensured for Strobemers 4 and a window length of $k/2+8$, a maximal gap of length 8 to the previous strobe is ensured for Strobemers 8.

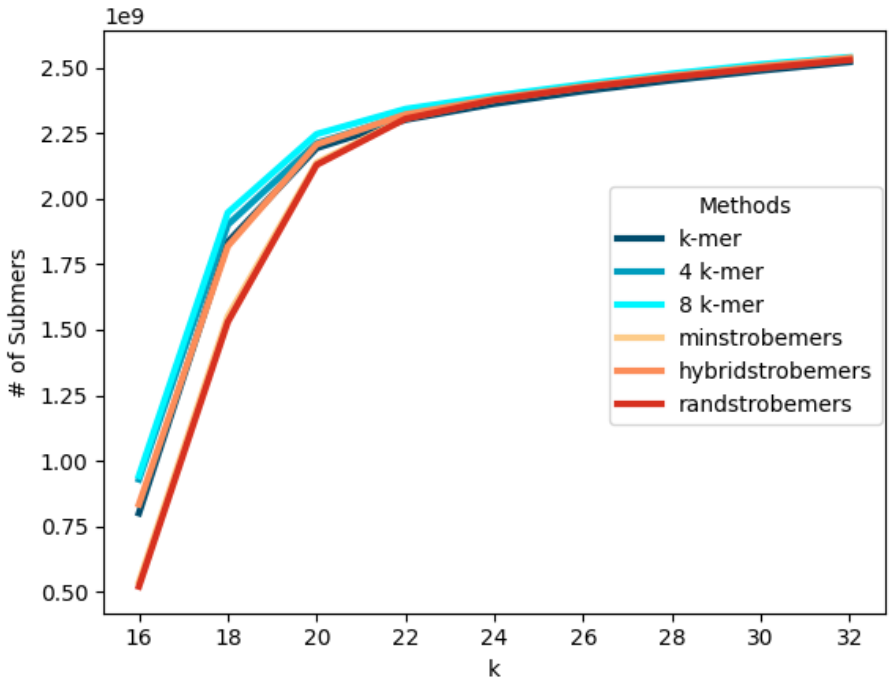
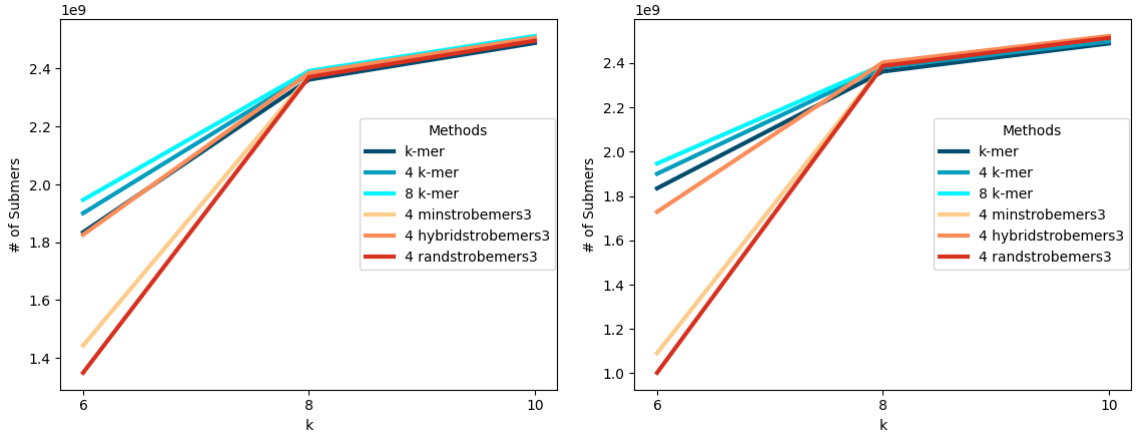
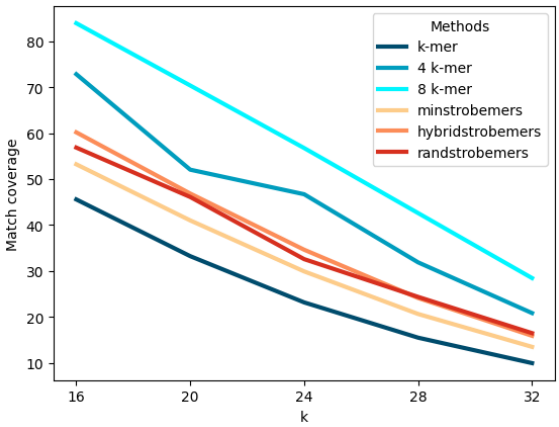


Figure A.3: **Set size of different submer methods for the human genome.** 4 and 8 k-mer refer to the gapped $k+4$-mer or $k+8$-mer method with 4 or 8 gaps. The strobemers have an order of 2, so the singular strobes have a length of $k/2$ and they have a window length of $k/2+8$, so that a maximal gap of length 8 to the previous strobe is ensured.

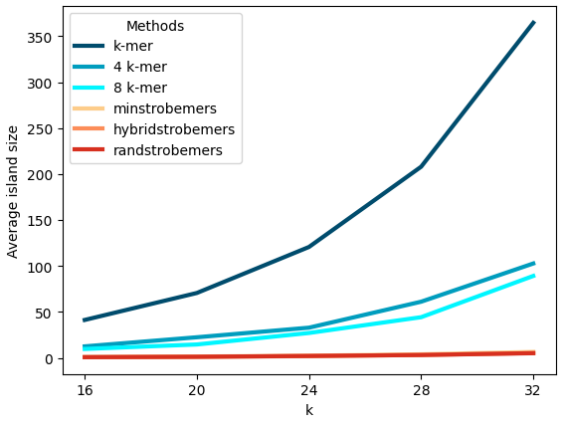


(a) Strobemers 4 (b) Strobemers 8

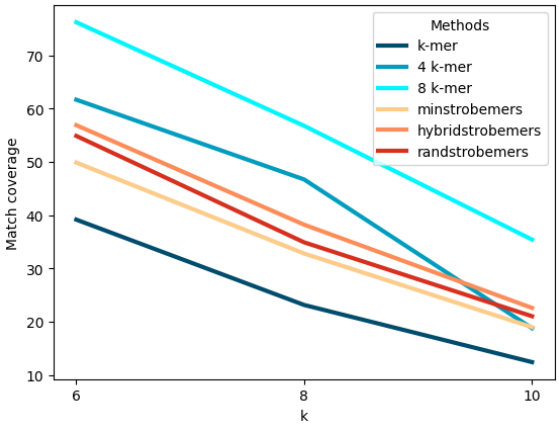
Figure A.4: **Set size of different methods for the human genome with strobemers of order 3.** 4 and 8 k-mer refer to the gapped $k+4$-mer or $k+8$-mer method with 4 or 8 gaps. The strobemers have an order of 3, so the singular strobes have a length of $k/3$ and they have a window length of $k/2+4$, so that a maximal gap of length 4 to the previous strobe is ensured for Strobemers 4, and a window length of $k/2+8$, so that a maximal gap of length 8 to the previous strobe is ensured, for Strobemers 8.



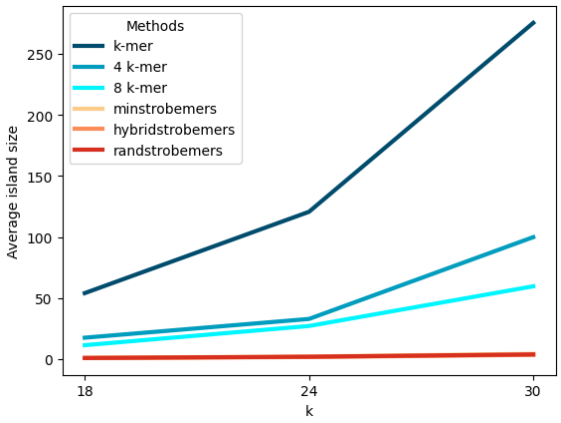
(a) Strobemers 8, Match Coverage



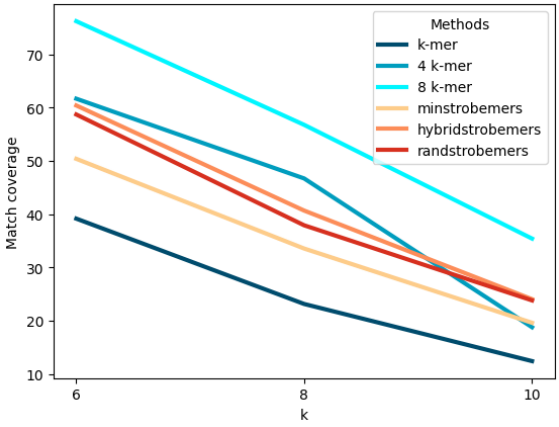
(b) Strobemers 8, Island Size



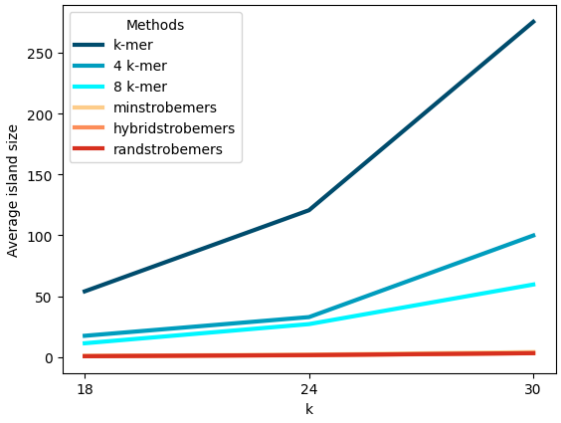
(c) Order 3, Match Coverage



(d) Order 3, Island Size

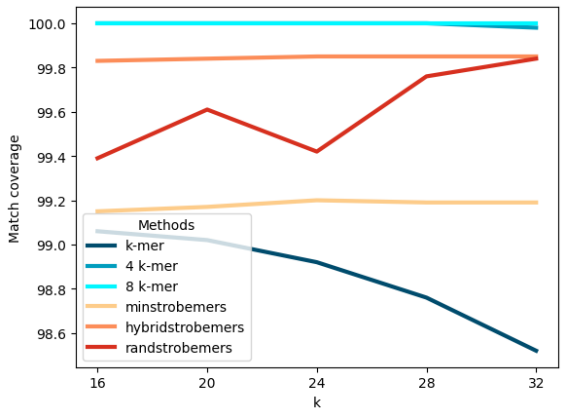


(e) Strobemers 8, Order 3, Match Coverage

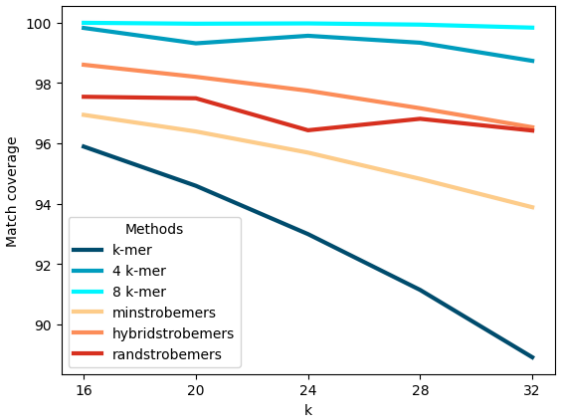


(f) Strobemers 8, Order 3, Island Size

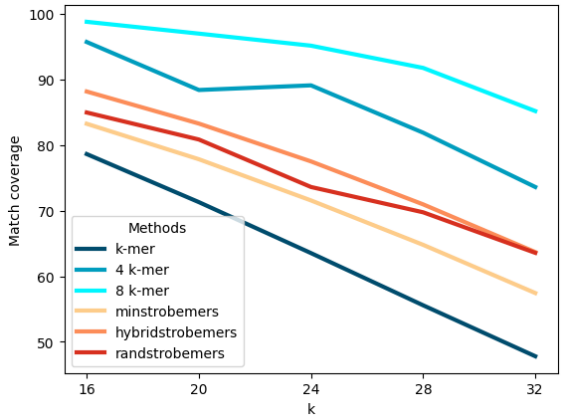
Figure A.5: **Match coverage and island size of different submer methods with substitution rate 10 for strobemers with a longer window length and order 3.** 4 and 8 k-mer refer to the gapped $k+4$-mer or $k+8$-mer method with 4 or 8 gaps. The strobemers have an order of 2 for Strobemers 8, so the singular strobes have a length of $k/2$ and they have a window length of $k/2+8$, so that a maximal gap of length 8 to the previous strobe is ensured. For the rest, the strobemers have an order of 3 with either a window length of of $k/2+4$ or of $k/2+8$ (Strobemers 8).



(a) Substitution rate of 1%

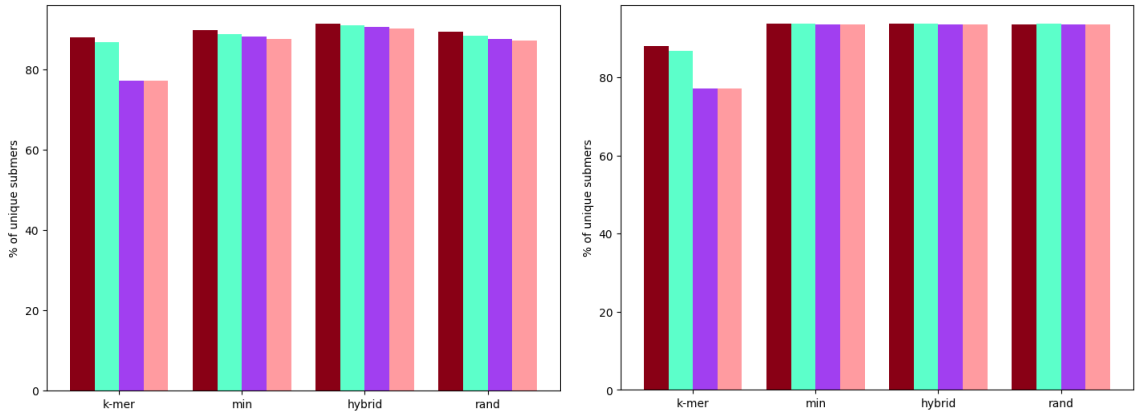


(b) Substitution rate of 2%



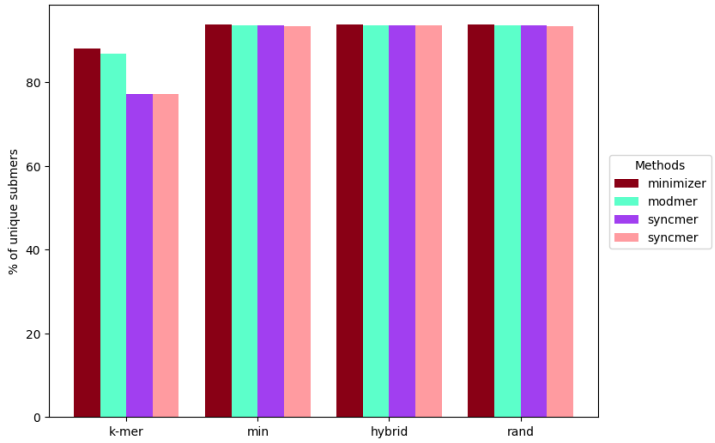
(c) Substitution rate of 5%

Figure A.6: **Match coverage of different submer methods for different substitution rates.** 4 and 8 k-mer refer to the gapped $k+4$-mer or $k+8$-mer method with 4 or 8 gaps. The strobemers have an order of 2, so the singular strobes have a length of $k/2$ and they have a window length of $k/2+4$, so that a maximal gap of length 4 to the previous strobe is ensured.



(a) Strobemers 8

(b) Strobemers 4, order 3



(c) Strobemers 8, order 3

Figure A.7: **Average percentage of unique submers for different representative methods for strobemers with a longer window length and of order 3.** The figure shows the average percentage of unique submers for all representative methods for k-mers and the different strobemers as underlying method. The average is calculated over the different parameters for the window size for minimizer, the modulo for modmers and the s size for syncmers. The following parameters were used: $[24, 28, 32, 36, 40]$ $((w, k_l)$-minimizer), $[3, 5, 7, 9, 11]$ $((k_l, m)$-modmers), $[18, 16, 14, 12, 10]$ $((k_l, s, [0], 1)$-syncmers) and $[15, 11, 7, 3, 1]$ $((k_l, s, [0, 20 - s], 1)$-syncmers) with k_l being 20 for a) and 27 for b) and c). For a), as underlying submer method, 20-mer were used for the k-mers and for the different strobemers $(2, 10, 10, 18)$-strobemers were used. For b) and c), 27-mers were used for the k-mers and for the different strobemers $(3, 9, 9, 13/17)$-strobemers were used.

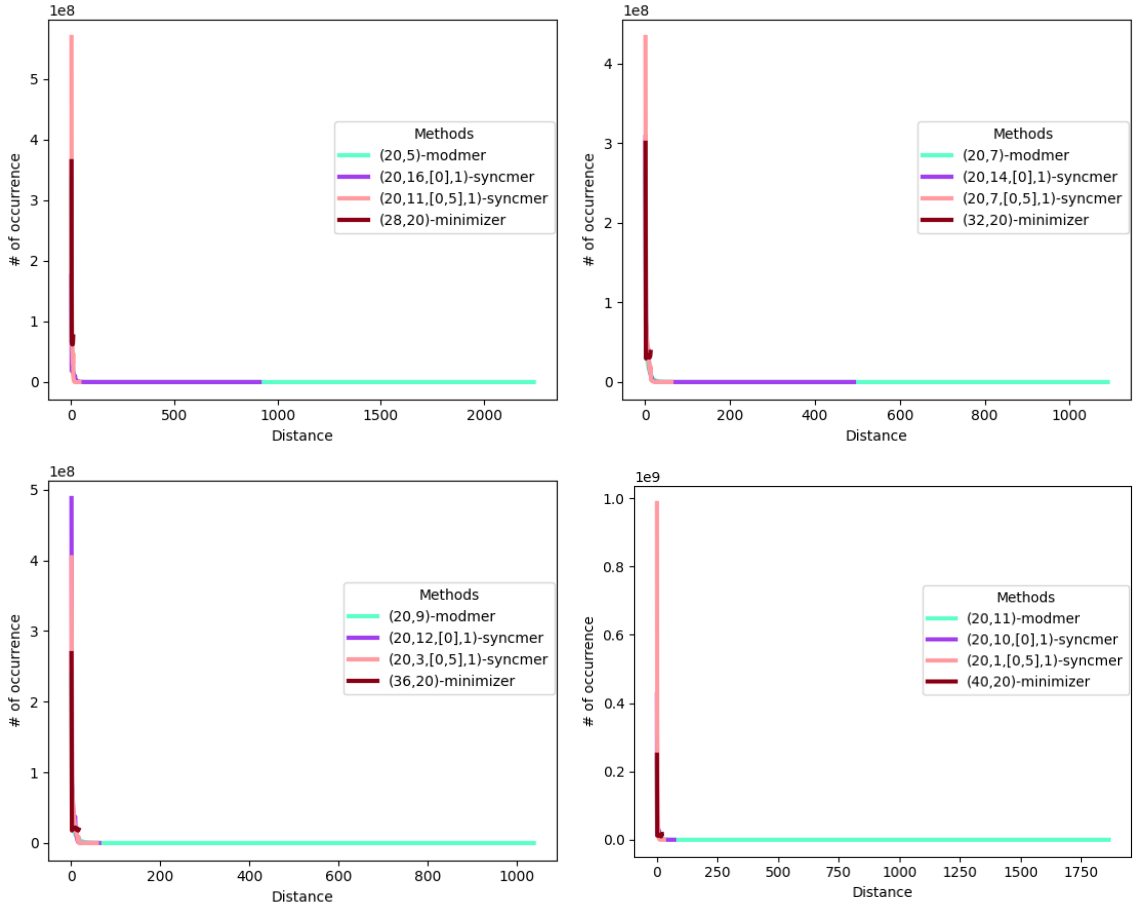
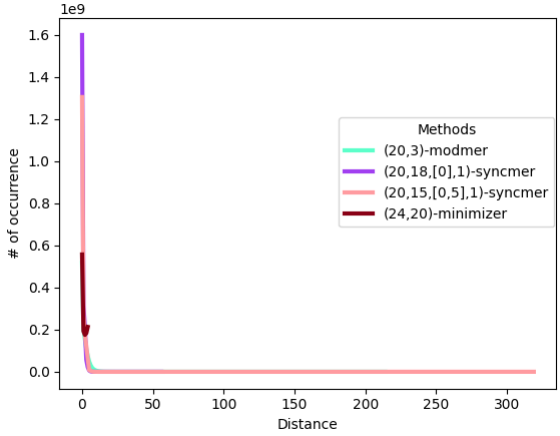
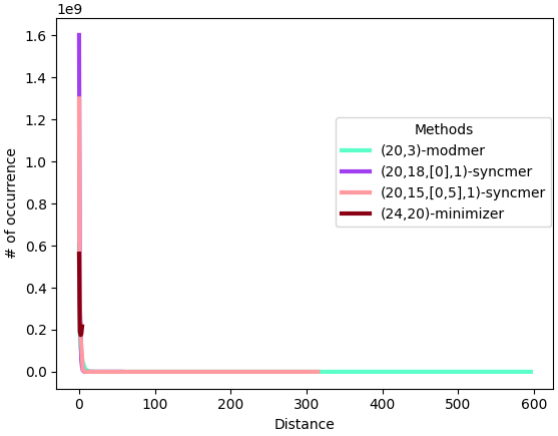


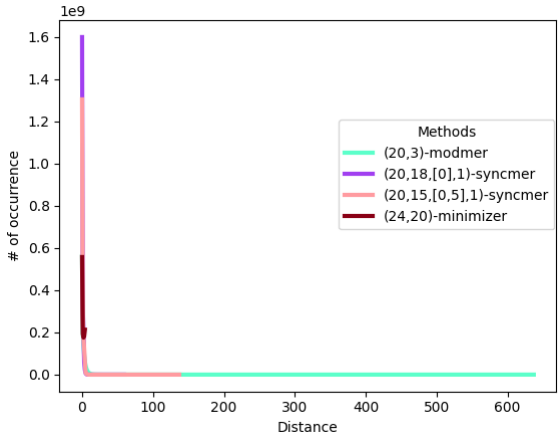
Figure A.8: **Distance between representative submers of different representative methods.** The figure shows the occurrence of different distances between adjacent representative submers on the human genome for different representative submer methods based on k-mers.



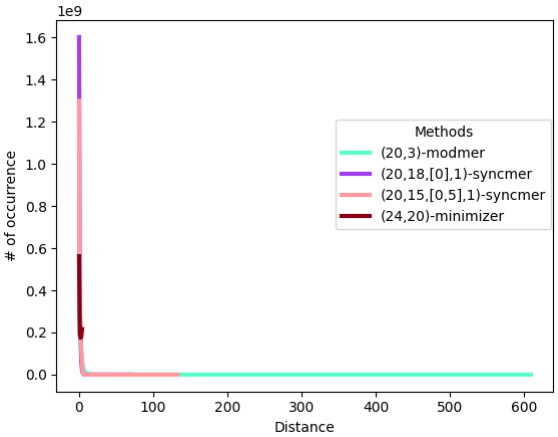
(a) Hybridstrobemers, 4



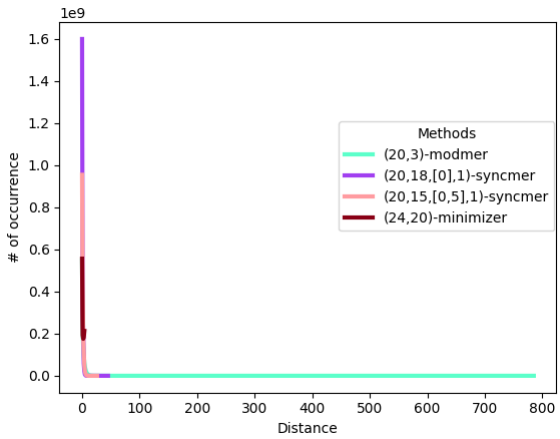
(b) Hybridstrobemers, 8



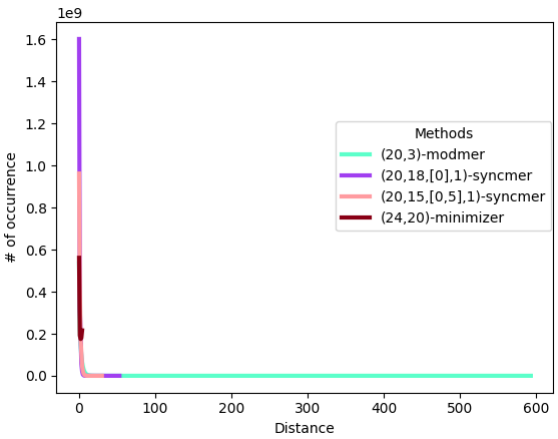
(c) Minstrobemers, 4



(d) Minstrobemers, 8

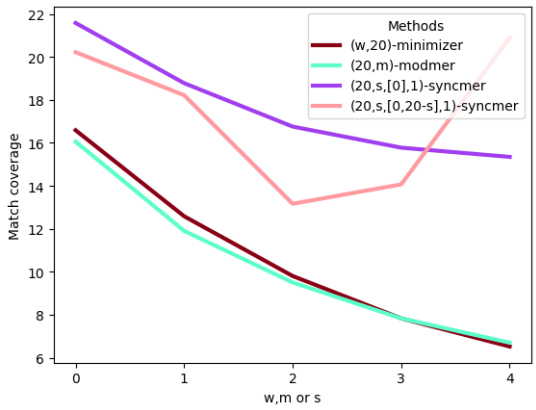


(e) Randstrobemers, 4

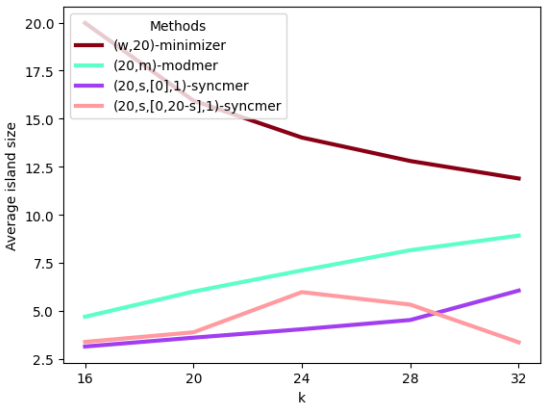


(f) Randstrobemers, 8

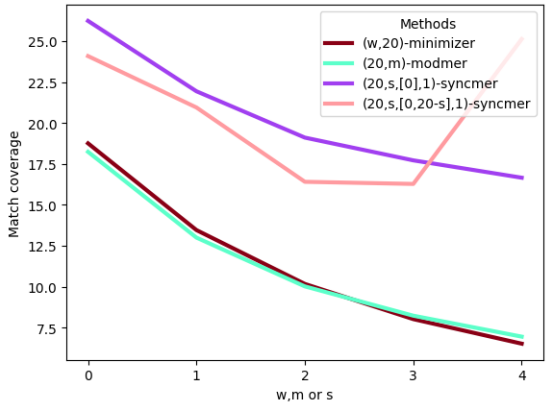
Figure A.9: **Distance between representative submers of different representative methods.** The figure shows the occurrence of different distances between adjacent representative submers on the human genome for $(24, 20)$-minimizers, $(20, 3)$-modmers, $(20, 18, [0], 1)$-syncmers and $(20, 15, [0, 5], 1)$-syncmers based on different strobemers of order 2. The 4 in the caption references the $(2, 10, 10, 14)$-strobemers and the 8 the $(2, 10, 10, 18)$-strobemers.



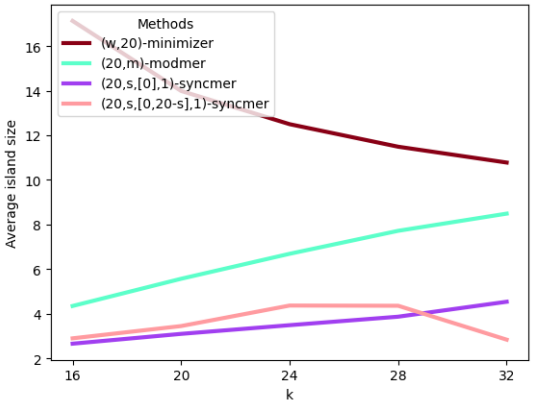
(a) Match Coverage, Minstrobemers



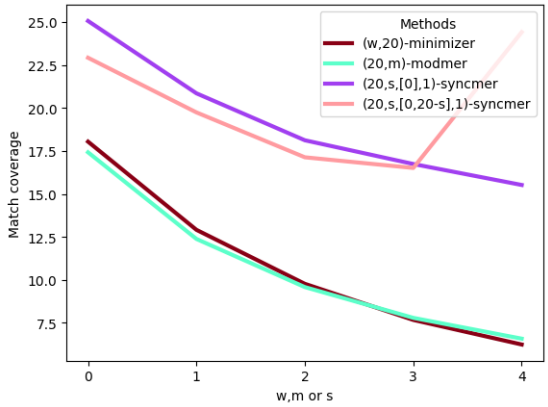
(b) Island Size, Minstrobemers



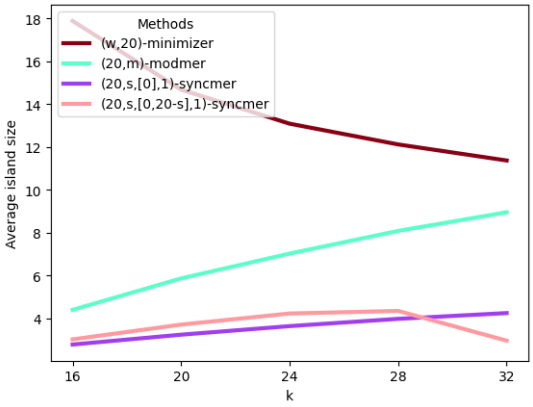
(c) Match Coverage, Hybridstrobemers



(d) Island Size, Hybridstrobemers



(e) Match Coverage, Randstrobemers



(f) Island Size, Randstrobemers

Figure A.10: **Match coverage and island size of different representative submer methods based on strobemers of order 2 with maximal 8 gaps between two strobes.** The figure shows the match coverage and island size of representative submers based on $(2, 10, 10, 18)$-strobemers. The window size for minimizer, the modulo for modmers and the s size for syncmers changes for every value on the x-axis. The x-value indicates the position in the following lists: $[24, 28, 32, 36, 40]$ (minimizer), $[3, 5, 7, 9, 11]$ (modmers), $[18, 16, 14, 12, 10]$ $((20, s, [0], 1)$-syncmers) and $[15, 11, 7, 3, 1]$ $((20, s, [0, 20 - s], 1)$-syncmers).

A.3 Indexing big data

A.3.1 Overview of indices

ASSBT	[119]
BFT	[30]
BIGSI	[123]
BLight	[165]
COBS	[21]
Counting DBG	[95]
Gazelle	[127]
HDSBT	[121]
IBF	[66]
kcollections	[43]
kmtricks	[111]
kmindex	[166]
Mantis	[110]
Manis-MST	[115]
Mantis-BS	[116]
MetaGraph	[101]
MetaProFii	[124]
Multi-BRWT	[128]
Needle	[32]
PAC	[103]
Rambo	[125]
Raptor	[65]
Raptor 2.0	[126]
Reindeer	[8]
SBT	[97]
SeqOthello	[102]
SSBT	[120]

Table A.1: **References for the Figure 3.1.**

A.3.2 Space and Speed Analyses

All benchmarks and analyses can be found in the *Needle* repository (`https://github.com/seqan/needle`) under the folder "utils". The time and memory consumption was always measured by */usr/bin/time -v*.

For the comparison of the different *Needle* parallelization methods, for the comparison of the preprocessing for all quantitative methods and for showing the performance of *Needle*'s insertion and deletion, the following four sequencing experiments were used:

- SRR1313229

- SRR1313228
- SRR1313227
- SRR1313226.

Construction

normal	clean	130	2.1
	build	22	50.5
	transform	39	7.4
	build 2	7	23.7
	transform 2	2	5.1
	annotate	445	6.0
	transform_anno 0	13	32.3
	transform_anno 1	77	347.1
	transform_anno 2	99	312.6
	transform_anno	74	52.1
	relax_brwt	8	20.4
smooth	clean	118	2.1
	build	22	50.5
	transform	23	7.4
	build 2	7	23.7
	transform 2	2	5.2
	annotate	363	5.4
	transform_anno 0	9	32.0
	transform_anno 1	82	345.0
	transform_anno 2	93	149.2
	transform_anno	67	20.4
	relax_brwt	5	8.8

Table A.2: **Speed and space analyses of the individual MetaGraph steps.** The time is given in minutes and the RAM in GB. All commands were run with 4 threads. The column on the left specifies, if the counts were smoothed (named smooth) or not (named normal). For the exact commands, see `https://github.com/seqan/needle/blob/master/utils/metagraph.sh`

The index size was determined by taking into account all files that are needed for a successful query. The file suffixes for each application are listed here:

- MetaGraph: .dbg, .dbg.weights, relaxed.row_diff_int_brwt.annodbg
- Needle: IBF_Data, IBF_Level_0-14, IBF_Levels.levels, IBF_FPRs.fprs
- Reindeer: index.gz, matrix_eqc.gz, matrix_eqc_info, matrix_eqc_position.gz.