REINDEER: efficient indexing of *k*-mer presence and abundance in sequencing datasets

Camille Marchet[1], Zamin Iqbal[2], Daniel Gautheret[3], Mikael Salson[1], and Rayan Chikhi[4]

[1] University of Lille, France, camille.marchet@univ-lille.fr,
[2] European Bioinformatics Institute, UK,
[3] Université Paris-Saclay, France
[4] Institut Pasteur & CNRS, France

Abstract. Analyzing abundances of sequences within large collections of sequencing datasets is of prime importance for biological investigations, such as the detection and quantification of variants in genomes and transcriptomes. In this work we present REINDEER, a novel computational method that performs indexing of k-mers and records their counts across a collection of datasets. We demonstrate that REINDEER is able to index counts within 2,585 human RNA-seq datasets using only 36 GB of RAM and 60 GB of disk space during construction, and 75 GB of RAM during random access queries. To the best of our knowledge, REINDEER is the first practical method that can index k-mer counts in large dataset collections. Briefly, REINDEER constructs the compacted de Bruijn graph (DBG) of each dataset, then implicitly represents a merged DBG of all datasets. Indexing all the k-mers present in the merged DBG would be too expensive; instead REINDEER indexes a set of *minitigs*, which are sequences of coherently grouped k-mers. Minitigs are then associated to vectors of counts per dataset. Software is available at `github.com/kamimrcht/REINDEER`.

Keywords: sets of k-mer sets·quantification·high-throughput sequencing.

1 Introduction

An overwhelming amount of sequencing data is now publicly available in repositories such as the European Nucleotide Archive (ENA) [3] and NCBI's Sequence Read Archive (SRA) [15]. They contain a wealth of DNA and RNA sequencing experiments on model species, non-model species, tumor biopsies, cell lines, metagenomes. Searching within them, in a holistic setting, would open the tantalizing possibility of making scientific discoveries using world-scale biological data. However, the immense size of these repositories renders any large-scale investigation extremely difficult. SRA hosts around 30 petabases of experiments at the time of writing. Not only is searching within this mass of data impractical, but also merely downloading a copy of ENA or SRA would take in the order of years. Thus, there is a pressing need for solutions that, possibly using a centralized server, would provide search functionality across these databases.

We use the term *dataset* to refer to a set of reads resulting from sequencing an individual sample. Searching for a sequence within a single assembled genome (e.g., with BWA-MEM [16]) or within a single dataset (e.g., with BEETL [14] or SRA-BLAST [8]) is now routine and can be considered to be a well-managed computational problem. Since two decades, searching within collections of assembled genomes (e.g., with BLAST [2] or DIAMOND [7]) is equally practical. However, searching for sequences within collections of (unassembled) datasets is still extremely challenging, and under very active computational investigation [20]. Recent works have introduced novel k-mer (substrings of size k in biological sequences) data structures, showing that k-mer searches are a useful proxy for exact and approximate sequence search. In a nutshell, state of the art methods are able to determine the presence/absence of any k-mer within collections containing up to around 10^4 indexed datasets. By extension, any longer sequence such as a gene or a transcript can be queried. To the best of our knowledge, no method is currently able to record abundances of k-mers across collections at this scale.

While nearly all recent works on indexing datasets have focused on recording the presence/absence of k-mers, in this work we focus on the more space-expensive matter of indexing their counts, *i.e.*, how many times is a k-mer present within each dataset. We argue that querying abundances of sequences within large collections of datasets is of prime importance in several biological scenarios. Foremost, large-scale efforts for indexing abundances in transcriptome collections are currently ongoing. For example the GTEx project [18] hosts a database of transcript-level abundances across 17k tissue samples (as of January 2020). Pan-cancer studies such as TCGA [29] collects the expression of genes, exons, miRNAs, as well as genomic copy-numbers. Those signals are usually recorded at the level of genes or comparatively long genomic regions, yet they would further benefit from nucleotide-level resolution, for example to investigate single-nucleotide variants and splice junctions. Secondly, metagenomic sequencing is providing a widening window onto the diversity of life on earth. Comparing abundances of contigs, SNPs or species across different metagenomic datasets is a vital step in understanding how community composition varies with context. Thirdly, sequencing is moving from simply a route to single-genome assembly, to providing a counter or sensor for events: this progression can be seen from CHIP-seq, ATAC-seq, through to single-cell sequencing - after which counts are modelled based on the precise experiment.

We also highlight that merely adapting existing data structures by transforming the 1-bit presence/absence information into a 16-bit counter is unlikely to be a viable strategy. For instance, consider the HowDeSBT data structure [12], a recent technique for indexing the presence/absence of k-mers across dataset collections. It saves space by using a single memory location to encode the presence of a k-mer across multiple datasets. Yet this scheme cannot be adapted to record abundances, as a k-mer may be present in multiple datasets at significantly different abundances, which cannot be recorded by a single memory location. Likewise, BIGSI [5] uses Bloom filters with 25% false positive rate to encode presence/absence of k-mers; extending Bloom filters to support abundance queries (e.g. using Count-Min sketches) at a comparable false positive rate would introduce significant abundance estimation errors.

Here we introduce REINDEER, a novel computational method that performs indexing of k-mers and records their counts across a collection of datasets. REINDEER uses a combination of several concepts. The first novelty is to associate k-mers to their counts within datasets, instead of only recording the presence/absence of k-mers as is nearly universally done in previous works. To achieve this, a second novelty is the introduction of *monotigs*, which allows space-efficient grouping of k-mers having similar count profiles across datasets. An additional contribution is a set of techniques to further save space: discretization and compression of counts, on-disk row de-duplication algorithm of the count matrix. As a proof of concept, in this article we apply REINDEER to index a *de facto* benchmark collection of 2,585 human RNA-seq datasets, and provide relevant performance metrics. We further illustrate its utility by showing the results of queries on four oncogenes and three tumor suppressor genes within this collection.

2 Problem statement

The questions addressed by REINDEER can be formally framed as follows. Let S be a DNA sequence of arbitrary length (such as a gene, a transcript, or a shorter sequence), and C a collection of datasets. The aims are either to a) *determine the presence* of S (either as an exact or an inexact match) or b) *count the number of occurrences* of S within elements of C. In its simplest form, aim a) amounts to returning the subset of datasets in which S appears. At first glance the complexity of answering either a) or b) appears to depend on the cardinality of C and the sizes of its constituent datasets. In a seminal work, Solomon and Kingsford [27] slightly reformulated aim a) in order to facilitate its resolution. Considering the set Q of all k-mers extracted from S. they propose to instead determine the subset of datasets in which at least t k-mers of Q appear, with $0<t\leq|Q|$. This is not strictly equivalent to computing the (exact or approximate) match of S within datasets. In this new formulation k-mers from Q may also be found in different sequences across a dataset, e.g. if ACT and CTG are two k-mers seen in two different reads, sequence ACTG is assumed to be present in the dataset regardless on whether it is actually part of a read. Yet this reformulation can be considered as a reasonable approximation. We also adopt it in this work, and propose to support the following two types of queries:

1. **Assess S's count in datasets.** Estimate the abundance of S in each dataset by reporting the mean count of all k-mers from Q that are (exactly) present in the dataset.
2. **Assess S's presence in datasets.** Output a list of datasets in which at least t k-mers from Q appear.

Supporting queries of type 1 will be the main improvement proposed by this work compared to the state-of-the-art. In the rest of the manuscript, we will only focus on solving type 1 queries. For queries of type 2 the same principles will apply, as they can be derived from queries of type 1: S is considered to be present if and only if it has non-zero abundance.

3 Preliminaries

3.1 De Bruijn graphs and associated concepts

In this section, we recall the previously known concepts of de Bruijn graphs, unitigs, and compacted de Bruijn graphs. We further describe two recent concepts: spectrum-preserving string sets, and a de Bruijn graph indexing scheme (BLight [21]).

We consider an alphabet Σ of fixed size 4 (w.l.o.g, $\Sigma=\{A,C,G,T\}$). A *set of reads* R (also called *dataset*) is a set of finite strings over Σ. We will consider all the strings of length k (called *k-mers*) present in R, for some fixed value of $k>0$. For simplicity of presentation, reverse-complements are ommited from the definitions, but they are taken into account in the canonical way in the software.

Definition 1. de Bruijn graph (DBG). *The de Bruijn graph is a directed graph $G_k(R)=(V,E)$ where V is the set of k-mers that appear in R, and for $u,v\in V$, $(u,v)\in E$ if and only if $u[2,k]=v[1,k-1]$.*

Importantly, this definition of the Bruijn graphs is node-centric [9], i.e. the set of edges can be inferred given the nodes. A node-centric de Bruijn graph contains the same information as a k-mer set.

We call *unipath* in G_k a set of nodes $s=u_0,...u_n$ such that, for each $0\leq i\leq n+1$, $(u_i,u_{i+1})\in E$, in and out-degrees of each u_i for $1\leq i\leq n-1$ are 1, and the in-degree of u_0 and the out-degree of u_n are >1. Intuitively, a unipath is a linear portion of the DBG.

Definition 2. Unitig. *The string corresponding to a maximal-length unipath $s = u_0,...u_n$ can be defined by concatenating u_0 with all the $u_i[k-1]$ (the k-th letter of u_i), in order, for $0<i\leq n$. The resulting string of length $k+n$ is called a unitig.*

Definition 3. Compacted de Bruijn graph. *The directed graph where nodes are unitigs, and edges are $k-1$-overlaps between two nodes sequences, is called a compacted de Bruijn graph.*

Contrary to a regular DBG, a compacted DBG has nodes of various length ($\geq k$). In the following, unless explicitly mentioned, the objects referred to as "de Bruijn graphs" are compacted de Bruijn graph. Even though compacted DBGs are technically not DBGs, they do equivalently represent the same k-mer set.

Each k-mer has a count corresponding to its abundance in the reads of a dataset. The k-mer counts of each k-mer can be obtained for each dataset using a k-mer counter [11].

Definition 4. Unitig abundance. *We call the abundance $c(u)$ of a unitig u:*

$$c(u)=\frac{1}{n}\sum_{i=1}^{n} count(k_i)$$

for $k_1,..,k_n$ the k-mers of the unitig u of length $n+k-1$.

Definition 5. Union De Bruijn graph. *Given a multiset of de Bruijn graphs $\{G_1,...G_k,...G_c\}$ generated from c datasets, we call union de Bruijn graph the de Bruijn graph which has a node set equal to the union of the nodes set of $\{G_1,...G_k,...G_c\}$. The edges are $k-1$-overlaps between the nodes of the union de Bruijn graph.*

It follows that a union DBG also represents a k-mer set (the union of c k-mer sets). We now recall the recent concept of spectrum-preserving string sets [25], in order to draw new bridges between previous works, and also with our contribution.

Definition 6. Spectrum-preserving string set (SPSS). *Given a set of k-mers X, a set of strings S of length $\geq k$ is a spectrum-preserving string set representation of X if the set of distinct k-mers present in S is exactly X.*

In particular, a SPSS can be used to represent the k-mer set of a DBG.

Given a set of k-mers X, the set of strings representing each k-mer is itself a SPSS of X, albeit not a very space-efficient one. The set of unitigs constructed from X is also of SPSS of X. We will discuss more types of SPSSs below.

3.2 Spectrum-preserving string sets in relation to k-mer indexing

One of the fastest and most space-efficient techniques available to index and associate information to k-mers is BLight [21], a type of hash table tailored to large sets of k-mers. Given a collection of datasets, the input data given to BLight is a SPSS of the k-mers from the union de Bruijn graph of this collection. In its inner steps, BLight relies on a particular type of SPSS, where in each sequence of the SPSS all k-mers need to have the same minimizer.

Definition 7. Minimizer [26]. *The m-minimizer of a sequence S is the smallest substring of size m in S, according to some ordering.*

Here m-minimizers are computed on k-mers, hence $m<k$, and the order on m-mers is given by a hash function (i.e., it is not the lexicographic order).

Definition 8. BLight indexing scheme [21]. *Given a set of k-mers X, let S be a SPSS of X such that in each string $s \in S$, all k-mers in s have the same m-minimizer. Then BLight uses S to compute a static, collision-free hash table where the set of keys is X.*

In the original BLight article, the SPSS of X is the set of all the super-k-mers found in the unitigs, as defined below.

Definition 9. Super-k-mer [11]. *Given a sequence S, a super-k-mer is a substring of S of maximal length such that all k-mers within that substring have the same m-minimizer sequence.*

Observation 1 *Given a set of k-mers X, the set of all super-k-mers constructed from the unitigs of the DBG of X is a SPSS of X.*

Concretely, the original version of BLight takes as input the unitigs of the DBG of X, and transforms them into another SPSS: the super-k-mers of unitigs, partitioned according to their minimizers. However, it is important to note that this is not the only possible SPSS scheme that can be used by BLight. In fact, in this article we will use another one.

In the inner workings of BLight, querying a k-mer x consists in first computing the minimizer of x in order to identify in which partition it is expected to be found. Then a minimal perfect hash function [17] associates x to its position within a sequence of the SPSS. The position is then converted into a number that indicates a location in memory, that can be used to associate a k-mer to some piece of information. Of note, BLight only records information about the set of k-mers itself, but has no additional information such as their presence/absence within each dataset, or their counts. Another data structure will be needed along with BLight to record such information, which is in fact the purpose of REINDEER.

SPSS scheme	Description
k-mers	trivially, a set of k-mer strings is its own SPSS
unitigs	compacted maximal unipaths in a de Bruijn graph
super-k-mers of reads (KMC [11])	substrings of reads of maximal length such that all k-mers in a super-k-mer share the same m-minimizer
super-k-mers of unitigs (BLight [21])	same as above, except substrings of unitigs instead of substrings of reads
UST [25]	set of sequences obtained by greedily concatenating unitigs in order to minimize the total number of nucleotides in the SPSS
simplitigs [6]	similar to [25]
monotigs	a set of paths that covers the (uncompacted) de Bruijn graph such that all k-mers have an identical count vector

Table 1. Categories of spectrum-preserving string set schemes known from previous literature (and monotigs, introduced in this article). See also Fig. 1 for an example of each category.

So far, we have reviewed the following SPSSs for a set of k-mers X: X itself, the unitigs of X, any set of super-k-mers that together contains all k-mers of X (such as the super-k-mers of the sequencing reads where X originated from), and the super-k-mers of the unitigs of X. To this list we can add the recently-introduced (and equivalent) concepts of UST and simplitigs [25,6]. They are SPSSs that aim to minimize their total number of nucleotides. Note that contigs resulting from a genome or transcriptome assembly of X, unlike unitigs, are not a SPSS as they typically discard some of the k-mers to generate consensus sequences. Table 1 recapitulates all SPSS schemes (monotigs will be defined later in the article). In the next section, we will show that the choice of SPSS is important for our specific application. We will introduce a new SPSS that is better suited to the purpose of indexing collection of datasets.

4 Introducing SPSS for indexing k-mer counts

Due to the stochasticity of sequencing coverage, genomes and transcriptomes are not evenly covered by sequencing reads. Thus, k-mers coming from regions that have the same copy-number (for genomes – or abundance, for transcripts) often have close but non-identical counts. Yet, if those k-mers had identical counts, one could exploit such redundancy in order to save space when indexing counts. While it would be unrealistic to assume that k-mer counts are constant across genomic regions of same copy-counts (or regions of same abundances across a transcript), we will make the following assumption throughout the rest of the article:

Assumption 1 *Within a dataset, the counts of k-mers that are part of the same unitig are assumed to all be identical.*

We argue that Assumption 1 yields a robust approximation of k-mer counts. To support this claim, we computed the Spearman correlation coefficient of true k-mer counts versus counts averaged per unitig, across all unitigs of a RNA-seq dataset (SRR1002076, $k=21$), resulting in $\rho=0.87$.

4.1 Count vectors and SPSSs

Using the indexing scheme presented in the previous section (a SPSS indexed by BLight), our aim is to associate to each k-mer its abundances in datasets.

Definition 10. Count-vector. *Given an ordered list of datasets and a k-mer x, the* count-vector *of x is a vector in which the integer at position i represents the number of times x is seen in the i-th dataset.*

Our initial motivation is to seek a SPSS scheme where a single count-vector can be associated to each string of the SPSS. In other words, for each string s of the sought SPSS, all k-mers in s need to have the same count-vector.

It is immediate that one may achieve this goal by taking X to be its own SPSS, i.e. by considering the set of strings consisting of each k-mer taken individually. Since each string of the SPSS only contains a single k-mer, a single count-vector is associated to each element of the SPSS. However this is not a scalable solution, as thousands of mammalian-sized datasets contain billions of distinct k-mers, leading to prohibitively high memory consumption.

Yet recall that BLight allows to index any SPSS, as long as it satisfies the minimizer condition of Definition 8. It turns out, perhaps surprisingly, that none of these SPSS schemes described in the previous section would be suitable for associating elements to count-vectors. We describe why below.

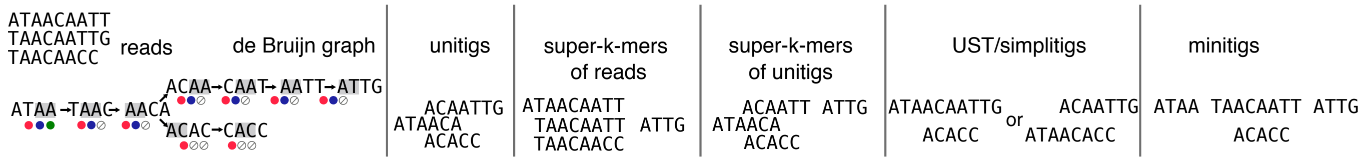

Fig. 1. Example of a de Bruijn graph (left part) and various SPSS schemes over the set of nodes of the graph (rightmost panel). There are three datasets (red/blue/green) with one read per dataset. In this first panel, the minimizer of each k-mer is highlighted in grey ($k=4$, $m=2$). Dots below k-mers symbolize their non-zero abundance (filled circle) or absence ($\emptyset$) across the three datasets. In the five other panels, sequences within each SPSS scheme are roughly represented according to the positions of constituent k-mers in the DBG, only for visual indication. Observe that monotigs are the only SPSS scheme where each sequence can be associated to a single combination of colored dots.

Most of the time, the k-mers within a unitig do not share the same minimizer, thus unitigs cannot be used as a SPSS for BLight. This is why in the original BLight scheme, super-k-mers are extracted from the unitigs in order to construct a suitable SPSS. Even so, a second issue occurs: in the union DBG, k-mers coming from different datasets can overlap, despite their overlap being seen in no read. We will call such unitigs *chimeric* (see Figure 2). It is clear that k-mers within chimeric unitigs do not all have identical count-vectors, and in particular they contain k-mers that do not all belong to the same set of datasets. This leads to the impossibility of associating a single count-vector to chimeric unitigs or even to super-k-mers of chimeric unitigs. This is shown in the Observation below.

Observation 2 *Under Assumption 1, it is possible that not all k-mers in super-k-mers from the unitigs of a union DBG have the same count-vector.*

Proof. We prove the observation by showing an example (illustrated by Figure 2, second column). Suppose we have two datasets, d_1 and d_2. Dataset d_1 (resp. d_2) contains a single unitig u_1 (resp. u_2). Further assume that u_1 is a prefix of u_2 and $u_1 \neq u_2$. Observe that u_2 remains a unitig in the union DBG of $d_1 \cup d_2$. The first $|u_1|-k+1$ k-mers of u_2 have the same count-vector: $[s_1,s_2]$ (s_1 and $s_2>0$). The last $|u_2|-|u_1|-k+1$ k-mers of u_2 have count-vector $[0,s_2]$. It is possible to design sequences for u_1 and u_2 such that the last k-mer x of u_1 (which is also a k-mer in u_2) shares a minimizer with the k-mer y that follows x in u_2. Then the super-k-mer that contains both x and y will be associated to at least two count-vectors: $[s_1,s_2]$ through x and $[0,s_2]$ through y, hence does not represent a single count-vector. □

Thus, super-k-mers of unitigs are not even an appropriate SPSS due to the absence of guarantee of having a single count-vector for each element of the SPSS. Of note, the other SPSS schemes UST and simplitigs aim to merge unitigs as much as possible. These schemes suffer from the same issue of having multiple count-vectors within a single sequence, at an even more extreme level than with single unitigs.

4.2 Monotigs

As we saw in the previous section, super-k-mers constructed from the unitigs of the union DBG were likely to a suitable SPSS, except that Observation 2 shows that a single count-vector cannot always be associated to each of them. However, this issue does not arise when considering a single dataset at a time.

Observation 3 *Under Assumption 1, within a given dataset, k-mers from super-k-mers of unitigs necessarily have the same counts.*

This leads us to introduce the new concept of monotig, and describe a simple construction algorithm for such objects.

Definition 11. Monotig. *A* monotig *is the sequence of a path in the union DBG in which all constituent k-mers have the same count-vector and the same minimizer.*

Observe that monotigs are not necessarily substrings of super-k-mers of unitigs, nor substrings of unitigs, as they can possibly span multiple unitigs. The set of monotigs forms a partition of the nodes of the DBG.

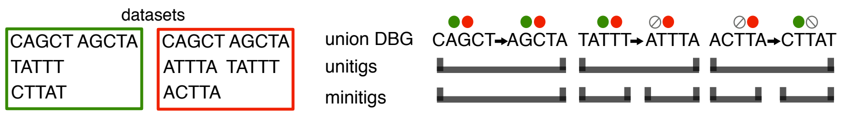

Fig. 2. Two datasets and their k-mers (k=5). A union DBG is built over these k-mers, and contains 3 unitigs. Per-dataset abundances are not shown, but colored spots above each k-mer symbolize them. In the left-most unitig, the two k-mers have the same abundance information. Thus the monotig is here equivalent to the unitig. In the center and right-most unitigs, k-mers that have contrasted abundances share a $k-1$ overlap and can be compacted. On the contrary, monotigs will stop when a discrepancy occurs between two overlapping k-mers.

5 REINDEER data structure

Given a collection of datasets, REINDEER is an index that associates to any k-mer its counts in all datasets. As a byproduct of recording counts, it is also possible to query the presence/absence of k-mers within these datasets (boolean result). However, in the following we will focus on abundance queries as they are the main focus and the main contribution in REINDEER.

In line with recent works [23,30,12], REINDEER conceptually transforms an input collection C of datasets into a set of multisets of k-mers. As a pre-requisite, a compacted DBG needs to be built for each dataset (e.g. using BCALM2 [10]). REINDEER then takes as input the collection of compacted DBGs, and a single count value per unitig present in each DBG. It represents implicitly the union DBG of all the compacted DBGs, which is never explicitly constructed in memory. To summarize, REINDEER is built in three steps, illustrated in Figure 3:
1. For each dataset, build an individual DBG that also records k-mer counts, averaged per unitig.
2. Construct monotigs (according to the greedy algorithm of Section 5.1) to index all k-mers from the union DBG of all datasets.
3. Associate count-vectors to monotigs, and further discard duplicated count-vectors.

The following three sections describe the monotig construction, and the association of monotigs to count-vectors in more details.

5.1 Construction of the monotigs

We propose a greedy algorithm to compute a set of monotigs efficiently. The algorithm takes as input a set of compacted DBGs (one DBG per dataset).

First, super-k-mers of unitigs are computed for each dataset. The rationale for using super-k-mers instead of k-mers is to save space, as opposed to having to store all k-mers on disk. Each super-k-mer records its count in the DBG it originates from (there is a unique count, according to Observation 3). The super-k-mers are partitioned in files according to their minimizers. At this step, the same k-mer may be present in multiple super-k-mers within a file.

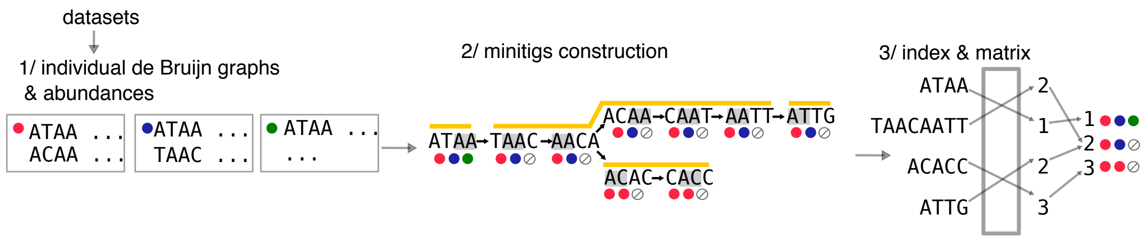

Fig. 3. Overview of REINDEER index construction. At step 1/, for each dataset, the (compacted) de Bruijn graph is given as input along with the count of each unitig. At step 2/, monotigs (corresponding to paths in yellow) are computed on the union De Bruijn graph of all datasets, and these monotigs are indexed. During step 3/ each monotig is associated (though an integer array) to a row in the de-duplicated count-vector matrix (rightmost matrix). In this matrix no two rows are equal, and in practice each row is compressed using run-length encoding.

Then, for each minimizer file, each super-k-mer is broken into k-mers. The k-mers are indexed, using an efficient hashing technique. When a k-mer is first indexed, it records in its count-vector the count from the super-k-mer it comes from. When a duplicate of an indexed k-mer is met by parsing another super-k-mer of the same minimizer (i.e., the same k-mer coming from another DBG), the k-mer's count-vector is updated with the new count information. Observe that each minimizer file can be processed independently and in parallel.

5.2 Construction of the count-vector matrix

Once all monotigs are computed, they are passed to BLight. The BLight hash table scheme is slightly modified in order to associate a single memory location to all k-mers within a monotig, instead of one memory location per k-mer. This is done in a straightforward way by returning the identifier of the monotig instead of the position of the queried k-mer within that monotig.

Once the BLight index is built, a *count-vector matrix* M is created such that its rows are the monotigs count-vectors, organized according to the order of the monotigs referenced by the values of the BLight hash table. In other words, the value $M[i][j]$ in this matrix gives the count of all k-mers within the ith monotig in the jth dataset. The matrix is stored in a row-major order.

5.3 De-duplication of rows in the count-vector matrix

Colored de Bruijn graphs and similar works initially used matrices to record colors (i.e., presence/absence information) of k-mers within a de Bruijn graph [22,13]. More recently, color equivalence classes have been introduced to decrease the redundancy in the matrix [23]. Here we apply a similar idea to save space, by de-duplicating rows of the count-vector matrix. Simply put we do not store the count-vector matrix, but instead we store a transformed version (the *de-duplicated count matrix*) which contains only distinct rows. In addition, we maintain an integer vector that associates to each monotig its corresponding row in the de-duplicated count-matrix.

5.4 Query

Queries in REINDEER are performed as follows. A query sequence S of size L is decomposed into its constituent k-mers, which are then individually queried in the index. The index converts each k-mer into a monotig identifier, that is then used to access a corresponding row of the de-duplicated count-vector matrix to retrieve the counts associated to this k-mer in all datasets.

Note that the presence/absence of a query k-mer x can be directly inferred from the count-vector matrix. Indeed, if x has a non-zero count in dataset j, it means that x is present in one of the unitigs of the DBG of dataset j and thus in some monotig i of the union DBG. Thus, the presence of x can be deduced by computing whether $M[i][j] > 0$.

To determine the abundance of S in each dataset, we make sure first that S is *present* in a dataset, i.e., at least θ of its constituent k-mers are present in a dataset (in other words, at least θ k-mers have a count ≥ 0 in the given dataset). θ is a parameter that can be set to the user's call.

If Reindeer is used to determine the presence/absence of S per datasets, the query stops after that step. For abundance, we maintain an integer vector array *Count* of size $c \times m$ (with c the number of datasets, and m being at most $L-k+1$ the number of k-mers in S), which is updated with a new vector each time a k-mer of S is found in a dataset.

The final query result for a given sequence S is a vector of tuples of size c. For a given dataset i, if for all for vectors $\{v_0,v_j,...v_m\}$, $Count[j][i]$ are identical (i.e., all k-mers of S present in i have the same count in i), then we report a single count value for this dataset i. Then each tuple in the result vector is of size 1. On the contrary, if not all $Count[j][i]$ are identical (i.e., some k-mers count values differ in i), then a tuple containing all seen count values is reported for i. This second case happens when k-mers from S fall in different unitigs in the DBG of dataset i, and these unitigs have different counts such as defined in Definition 4.

Relying on a hashing technique (BLight) during queries allows for fast queries as, unlike methods based on the Burrows-Wheeler transform, a constant (and small) number of memory accesses are performed. Query complexity for the presence/absence (or abundance) of a sequence of length L ($L > k$) in an index of c datasets is in time $\mathcal{O}(L \times c)$.

5.5 Implementation details

Reindeer index The BLight index and the color/abundance matrix are compressed (RLE+gzip) and dumped on disk, and are reloaded when doing a query. The software is available at `https://github.com/kamimrcht/REINDEER`.

Discretization of counts If exact k-mer counts are not required, then encoding counts using reduced integer precision leads to more effective de-duplication of the count-vector matrix. In REINDEER, we implement two different techniques. 1) Applying a function (termed *discretization*) that encodes counts from 0 to 50,000 in 8-bit integers, ensuring that discretized counts are at most 5% different from the real raw counts[5]. 2) A $\log_2$ transformation of counts (formally, $\max(0, \lceil \log_2(count) \rceil)$) yields an even more extreme discretization (in terms of loss of precision). An example of these transformations is shown in Table 2.

Low-memory de-duplication of rows in the matrix We iteratively compute the count-vectors of monotigs and compress them with run-length encoding[6], and then partition the whole multi-set of count-vectors by hashing into files stored on disk. When a count-vector is written to the disk, its monotig identifier given by BLight is also recorded next to it. Then, reading each file separately, we select a single representative of each vector by hashing them in an efficient dynamic hash table[7]. Once a file processed, we write the resultant representative on the disk, and record the correspondence between the monotig index and the position in the de-duplicated count-matrix matrix.

Type	Sample values for 4 datasets			
Raw counts	27	0	101	40812
Discretized counts	27	0	102	41000
$\log_2$-transformed counts	4	0	7	16
Presence/absence	1	0	1	1

Table 2. Types of count-vectors associated to k-mers.

Query speed-up and low-memory footprint In order for a query to not increase the memory footprint, we only decompress a single count-vector at a time, i.e. we process k-mers sequentially. We also take advantage of the fact that consecutive k-mers x,y in a query are also likely to be present in the same monotig. If it is the case, we directly copy the information of x for y instead of again interrogating the de-duplicated count-vector matrix.

6 Results

In their work, Solomon and Kingsford [27] indexed 2,652 human RNA-Seq samples to demonstrate the scalability of their approach. Since then other competing methods have also used this dataset, which has become a *de facto* benchmark. This benchmark was further reduced to 2,585 datasets to remove experiments containing short reads that

[5] Following code by G. Rizk, `github.com/GATB/gatb-core/blob/f58d96bbea953b37edaf455e4c3e404c89e95027/gatb-core/src/gatb/tools/collections/impl/MapMPHF.hpp#L85`

[6] `github.com/powturbo/TurboRLE`

[7] `github.com/martinus/robin-hood-hashing`

could not be indexed with 21-mers [12]. We use results from a recent and exhaustive review [21] in order to compare REINDEER with the state of the art in indexing collections of datasets. Despite not having run all methods on the same machine, the comparison of space usages for each data structure will nevertheless be accurate; results regarding execution times will certainly be machine-dependent but the magnitudes are expected to be roughly preserved.

We compare our approach with Solomon and Kingsford's original Sequence Bloom Tree (SBT) [27]; HowDeSBT, the most recent improvement of SBT [12]; BIGSI [5]; and methods relying on a de Bruijn graph: Mantis, where colors are stored in a counting quotient filter [23,1]; SeqOthello, storing k-mers with Othello hashing [30].

Among those approaches, only REINDEER and Mantis support exact k-mer queries, all the others are probabilistic, meaning that a given k-mer may be said to be present, while it is not. REINDEER is the only index to provide count information (as opposed to just presence/absence).

We provide scripts and guidelines to reproduces the following results on REINDEER's page[8].

6.1 Index construction

Comparison to state-of-the-art We first build REINDEER indexes on the 2,585 datasets, and filter out the rare k-mers on the same criteria as the other methods. We then compare our performance metrics to those reported for other tools on that same dataset [20]. Several construction modes are assessed: indexing presence/absence information only (for the sake of comparison only as our contribution consists in providing counts); and indexing counts in different modes (normal, discretized, $\log_2$). In order to be comparable to other results, we used a k value of 21 (and $m=10$). Other parameters in REINDEER are set to default values.

Table 3 shows the resources required at construction. Globally, REINDEER requires comparable resources as the existing methods, yet they only support presence/absence queries. REINDEER requires more disk space and the index is about as space consuming as Mantis, which also offers exact k-mer queries. However REINDEER was not conceived with presence/absence queries in mind, but for count queries. In such a case we see that the index is very efficient as it is about 2 to 4 times more space-expensive than presence/absence methods (yet with many of them having false positives). REINDEER space consumption can be additionally reduced by a third when storing approximate counts.

Tool	Max Ext. Memory (GB)	Time (h)	Peak RAM (GB)	Index Size (GB)	Counts (Y/N)
SBT	300	55	25	200	N
HowDeSBT	30	10	N/A	15	N
Mantis	3,500	20	N/A	30	N
SeqOthello	190	2	15	20	N
BIGSI	N/A	N/A	N/A	145	N
Reindeer - presence/absence	7,000	40	27	36	N
Reindeer - raw counts	7,000	45	56	52	Y
Reindeer - discretized	7,000	40	44	44	Y
Reindeer - log2	7,000	64	32	34	Y

Table 3. Index construction resource requirements on more than 2,000 RNA-seq datasets on REINDEER and as reported in the literature for the other indexes. The "Max Ext. Memory" column reports the maximal external (i.e., disk) space taken by the method. Execution times of methods are reported in wallclock hours. N/A stands for non-available, and means that the value was not reported in the article of the given method.

REINDEER's performances In this paragraph we assess REINDEER's performance using several datasets sizes and k sizes. In order to fairly compare the different runs, we selected a set of 2,512 datasets that have reads ≥ 31 bases among the 2,585 previously mentioned datasets.

We assessed the scalability of the REINDEER index (for raw counts) on 10 to 2,512 datasets (Table 5). The resources required grow roughly linearly with the number of distinct k-mers.

REINDEER's performances can be impacted by the chosen k-mer size. Intuitively, longer k-mer can lead to larger monotigs, thus to a reduction of REINDEER's index. We show REINDEER's behavior on several k values in Table 4. monotigs are longer with $k=31$ (124 bases on average, compared to 42 bases on average for $k=21$).

[8] github.com/kamimrcht/REINDEER/tree/master/reproduce_manuscript_results

k size	Millions of k-mers	Max Ext. Memory (GB)	Wallclock time (h)	Index size (GB)	RAM peak (GB)
21	∼ 3,755	6,500	39	55	51
31	∼ 4,426	4,700	32	47	37

Table 4. Comparison of REINDEER's performances (raw counts) using different k values. The datasets used for this experiment correspond to the 2512 datasets that contain sequences of size ≥ 31. 20 threads were used.

Number of datasets	Millions of k-mers	Max Ext. Memomy (GB)	Wallclock time (h)	Index size (GB)	Peak RAM (GB)
10	∼112	1.7	0.8	0.43	1.0
100	∼479	39	0.97	3.6	1.8
500	∼792	288	2.6	9.4	5.7
1000	∼1,265	894	6.6	18	9.5
2512	∼3,755	6,500	39	55	51

Table 5. Time of construction, index size after construction and memory peak during construction reported for REINDEER (raw counts, $k=21$) on increasing sizes of human RNA-seq datasets (subsamples of the 2,512 datasets benchmark).

Comparison with the Jellyfish hash table Jellyfish [19] is a k-mer hash table that can record k-mers counts for a single dataset at a time. Here, despite the comparison being unfair, we compare REINDEER and Jellyfish by estimating the space consumption of Jellyfish if it was to store integer positions (e.g. for row of the de-duplicated count-vector matrix) instead of counts.

The space usage of recording 4 billion distinct k-mers with Jellyfish is approximately $2^l \times \frac{2k-l+r+1}{8} = 92$ GB (according to Jellyfish manual, with $l = log_2(K/0.8)$, $r = \lfloor log_2(62) \rfloor - 1$, $k = 21$ and $K = 4 \cdot 10^9$). In comparison, REINDEER encoded monotigs and their association to a memory location using roughly 23 GB of memory, that is, a 75% space saving relative to Jellyfish.

6.2 Query

We then estimate the abundance of some test sequences in the 2,585 datasets using the REINDEER index. Queried sequences were randomly sampled from the RefSeq human transcripts, in query files of 10, 100, 1000 and 10,000 sequences in order to show the increasing query time. The mean sequence size over the batches is 3,341 nucleotides. Each batch size was repeated 10 times with random sampling, and results show the mean, max and min query wallclock times for each batch type (using 20 threads). Results are shown in Table 6. The query wallclock time is composed of a first step in which Reindeer loads the index (Index loading in Table 6), and a second in which k-mers from each query are asked to the REINDEER structure and the final results are computed and written on an ouptut file (Sequences query and I/O in Table 6). We used the same parameters as for the index construction (k=21,m=10, $-S$=75, with $-S$ being the minimum percentage of shared k-mers between the query and the dataset). All the experiments were performed after first warming the cache with the index. The peak of RAM consumption is also represented in Table 6, and is stable as it represents the size of the un-gzipped index in memory, plus an additional structure corresponding to the n count-vectors of size 2585 for the n k-mers of a query sequence.

This type of time measure is very architecture dependant. We only mention for the record that according to a recent benchmark [4], SeqOthello and Mantis allows to query 1000 sequence on a similar experiment in a minute only. SBT method would take an hour or so for the same task, while the more recent HowDe-SBT has a query time around 5 minutes.

Batch size	**Index loading (s, wallclock)** **mean**/min/max	**Sequences query and I/O (s, wallclock)** **mean**/min/max	**Peak RAM (GB)**
10 sequences	**867**/836/916	**92**/89/95	84.8
100 sequences		**149**/139/160	88.4
1,000 sequences		**394**/362/410	83.8
10,000 sequences		**1042**/986/1095	87.0

Table 6. Query times for different human transcripts batch sizes

6.3 Abundance matters: an example with oncogenes

Although the primal causes of individual cancers are often not known, there are genes which are known to be highly correlated with cancer progression, and involved with abnormal cell growth - these are termed oncogenes. Some oncogenes are normally expressed in healthy patients but can be over-expressed in cancer patients. With our REINDEER index we will demonstrate a simple *in silico* experiment involving abundances of oncogenes.

For genomic data, where reads are used to reconstruct the genome, k-mer-indexes can be used to query presence of alleles, but (modulo copy-number differences) all of them should be there at the genome-wide average abundance. In stark contrast, the situation is different for RNA-seq. There, every transcript may be expressed at a different abundance, and one might look for different patterns of expression. Given an index of an expression archive, it becomes possible to perform hitherto impossible queries. One can take an array of genes/transcripts, and collect an abundance signature across thousands of datasets, and then perform unsupervised clustering to look for patterns. One can pick a transcript of interest (e.g., from an oncogene) and compare the abundance of that and a control transcript, across all datasets, or maybe compare this query across tumours and normals (assuming appropriate metadata is available).

As proof of concept we selected four oncogenes well known for their role in breast cancers (ERBB2, FOXM, MYC, PIK3CA), as well as three tumor suppressor genes (BRCA1, PTEN, TP53) [24,28]. We launched REINDEER on the longest transcript of those genes. The transcript was split in 100bp long sequences, to be able to take into account matches on a few exons. We required at least 78% of the k-mers to be found in the split sequences. We kept the average of the counts returned by REINDEER. The distribution in each dataset, for which counts are reported, are displayed in Fig. 4. We emphasize that the counts are not normalized.

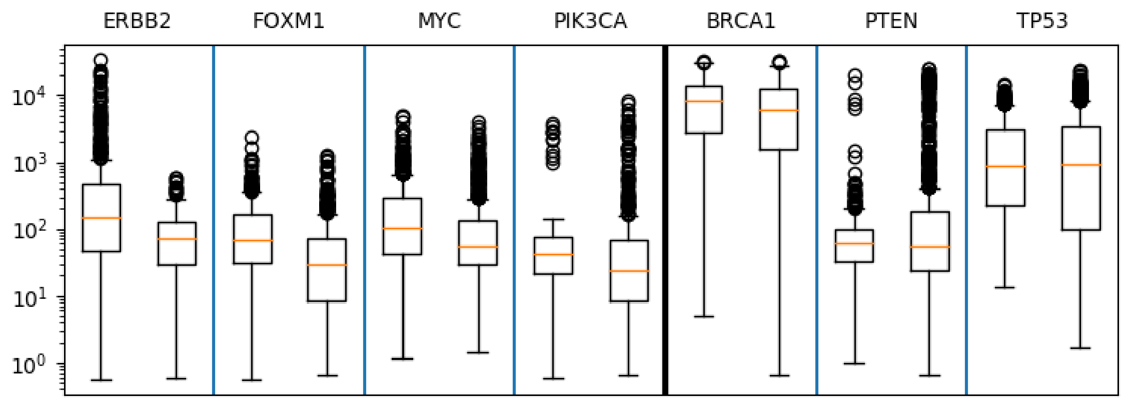

Fig. 4. Distribution of the average counts reported by REINDEER in the longest transcript, both in "cancer-related" datasets (left barplot in each panel) and in "non cancer-related" (right barplot) datasets for four oncogenes (ERBB2, FOXM1, MYC, PIK3CA) and three tumor suppressor genes (BRCA1, PTEN, TP53) considered. The median fold changes are respectively 2.7, 3.2, 2.3, 1.5, 1.3, 1, 0.97.

As expected the counts in oncogenes are higher in the "cancer-related" datasets compared to the "non cancer-related". This is particularly true for the ERBB2 gene where counts higher than 1,000 are only present in the cancer-related samples. The tumor suppressor genes have noticeably closer cancer/non-cancer medians abundances, but further investigations such as normalization of counts and improved metadata classification would be needed to draw clear conclusions.

Those experiments demonstrate that (even exact) presence/absence queries on such transcripts would not be sufficient to reveal important biological information. The level of expression brings valuable information as shown with the over-expression of oncogenes in cancer-related samples.

7 Conclusion

REINDEER is the first and currently only tool to enable the scalable indexing of k-mer counts in a collection of sequencing datasets. As a byproduct it allows quantification of arbitrary sequences within these datasets. REINDEER relies on the new concept of monotigs, on run-length encoding of count-vectors, optional lower-precision encoding of k-mer counts, and on the de-duplication of rows within the count-vectors matrix. To the best of our knowledge and despite their apparent simplicity, neither monotigs nor their greedy construction algorithm described here have previously appeared in the literature. Note that REINDEER does not encode k-mer counts as they exactly appear in a dataset, but instead relies on averaged counts per unitig as an approximation; thus it is not an index that records exact raw counts. REINDEER is able to answer presence/absence queries, although it is not particularly optimized to this task: other techniques such as HowDeSBT, Mantis, and SeqOthello, produce indices that occupy smaller space.

Several exciting future features can be envisioned for REINDEER. We plan on exploring index partitioning (i.e., constructing several REINDEER indices on subsets and merging query results) to accelerate REINDEER construction and aim towards indexing SRA-scale collections of datasets. In order to reduce memory usage at the expense of query speed, we are also currently investigating the possibility of performing on-disk queries of count-vectors.

Dataset quality control is also crucial for some applications such as RNA-seq, in order to draw correct biological conclusions. Removing low abundance k-mers before indexing appears to be necessary, as numerous artefactual k-mers are created due to sequencing errors. In our experiments with the 2,585 datasets benchmark, we used k-mer counts thresholds set in previous literature, but more adaptive thresholds would allow a better compromise between index size and loss of information. Another important aspect is the normalization of returned counts (taking into account dataset size), in order to accurately quantify transcripts, e.g. for the purpose of differential expression analysis.

Funding. This work was supported by ANR Transipedia (ANR-18-CE45-0020) for CM, DG, MS and RC, by INCEPTION (PIA/ANR-16-CONV-0005) for RC.

Acknowledgements. Part of the experiments presented in this work were made possible thanks to the EBI computing resources and the Institut Pasteur cluster. The authors would like to thank Antoine Limasset and Maël Kerbiriou for their immense help on BLight software, technical advice. We also thank Xue Haoliang for his tests on REINDEER, and Paul Medvedev for helpful comments on the manuscript.

References

1. Almodaresi, F., Pandey, P., Ferdman, M., Johnson, R., Patro, R.: An Efficient, Scalable and Exact Representation of High-Dimensional Color Information Enabled via de Bruijn Graph Search. In: International Conference on Research in Computational Molecular Biology. pp. 1–18. Springer (2019)
2. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic local alignment search tool. Journal of molecular biology **215**(3), 403–410 (1990)
3. Amid, C., Alako, B.T., Balavenkataraman Kadhirvelu, V., Burdett, T., Burgin, J., Fan, J., Harrison, P.W., Holt, S., Hussein, A., Ivanov, E., et al.: The european nucleotide archive in 2019. Nucleic acids research **48**(D1), D70–D76 (2020)
4. Bingmann, T., Bradley, P., Gauger, F., Iqbal, Z.: Cobs: a compact bit-sliced signature index. In: International Symposium on String Processing and Information Retrieval. pp. 285–303. Springer (2019)
5. Bradley, P., den Bakker, H.C., Rocha, E.P., McVean, G., Iqbal, Z.: Ultrafast search of all deposited bacterial and viral genomic data. Nature biotechnology **37**(2), 152 (2019)
6. Břinda, K., Baym, M., Kucherov, G.: Simplitigs as an efficient and scalable representation of de Bruijn graphs. bioRxiv (2020)
7. Buchfink, B., Xie, C., Huson, D.H.: Fast and sensitive protein alignment using diamond. Nature methods **12**(1), 59 (2015)
8. Camacho, C., Coulouris, G., Avagyan, V., Ma, N., Papadopoulos, J., Bealer, K., Madden, T.L.: Blast+: architecture and applications. BMC bioinformatics **10**(1), 421 (2009)
9. Chikhi, R., Holub, J., Medvedev, P.: Data structures to represent sets of k-long dna sequences. arXiv preprint arXiv:1903.12312 (2019)
10. Chikhi, R., Limasset, A., Medvedev, P.: Compacting de Bruijn graphs from sequencing data quickly and in low memory. Bioinformatics **32**(12), i201–i208 (2016)
11. Deorowicz, S., Kokot, M., Grabowski, S., Debudaj-Grabysz, A.: KMC 2: fast and resource-frugal k-mer counting. Bioinformatics **31**(10), 1569–1576 (2015)
12. Harris, R.S., Medvedev, P.: Improved Representation of Sequence Bloom Trees. bioRxiv (2018)
13. Holley, G., Wittler, R., Stoye, J.: Bloom Filter Trie: an alignment-free and reference-free data structure for pan-genome storage. Algorithms for Molecular Biology **11**(1), 3 (2016)
14. Janin, L., Schulz-Trieglaff, O., Cox, A.J.: Beetl-fastq: a searchable compressed archive for dna reads. Bioinformatics **30**(19), 2796–2801 (2014)
15. Leinonen, R., Sugawara, H., Shumway, M., Collaboration, I.N.S.D.: The Sequence Read Archive. Nucleic acids research **39**(suppl_1), D19–D21 (2010)
16. Li, H.: Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. arXiv preprint arXiv:1303.3997 (2013)
17. Limasset, A., Rizk, G., Chikhi, R., Peterlongo, P.: Fast and scalable minimal perfect hashing for massive key sets. arXiv preprint arXiv:1702.03154 (2017)
18. Lonsdale, J., Thomas, J., Salvatore, M., Phillips, R., Lo, E., Shad, S., Hasz, R., Walters, G., Garcia, F., Young, N., et al.: The genotype-tissue expression (GTEx) project. Nature genetics **45**(6), 580 (2013)
19. Marcais, G., Kingsford, C.: Jellyfish: A fast k-mer counter (2012)

20. Marchet, C., Boucher, C., Puglisi, S.J., Medvedev, P., Salson, M., Chikhi, R.: Data structures based on k-mers for querying large collections of sequencing datasets. bioRxiv p. 866756 (2019)
21. Marchet, C., Kerbiriou, M., Limasset, A.: Indexing De Bruijn graphs with minimizers. bioRxiv (2019). https://doi.org/10.1101/546309
22. Muggli, M.D., Bowe, A., Noyes, N.R., Morley, P.S., Belk, K.E., Raymond, R., Gagie, T., Puglisi, S.J., Boucher, C.: Succinct colored de Bruijn graphs. Bioinformatics **33**(20), 3181–3187 (2017)
23. Pandey, P., Almodaresi, F., Bender, M.A., Ferdman, M., Johnson, R., Patro, R.: Mantis: A fast, small, and exact large-scale sequence-search index. Cell systems **7**(2), 201–207 (2018)
24. Perera, R.M., Bardeesy, N.: On oncogenes and tumor suppressor genes in the mammary gland. Cold Spring Harbor perspectives in biology **4**(6), a013466 (2012)
25. Rahman, A., Medvedev, P.: Representation of k-mer sets using spectrum-preserving string sets. bioRxiv (2020)
26. Roberts, M., Hayes, W., Hunt, B.R., Mount, S.M., Yorke, J.A.: Reducing storage requirements for biological sequence comparison. Bioinformatics **20**(18), 3363–3369 (2004)
27. Solomon, B., Kingsford, C.: Fast search of thousands of short-read sequencing experiments. Nature biotechnology **34**(3), 300 (2016)
28. Song, X., Kenston, S.S.F., Zhao, J., Yang, D., Gu, Y.: Roles of foxm1 in cell regulation and breast cancer targeting therapy. Medical Oncology **34**(3), 41 (2017)
29. Tomczak, K., Czerwińska, P., Wiznerowicz, M.: The Cancer Genome Atlas (TCGA): an immeasurable source of knowledge. Contemporary oncology **19**(1A), A68 (2015)
30. Yu, Y., Liu, J., Liu, X., Zhang, Y., Magner, E., Lehnert, E., Qian, C., Liu, J.: SeqOthello: querying RNA-seq experiments at scale. Genome biology **19**(1), 167 (2018)