Journal of Bioinformatics and Computational Biology
Vol. 17, No. 5 (2019) 1940011 (15 pages)

DOI: 10.1142/S0219720019400110

Indexing *k*-mers in linear space for quality value compression

Yoshihiro Shibuya[*,†] and Matteo Comin[*,‡]

[*]*Department of Information Engineering*
University of Padua, via Gradenigo 6B, Padua, Italy
[†]*Laboratoire d'Informatique Gaspard-Monge (LIGM)*
University Paris-Est Marne-la-Vallée
Bâtiment Copernic - 5, bd Descartes
Champs sur Marne, France
[‡]*comin@dei.unipd.it*

Received 28 May 2019
Revised 25 June 2019
Accepted 15 July 2019
Published 20 November 2019

Many bioinformatics tools heavily rely on *k*-mer dictionaries to describe the composition of sequences and allow for faster reference-free algorithms or look-ups. Unfortunately, naive *k*-mer dictionaries are very memory-inefficient, requiring very large amount of storage space to save each *k*-mer. This problem is generally worsened by the necessity of an index for fast queries. In this work, we discuss how to build an indexed linear reference containing a set of input *k*-mers and its application to the compression of quality scores in FASTQ files. Most of the entropies of sequencing data lie in the quality scores, and thus they are difficult to compress. Here, we present an application to improve the compressibility of quality values while preserving the information for SNP calling. We show how a dictionary of significant *k*-mers, obtained from SNP databases and multiple genomes, can be indexed in linear space and used to improve the compression of quality value.

Availability: The software is freely available at https://github.com/yhhshb/yalff.

Keywords: *k*-mers; indexing; quality score; read compression.

1. Introduction

The compression of DNA is usually as simple as assigning a two-bit encoding to each of the four bases. This encoding achieves almost similar results to standard lossless compressors.[22] On the other hand, the quality values, produced by sequencing technologies, span a wider range of values, and when compressed, they can sum up to about 70% of the total space to encode a FASTQ file.[17] Quality values are usually encoded using the Phred system.[11] Quality values are often essential for assessing

‡Corresponding author.

sequence quality, mapping read to a reference genome, detecting mutations for genotyping, assembling genomic sequences, read clustering[8,9] and comparison.[32]

Quality scores are more difficult to compress due to a larger alphabet (63–94 in original form) and intrinsically have a higher entropy.[37] With lossless compression algorithms and entropy encoders reaching their theoretical limits and delivering only moderate compression ratios,[4] there is a growing interest to develop lossy compression schemes to improve compressibility further.

To further reduce the file sizes, Illumina proposed a binning method to reduce the number of different quality values from 42 to 8.[18] With this proposal, Illumina opened the doors for allowing lossy compression of the quality values. Another approach called P-Block[7] involves local quantization so that a representative quality score replaces a contiguous set of quality scores that are within a fixed distance of the representative score. Similarly, the R-Block[7] scheme replaces contiguous quality scores that are within a fixed relative distance of a representative score. Other lossy approaches improve compressibility and preserve higher fidelity by minimizing a distortion metric such as mean-squared-error or L1-based errors (Qualcomp and QVZ).[22,26] The drawback of lossy compression of quality values is that downstream analysis could be affected by the loss incurred with this type of compression. This could be the case for the above methods that process only the string of quality scores without considering the DNA sequence associated to the read. However, Refs. 17, 27 and 37 showed that quality values compressed with more advanced methods could achieve not only a better performance in downstream analyses than Illumina-binned quality values, but even better performance than the original quality values in some cases because these methods remove noise from the quality data.

The most promising methods are those using both sequence and quality information. The first method proposed in this class is in Ref. 19, where the authors applied the Burrows–Wheeler Transform (BWT) to the read collection in order to detect groups of suffixes starting with the same prefix (with size at least k). All quality values in a group are replaced with the mean value. This substitution is also referred to as smoothing, in which a series of quality values are replaced with a fixed quality, making the overall qualities more compressible. Leon[1] constructs a reference from the input reads in the form of a bloom filter compressed de-Bruijn graph and then maps each nucleotide sequence as a path in the graph. If a base is covered by a sufficiently large number of k-mers from the reference, its quality is set at a fixed high value. Among the most interesting tools, Quartz,[37] similar to Leon, relies on an external reference to decide if a given nucleotide is wrong or not. This reference database is implemented as a list of 2.5 Billion k-mers stored explicitly, which requires 24 GB when gzipped. Similarly, GeneCodeq[17] also has a list of k-mers as ground truth, but the algorithm involved during quality compression, i.e. smoothing, is more complex than Quartz. Each base has its associated error probability recalculated using a Bayesian framework and the smoothing takes place only if the new quality is greater than the old one. Both Quartz[37] and GeneCodeq[17] require a machine with at least 32 GB of RAM because of the size of the reference database.

In Ref. 35, we have proposed a method for quality value compression based on a single reference genome. However, the authors of Ref. 37 demonstrated that the use of significant k-mers coming from multiple genomes or SNP databases is beneficial for quality score compression and SNP calling. Hence, in this paper, we explore the use of multiple sources of mutations, i.e. SNP databases, for quality value compression.[a]

The most common procedure to obtain a reference list of k-mers from a set of sequences is by using a k-mer counting procedure. Most of the existing implementations rely on some hashing scheme. The hashing usually involves a simple application of a hash function to each k-mer to obtain a numeric value which is then used to access a hash table where the actual counters are kept. There exist efficient implementations of hashing functions which focus on optimizing and accelerating the generation of the hash value by using the previous computed hash like in Ref. 25 or Refs. 15 and 16. On the other hand, a very fast implementation of a hash table can be found in Jellyfish,[23] which uses a lock-free table allowing access from multiple threads if they do not collide in the same bucket for writing operations. Another approach is found in Ref. 31 and involves the use of minimizers, a type of Locality Sensitive Hashing to efficiently group similar k-mers into buckets.

The next step usually involves some sort of sorting and indexing for allowing fast retrieval of some particular k-mer (or its neighbors) and link this information to the position in the original sequence. This step is necessary whenever the counting procedure is the first step of a more complex pipeline. It is not so uncommon for some applications to index a sequence using the positions of each of its k-mers.[17,33] While this approach is advantageous when a proper counting of the k-mers is required, it is not necessary in general when the final goal is to support set queries and/or retrieval of a specific sequence or its position inside the original string. For this purpose, it is generally possible to strip a dictionary of k-mers of the counters to obtain a smaller representation. Unfortunately, even in the hypothesis of removing all the under-represented k-mers and all counters, the space required by a full table is still high. For example, LAVA[33] uses databases of k-mers extracted from dbSNP,[34] because of the large size of this database, it requires 60 GB of RAM. In fact, for a DNA sequence of length n, the potential number of k-mers is $n - k + 1$ each of length k for a total amount of space in the order of $O(k(n - k + 1)) = O(kn)$ bytes.

If the k-mer table has to be transmitted, a naive solution would be to compress it using a standard compressor such as `gzip` or `xz`. This procedure only works for moving the database from one place to another because to fully restore its functionality, it must be decompressed anyway. Another question is how efficient the standard compressors (or a two bit encoding) are in reducing the size of the table. What is the best way to compress a set of k-mers? Is it possible to use them for an optimal compression scheme removing redundancy and exploiting the peculiar structure of each k-mer and its relation with the others?

[a] A preliminary version of this paper was presented in Ref. 36

To answer this question in this work, we analyze a particular application of k-mer dictionaries, the compression of quality scores. In this work, we investigate how to construct a linear string, or set of strings, that contains all input k-mers. In particular, we will consider a special set of k-mers extracted from popular SNP databases like dbSNP[34] or Affymetrix Genome-Wide Human SNP Array 6.0. In the following sections, we present an application of a k-mers assembly procedure to reduce a dictionary of k-mers into a set of sequences guaranteed to contain all the k-mers. This procedure will allow us to store the whole dictionary in linear form, reducing the memory requirements from $O(kn)$ to $O(n)$ without losing information. For querying the resulting dictionary, we will index it with the FM-Index and use the FM-index for quality value compression.

2. Methods

The main insight in order to reduce the size of the dictionary is that most of the information carried by a k-mer stored explicitly is redundant. This intuition is easily explained by recalling the k-mer counting procedure itself. All the k-mers counted comes from a set of sequences and the counting procedure is only necessary to remove the wrong ones. There is no need to keep the k-mers explicitly stored to answer simple yes/no queries over their set. Given two consecutive k-mers, it is possible to reassemble them into a single $(k+1)$-mer, thus reducing the storage requirement by $k-1$ bases.

2.1. *Linear string dictionary through K-mers assembly*

The first step is to find a set of significant k-mers that will be used as a basis for quality compression. For example, in Ref. 37, they use a k-mer counting procedure on a set of real reads coming from 97 individual from the 1000 Genome Project. Then, a filter is applied to the k-mers whose count is below a certain threshold in order to prevent false positives (F.P.). Note that this procedure will cover only the mutations present in the input reads. Moreover, the filter is effective only if the datasets used have a very high coverage, leading to increased memory and time requirements for this step, and thus this approach will not scale.

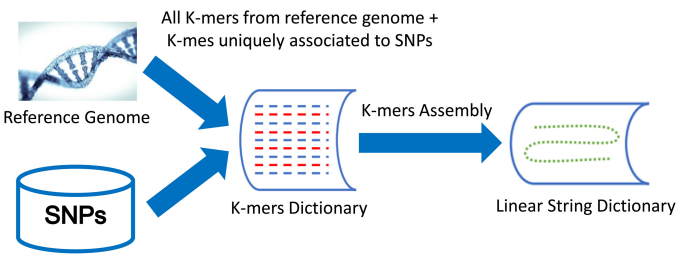

Fig. 1. Linear string dictionary construction from a set of k-mers extracted from a reference genome and an SNP database.

Instead, we directly use the information about mutations already present in SNP DB like dbSNP[34] and Affymetrix Genome-Wide Human SNP Array 6.0. We start from a reference genome and we extract all its k-mers. Then we would like to enrich this initial set with new k-mers associated to mutations for an SNP DB. These additional k-mers can be extracted from a list of known SNPs and a reference, like in LAVA,[33] where for each SNP, we can take the k-mers that overlap with the SNP with the reference allele replaced by the alternate. This will produce a set of k-mers uniquely associated to the SNPs, thus carrying important information to detect potential mutations that are crucial for quality compression.

The second step is to compress this k-mer dictionary into a linear string dictionary through k-mer assembly. This assembly step can be carried out on almost all dictionaries regardless of the tools used to generate them, leading to a linear sequence, or set of sequences, that contains all the input k-mer. This requirement is what makes this step different from a common *de-novo* assembly where the k-mers that do not align well with the graph are discarded leading to k-mers not present in the final assembled sequence. In general, this procedure will output more than one sequence because of the uniqueness of each k-mer in the resulting strings. Some redundant k-mers could allow for the concatenation of multiple contigs into one. The strings obtained are generally stored into a simple FASTA file where the identifiers can lead to unwanted memory usage especially if there are a very large number of short contigs. The memory requirements can worsen during the indexing of the FASTA file by adding information to account for the boundaries of each sequence. Simply concatenating each contig to each other produce $k-1$ spurious k-mers for each junction which might be a source of potential FPs during the application of quality value compression. The software used to perform this step is the greedy ProphAsm assembler[29] used in the Prophyle metagenomic suite.[2,3] In contrast to a similar software BCalm,[6] it doesn't stop at each discordant new k-mer producing less and longer contigs.

The problem of indexing the linear string dictionary in minute space, represented as a set of reference sequences, while providing full search capability has been widely studied and efficient data structure is now available. The data structure chosen for this purpose is the FM-Index[12,13] which is based on the BWT[5] of a sequence. The FM-index and its variants are now at the basis of many algorithms in the field of sequence analysis. For example, one of the most used tools for read mapping, BWA,[21] is based on this solution and it requires as input the FM-index of the reference genome.

For its simplicity, its relative easy installation and its widespread usage, we decided to use the BWA for indexing and querying our list of sequences. The FM-index will be used to search for k-mers. The procedure to retrieve the position of a k-mer is the enhanced *backward search* algorithm described in Ref. 20, which is also able to account for mismatches. In our case, we will search if a k-mer is present in the reference genome with up to one mismatch.

In our work, for compressing quality scores,[35] the use of BWA also had the collateral advantage of not requiring a separate indexed FASTA for compression by sharing the same indexed reference genome for reads alignments. In this work, the indexed sequences are not equivalent to a reference genome, but instead, they represent a set of k-mers that uniquely identifies known variants from dbSNP and Affymetrix SNP DBs.

2.2. *Quality value compression based on k-mers*

We evaluate the quality of the k-mer dictionary for the compression of quality values. This work extends our previous work on YALFF[35] where we presented a novel algorithm for compressing quality strings without losing precision during SNP calling. The main differences are in the reference sequence used for compression. In Ref. 35, we compress a FASTQ file always using a single reference genome (i.e. HG38), while in this study, we build a sequence dictionary based on a set of significant k-mers that appear in multiple genomes coming from the 1000 Genome Project, the dbSNP, and the Affymetrix SNP DBs. To keep the paper self-contained here, we briefly describe the YALFF algorithm.

The method employed by YALFF is a dictionary of k-mers to assess if a base of a read is correct. A read is decomposed into its constituent k-mers and they are searched in the reference dictionary. Given a base of the read, if all k-mers that cover the base are found in the dictionary, then we assume that the base is correct and thus we can modify (smooth) the corresponding quality score to a fixed high value, see Fig. 2 for an example. In the first version of YALFF, the database the k-mers were checked against is the well-known human reference genome `hg38` in the form of a BWA index. A summary of the algorithm is illustrated in Fig. 3. The threshold should be chosen depending on whether it is necessary to avoid as much distortion as possible or whether compression is considered much more important. As a rule of thumb, a higher threshold maintains more quality values unchanged, but this leads to an increase in the entropy of the output file. A good value found for this study was a quality value equal to the character (apex) which corresponds to a probability of error of 0.25119.

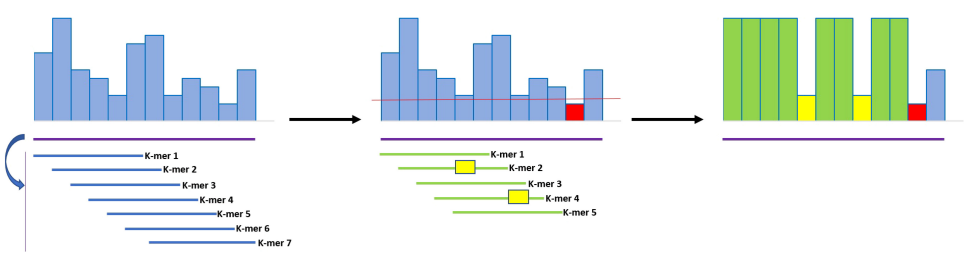

Fig. 2. An example of quality smoothing by YALFF including both mismatches with the k-mers DB and low quality values.

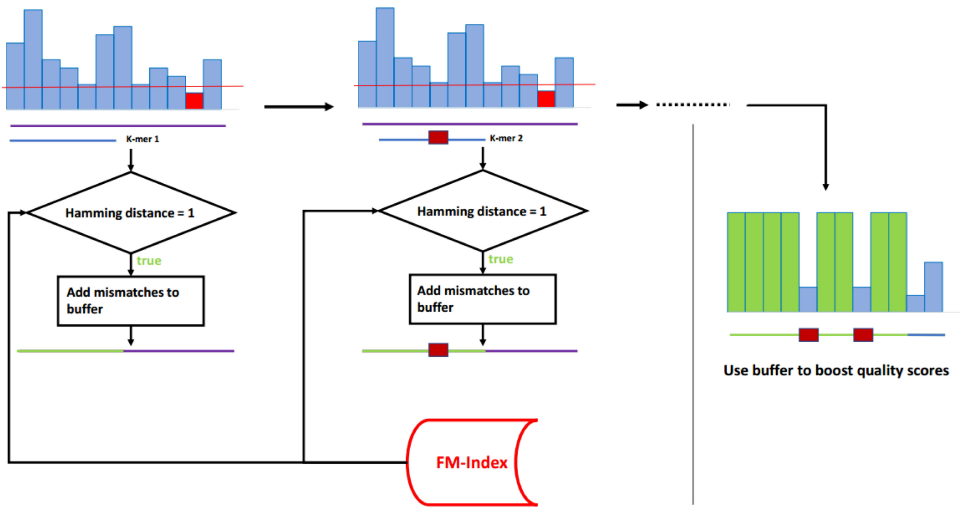

Fig. 3. Outline of quality compression with YALFF.

Quartz[37] is another FASTQ compressor which explicitly stores all the k-mers in a hash table. Quartz requires 25 GB of RAM whereas YALFF only need 5.7 GB. The k-mer counting procedure used to build the default database of Quartz was performed on multiple NGS reads coming from different individuals of the 1000 Genome Project and therefore contains more significant k-mers compared to the simple reference genome. Similarly, in LAVA,[33] the authors use two popular SNP DBs like dbSNP[34] and Affymetrix Genome-Wide Human SNP Array to select a set of k-mers that uniquely identify these mutations. Here, we will use these set of k-mers instead of a single reference genome.

All subsequent results use k equal to 32, the length used by other studies.[17,35,37] 32 has been chosen because the k-mers should be long enough to ensure that the number of all possible k-mers is much larger than the number of unique k-mers in the genome, so as to ensure incidental collisions between unrelated k-mers are rare. Also, k-mer length should ideally be a multiple of four, since a 4 bp length DNA sequence can be represented by a single byte and discriminating between forward and reverse strands is not important in this application (it is only required a yes/no response to a query). A 32-mer satisfies these constraints[17,37]; it is represented by a single 64-bit integer, with a relatively low probability of containing more than one sequencing error with Illumina sequences, as well as resulting in few k-mer collisions.

2.3. *Datasets, pipeline and parameters*

The dataset used in this study are real reads from the individual (NA12878) from the *1000 Genomes Project*.[10] For validation, we used an up-to-date high-quality genotype annotation generated by the Genome in a Bottle Consortium.[38] The GIAB gold standard contains validated genotype information for NA12878 from 14 sequencing

datasets with five sequencing technologies, seven read mappers and three different variant callers. To measure accuracy, we use loci in the SNP list which are also genotyped in the GIAB gold standard (so-called high-confident regions). This dataset has been widely used for benchmarking in other papers.[33,37] In line with previous studies, we use two sets of reads, namely SRR622461 (Low Coverage) and SRR622457 (High Coverage). For space limitations, the High Coverage dataset was downsampled.

One of the dictionary strings used as a reference is the human reference genome. The other k-mer databases are generated from dbSNPs[34] and Affymetrix Genome-Wide Human SNP Array 6.0 using as basis a reference genome. Although YALFF can be run on a normal laptop the ProphAsm assembler requires a powerful computer to load all the k-mers it needs to reassemble. For this reason, all tests were performed on a 14 lame blade cluster DELL PowerEdge M600 where each lame is equipped with two Intel Xeon E5450 at 3.00 GHz, 16 GB RAM and two 72GB hard disk in RAID-1 (mirroring).

The basic idea is to compare the compression results for the given dataset using both the original reference genome and the reassembled dictionary of k-mers of dbSNPs and Affymetrix Genome-Wide Human SNP Array with YALFF's algorithm. The performance evaluation of the compression procedure alone compares the number of retrieved SNPs from a compressed FASTQ to the ground truth associated to the original dataset, with each set of variants (stored in the output VCF file) are compared against the consensus set of variants. Each evaluation comprises three metrics: Precision, Recall and F-Measure. Once these parameters have been computed by using both dictionaries, they can be compared to each other to assess how much the additional variants affect the performances of the algorithm under evaluation. The genotyping pipeline is implemented as a single bash script which uses `bwa mem` for alignments, `bcftools` for SNP calling and `rtgplot` for evaluation. More details about the exact commands employed in the comparison can be found in the Supplementary Material.

3. Results

In this section, we compare different quality compression tools in terms of compression and genotyping accuracy. As reported in previous studies,[17,37] the most promising methods are those based both on sequence and quality values. For this reason, we compare YALFF with Leon in the following[1] and Quartz.[37] As YALFF can be used with different k-mer DBs, we also evaluated the impact of using different SNP repositories.

The following abbreviations are used in each table:

- None: The original FASTQ files without any modification.
- Quartz: The result of quality compressing the FASTQ files with Quartz and its standard k-mers dictionary.

- Leon: The result of quality compressing the FASTQ files with LEON.
- YALFF_hg: The result of quality compression of the FASTQ with YALFF when using the Human Genome as a reference database of k-mers.
- YALFF_Affy: YALFF when using the assembled database of significant k-mers from Affymetrix Genome-Wide Human SNP Array 6.0.
- YALFF_dbSNP: YALFF when using the assembled database of significant k-mers from dbSNP.
- YALFF_All: YALFF when using the assembled database of k-mers from the Human Genome, Quartz, dbSNP and Affymetrix Genome-Wide Human SNP Array 6.0.

3.1. *Effects on compression*

In this first test, we evaluate the ability of the different tools to improve the compressibility of quality values. The compression ratios is defined as $\frac{\text{uncompressed FASTQ size}}{\text{compressed FASTQ size}}$, where the compressed FASTQ file is first processed by the various tools and then compressed using `gzip`. The sizes of the uncompressed FASTQ files are 42 GB (low coverage) and 64 GB (high coverage).

The compression results are reported in Table 1.

First, we can observe that the processed FASTQ file is more compressible than the original file. This is expected as all methods tend to reduce the entropy of quality values. The compression ratio of YALFF is generally higher than Quartz and Leon, even when using just the Human Genome as a reference. Moreover, if more advanced k-mer dictionaries are used, then the compression ratio increases even further. As expected, the best compression ratio is obtained with YALFF_All, which uses the largest k-mer database.

The most evident advantage of using a compressed index is the huge reduction of the dictionary size, meaning less RAM used during execution. Quartz needs a powerful computer to run with large amount of memory, at least 32 GB of RAM, whereas YALFF requires about 6–7 GB of RAM, depending on the k-mer dictionary.

Table 1. Compression ratios for various tools on the two sets of reads (Low and High Coverage). The size of the k-mer dictionaries are also reported. The best results are in bold.

		Low cov.	High cov.
Method	Index [GB]	Compr.ratio	Compr.ratio
None	0	4.617	4.469
Quartz	19	6.925	7.115
Leon	6.3	7.098	7.256
YALFF_hg38	5.26	7.147	7.366
YALFF_Affy	**5.11**	7.133	7.342
YALFF_dbSNP	6.27	7.241	7.460
YALFF_All	6.7	**7.402**	**7.644**

Moreover, in these experiments, we used Quartz low memory mode that requires 32 GB of RAM, whereas the full version of Quartz needs 96 GB of RAM.

It must be noted that it is possible to reassemble all the unique k-mers in the Human Genome Reference into a more compact form. This solution is not shown here because it is exactly equivalent in terms of performances to the standard Human Genome dictionary which is more straightforward to compute, not involving the reassembly step. The only advantage is a reduced dimension of the k-mer dictionary from 5.26 GB to 5.09 GB.

The second advantage is the ability to store a large number of additional variations in linear space. For example, the total number of SNPs reported in the popular dbSNPs and Affymetrix SNPs are about 13M. We have shown that the Human Genome and all these 13M mutations can be indexed with as little as 6.7 GB.

3.2. *Effects on genotyping accuracy*

Another important metric of evaluation of the algorithms is to compare the number of retrieved SNPs from a compressed file to the ground truth. The latter comes with the original dataset and contains a set of variants validated using the genetic information of the parents whose child is the individual providing the source of the dataset. Each set of variants is stored in a VCF file analyzable with the software described earlier.

The benchmarking tools output the following values:

- True Positives (T.P.): All those variants that are both in the consensus set and in the set of called variants.
- False Positives (F.P.): All those variants that are in the called set of variants but not in the consensus set.
- False Negatives (F.N.): All those variants that are in the consensus set but not in the set of called variants.

These values are used to compute the following three metrics:

- Recall: This is the proportion of called variants that are included in the consensus set; that is, $R = \text{T.P.}/(\text{T.P.} + \text{F.N.})$,
- Precision: This is the proportion of consensus variants that are called by the variant calling pipeline; that is, $P = \text{T.P.}/(\text{T.P.} + \text{F.P.})$.
- F-Measure: The harmonic mean of precision and recall; that is, $F\text{-Measure} = 2 * (P * R)/(P + R)$.

Table 2 reports the results on the genotyping accuracy for various tools.

On the Low Coverage dataset, Quartz is the most aggressive in terms of raw numbers. Compared to the original dataset, it tends to add a very large number of True Positives and False Positives while greatly decreasing the False Negatives. This overall behavior would be more than adequate if well calibrated, but unfortunately,

Table 2. Comparison of the number of true positives, false positives and false negatives found by the SNP calling pipeline after compressing the input reads with the various methods.

	Low coverage			High coverage		
Method	T.P.	F.P.	F.N.	T.P.	F.P.	F.N.
None	2588159	219803	1493731	3165918	129697	962737
Quartz	2661218	237820	1420672	3221455	140892	907196
Leon	2278517	204803	1803366	2994023	127508	1134642
YALFF_hg38	2603620	221368	1478264	3187640	131072	941012
YALFF_Affy	2610302	222017	1471583	3196935	131420	931719
YALFF_dbSNP	2645734	222820	1436155	3247430	132323	881223
YALFF_All	2652457	224616	1429433	3256385	134552	872267

it tends to degrade the Precision. On this dataset, the algorithm YALFF_All can detect many True Positives, in line with Quartz, however with fewer false positives.

On the High Coverage data, the best results are obtained by YALFF_All with the largest number of True Positives. However, this does not come at the expense of a large number of False Positives, unlike Quartz. On these tests, Leon is not able to improve the number of True Positives with respect to the unprocessed input.

In order to better evaluate these raw numbers, we computed a number of metrics in Table 3.

For the Low Coverage dataset, Quartz has the highest F-Measure in line with YALFF_All. The high F-Measure of Quartz is obtained thanks to a high recall, which compensate for the lower precision. On the other hand, YALFF_All and YALFF_dbSNP show a more balanced behavior, whereas the high F-Measure is a consequence of the high value of precision and recall. In fact, these methods are the only two to improve the precision with respect to the original dataset.

On the High Coverage dataset, the results are even more significant, where the best performance for all three metrics is obtained by YALFF_All and YALFF_dbSNP. In summary, these two k-mer dictionaries appear to be the only ones that can improve the precision with respect to the unprocessed input and

Table 3. Comparison of precision, recall and F-Measure for the SNP calling pipeline after compressing the input reads with the various methods. Best results are in bold.

	Low coverage			High coverage		
Method	Precision	Recall	F-Measure	Precision	Recall	F-Measure
None	0.9217	0.6341	0.7513	0.9606	0.7668	0.8529
Quartz	0.9180	**0.6520**	**0.7624**	0.9581	0.7803	0.8601
Leon	0.9175	0.5582	0.6941	0.9592	0.7252	0.8259
YALFF_hg38	0.9216	0.6378	0.7539	0.9605	0.7721	0.8560
YALFF_Affy	0.9216	0.6395	0.7551	0.9605	0.7743	0.8574
YALFF_dbSNP	**0.9223**	0.6482	0.7613	**0.9608**	0.7866	0.8650
YALFF_All	0.9219	0.6498	0.7623	0.9603	**0.7887**	**0.8661**

increase the F-Measure. Moreover, YALFF can preserve a higher precision and a smaller number of false positives on SNP calling. This is very important in many medical applications where false positives should be avoided.

The reassembled dictionary of significant k-mers allows YALFF to inherit some characteristics of its rival. This can be easily noticed by looking at the increased recall. The larger the k-mer database, the better the recall, reaching its maximum with YALFF_All. Unfortunately, the precision does not follow a similar behavior, where YALFF_dbSNP is the best performer. The difference between the two smoothing algorithms account for the remaining differences, with Quartz being much more permissive in the definition of a modifiable quality value with YALFF being more conservative.

3.3. *Running time*

While the reassembly of multiple k-mers into a single linear reference allows for reduced memory consumption and the possibility to use widespread indexing algorithms such as the FM-Index, it also comes with reduced query speeds. The poor locality of succinct data structures, to which the FM-Index belongs, introduces a less efficient use of the cache. This may impact the execution time of YALFF. Leon is the fastest, requiring less than 10 h, but it is also the least accurate tool with the worst precision and recall. YALFF and Quartz are the most demanding compressors requiring about 30 h. However, the parameters of YALFF can be adjusted to reduce the running time down to 8 h. A comprehensive comparison of time and memory resources can be found in Ref. 35.

4. Conclusions and Future Work

This work has demonstrated the feasibility of combining a reassembly procedure with a string indexing algorithm to produce a very compact dictionary of k-mers which works as drop-in replacements whenever static k-mer hash tables are needed. Given a SNPs DB, like dbSNPs or Affymetrix SNPs, it is possible to find a set of k-mers that are uniquely associated to an SNP. This set of k-mers can be efficiently compressed into a string dictionary and used for quality value compression. These k-mer dictionaries are more informative than a single reference genome and they show better performance in terms of compression ratio and accuracy of genotyping, while keeping low memory requirements.

Future directions of research are the construction of a dynamic FM-Index with the ability to add and remove k-mers without recomputing the whole structure, another interesting problem is how to speed-up the search queries reducing cache misses. In the field of metagenomic read classification, most methods are based on k-mers indexes,[14,24,30] and only recently, the FM-index has been applied.[3] Similarly, the discovery of SNPs without mapping based on FM-index has been proposed only recently.[28] We believe that the use of FM-index will be beneficial in other alignment-free applications like pan-genomics.

References

1. Benoit G, Lemaitre C, Lavenier D, Drezen E, Dayris T, Uricaru R, Rizk G, Reference-free compression of high throughput sequencing data with a probabilistic de Bruijn graph, *BMC Bioinform* **16**:288, 2015.
2. Břinda K, Novel computational techniques for mapping and classifying Next-Generation Sequencing data. Ph.D. Thesis, Université Paris-Est, 2016.
3. Břinda K, Salikhov K, Pignotti S, Kucherov G, Prophyle: A phylogeny-based metagenomic classifier using the burrows-wheeler transform, *Poster at HiTSeq 2017*, 2017.
4. Bonfield JK, Mahoney MV, Compression of fastq and sam format sequencing data, *Plos One*, 2013.
5. Burrows M, Wheeler DJ, A block-sorting lossless data compression algorithm, Tech Rep, Digital Equipment Corporation, 1994.
6. Chikhi R, Limasset A, Medvedev P, Compacting de bruijn graphs from sequencing data quickly and in low memory, *Bioinform* **32**(12):i201–i208, 2016.
7. Cánovas R, Moffat A, Turpin A, Lossy compression of quality scores in genomic data, *Bioinform* **30**(15):2130–2136, 2014.
8. Comin M, Leoni A, Schimd M, Qcluster: Extending alignment-free measures with quality values for reads clustering, in Brown D, Morgenstern B (eds.), *Algorithms in Bioinformatics*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 1–13, 2014, ISBN 978-3-662-44753-6.
9. Comin M, Leoni A, Schimd M, Clustering of reads with alignment-free measures and quality values, *Alg Mol Biol* **10**(1):1–10, 2015.
10. Consortium TGP, An integrated map of genetic variation from 1,092 human genomes, *Nature* **491**(7422):56–65, 2012.
11. Ewing B, Hillier L, Wendl MC, Green P, Base-Calling of Automated Sequencer Traces UsingPhred. I. Accuracy Assessment, *Genome Res* **8**(3):175–185, 1998, doi: 10.1101/gr.8.3.175, http://genome.cshlp.org/content/8/3/175.
12. Ferragina P, Manzini G, Opportunistic data structures with applications, *Proc 41st Annual Symp Foundations of Computer Science*, pp. 390–398, 2000, doi: 10.1109/SFCS.2000.892127.
13. Ferragina P, Manzini G, Indexing compressed text, *J ACM* **52**(4):552–581, 2005.
14. Girotto S, Comin M, Pizzi C, Higher recall in metagenomic sequence classification exploiting overlapping reads, *BMC Genom* **18**(10):917, 2017.
15. Girotto S, Comin M, Pizzi C, Efficient computation of spaced seed hashing with block indexing, *BMC Bioinform* **19**(15):441, 2018.
16. Girotto S, Comin M, Pizzi C, Fsh: Fast spaced seed hashing exploiting adjacent hashes, *Alg Mol Biol* **13**(1):8, 2018.
17. Greenfield DL, Stegle O, Rrustemi A, GeneCodeq: Quality score compression and improved genotyping using a Bayesian framework, *Bioinform* **32**(20):3124–3132, 2016.
18. Illumina8bin, Quality scores for next-generation sequencing, Illumina Inc., Tech Rep, Illumina Inc., 2011.
19. Janin L, Rosone G, Cox AJ, Adaptive reference-free compression of sequence quality scores, *Bioinform* **30**(1):24–30, 2014.
20. Li H, Durbin R, Fast and accurate short read alignment with Burrows-Wheeler transform, *Bioinform* **25**(14):1754–1760, 2009.
21. Li H, Durbin R, Fast and accurate long-read alignment with Burrows-Wheeler transform, *Bioinform* **26**(5):589–595, 2010.
22. Malysa G, Hernaez M, Ochoa I, Rao M, Ganesan K, Weissman T, QVZ: Lossy compression of quality values, *Bioinfor* **31**(19):3122–3129, 2015.

23. Marçais G, Kingsford C, A fast, lock-free approach for efficient parallel counting of occurrences of k-mers, *Bioinform* **27**(6):764–770, 2011.
24. Marchiori D, Comin M, Skraken: Fast and sensitive classification of short metagenomic reads based on filtering uninformative k-mers, *Proc 10th Int Joint Conf Biomedical Engineering Systems and Technologies - Vol 3: BIOINFORMATICS, (BIOSTEC 2017)*, INSTICC, SciTePress, pp. 59–67, 2017.
25. Mohamadi H, Chu J, Vandervalk BP, Birol I, nthash: Recursive nucleotide hashing, *Bioinform* **32**(22):3492–3494, 2016.
26. Ochoa I, Asnani H, Bharadia D, Chowdhury M, Weissman T, Yona G, QualComp: A new lossy compressor for quality scores based on rate distortion theory, *BMC Bioinform* **14**:187, 2013.
27. Ochoa I, Hernaez M, Goldfeder R, Weissman T, Ashley E, Effect of lossy compression of quality scores on variant calling, *Brief Bioinform* **18**(2):183–194, 2017.
28. Prezza N, Pisanti N, Sciortino M, Rosone G, Detecting Mutations by eBWT, Parida L, Ukkonen E (eds.), *18th Int Workshop Algorithms in Bioinformatics (WABI 2018)*, Leibniz International Proceedings in Informatics (LIPIcs), Vol. 113, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp. 3:1–3:15, 2018, ISBN 978-3-95977-082-8.
29. ProphAsm, *Compressing k-mer Sets via Assembling Contigs*, https://github.com/prophyle/prophasm.
30. Qian J, Marchiori D, Comin M, Fast and sensitive classification of short metagenomic reads with skraken, *Commun Comput Inform Sci* **881**:212–226, 2018.
31. Rizk G, Lavenier D, Chikhi R, Dsk: k-mer counting with very low memory usage, *Bioinform* **29**(5):652–653, 2013.
32. Schimd M, Comin M, Fast comparison of genomic and meta-genomic reads with alignment-free measures based on quality values, *BMC Med Genom* **9**(1):41–50, 2016.
33. Shajii A, Yorukoglu D, William Yu Y, Berger B, Fast genotyping of known snps through approximate k-mer matching, *Bioinform* **32**(17):i538–i544, 2016.
34. Sherry ST, Ward MH, Kholodov M, Baker J, Phan L, Smigielski EM, Sirotkin K, dbsnp: The ncbi database of genetic variation, *Nucl Acids Res* **29**(1):308–311, 2001.
35. Shibuya Y, Comin M, Better quality score compression through sequence-based quality smoothing, *15th Annual Meeting of the Bioinformatics Italian Society Accepted in BMC Bioinformatics*, 2019.
36. Shibuya Y, Comin M, Indexing k-mers in linear-space for quality value compression, *Proc 12th Int Joint Conf Biomedical Engineering Systems and Technologies (BIOSTEC 2019) - Vol 3: BIOINFORMATICS*, Prague, Czech Republic, 22–24 February, 2019, pp. 21–29.
37. Yu YW, Yorukoglu D, Peng J, Berger B, Quality score compression improves genotyping accuracy, *Nat Biotechnol* **33**(3):240–243, 2015.
38. Zook JM, Chapman B, Wang J, Mittelman D, Hofmann O, Hide WA, Salit ML, Integrating human sequence data sets provides a resource of benchmark snp and indel genotype calls, *Nat Biotechnol* **32**:246–251, 2014.

Yoshihiro Shibuya received both his B.Sc. in Information Engineering and his M.Sc. in Computer Science Engineering at the University of Padua in 2015 and in 2017, respectively. Currently, he is studying Ph.D. in Computer Science at the Laboratoire d'Informatique Gaspard-Monge (LIGM) at the University of Paris-Est Marne-la-Vallée. His research interests are DNA sequence compression and efficient hashing techniques to speed up the analysis of large genomic databases.

Matteo Comin received both his M.Sc. degree in Computer Science Engineering and his Ph.D. from the University of Padua, Italy, in 2003 and in 2007, respectively. His research interests focus on the area of algorithms for computational biology. During his activity, he has been a research intern at IBM T.J.Watson Research Center twice where he developed several motif discovery systems for biological sequences, now part of their Bioinformatics and Pattern Discovery Toolbox.

He interests are in the computational methods for protein structural comparison and protein–protein docking prediction, and also in developing algorithms for next-generation sequencing. He has been a visiting researcher at University of Purdue (US) once and at Universitat Politcnica de Catalunya, Barcelona (Spain) thrice. Some of his results have been considered of great importance by IBM Research and three of his ideas are now U.S. patent. In 2007, he received from the University of Padova, the C. Offelli Award for the best young researcher. Since 2015, he is an Associate Professor at the University of Padova.