BIOINFORMATICS ORIGINAL PAPER Vol. 28 no. 5 2012, pages 628–635
doi:10.1093/bioinformatics/btr689

Sequence analysis

Advance Access publication December 13, 2011

Transformations for the compression of FASTQ quality scores of next-generation sequencing data

Raymond Wan[1,2], Vo Ngoc Anh[3] and Kiyoshi Asai[1,2]

[1]Department of Computational Biology, Graduate School of Frontier Sciences, University of Tokyo, 5-1-5 Kashiwanoha, Chiba-ken 277-8561, [2]Computational Biology Research Center, National Institute of Advanced Industrial Science and Technology (AIST), Tokyo 135-0064, Japan and [3]Department of Computer Science and Software Engineering, University of Melbourne, Victoria 3010, Australia

Associate Editor: Alex Bateman

ABSTRACT

Motivation: The growth of next-generation sequencing means that more effective and efficient archiving methods are needed to store the generated data for public dissemination and in anticipation of more mature analytical methods later. This article examines methods for compressing the quality score component of the data to partly address this problem.

Results: We compare several compression policies for quality scores, in terms of both compression effectiveness and overall efficiency. The policies employ lossy and lossless transformations with one of several coding schemes. Experiments show that both lossy and lossless transformations are useful, and that simple coding methods, which consume less computing resources, are highly competitive, especially when random access to reads is needed.

Availability and implementation: Our C++ implementation, released under the Lesser General Public License, is available for download at http://www.cb.k.u-tokyo.ac.jp/asailab/members/rwan.

Contact: rwan@cuhk.edu.hk

Supplementary information: Supplementary data are available at *Bioinformatics* online.

Received on August 23, 2011; revised on November 9, 2011; accepted on November 29, 2011

1 INTRODUCTION

Next-generation sequencing (NGS) offers new directions in genome science by allowing entire genomes to be sequenced at lower costs. Given its rapid growth, the problem of economically storing and quickly restoring sequencing data is becoming a concern for both researchers and operators of data centers.

The most notable examples of data repositories is the Sequence Read Archives (SRA) by the International Nucleotide Sequence Database Collaboration (INSDC) at NCBI, EBI and DDBJ, which helps in disseminating publicly funded data (Leinonen *et al.*, 2011). While the storing of raw data is infeasible, their hope is to be able to store at least the bases and their corresponding quality scores.

On the other hand, as the amount of data continues to rise, it is conceivable that the burden of storing such data will gradually shift to research laboratories, hospitals or even individuals. Effective means of storing sequencing data will be needed in these cases as well, even though the resources available will differ greatly.

*To whom correspondence should be addressed.

The above trends suggest that economical representation of sequencing data is important. In response, compression of DNA sequences has been an active research topic for many years. In contrast, relatively little attention has been devoted to quality scores. The set of quality scores is far larger than the four DNA nucleotides; this has the potential to make the problem more difficult than sequence compression. Whenever the economical representation of quality scores was taken into account, the scores were considered together with all other components. Hence, it is not clear how well the quality scores component can be compressed.

Note that the little attention given to quality scores does not mean that they are useless. In fact, they are slowly becoming a necessary part of many data analyses. They can be used to trim reads at either end [see the Galaxy tool (Blankenberg *et al.*, 2010) for an example] or be used for read mapping, as demonstrated by the `MAQ` software (Li *et al.*, 2008). Other future applications may also be possible.

In this article, we address the issue of economical representation of quality scores as a stand-alone component. This separation allows us to focus on the topic at hand, and does not limit us from combining our results with other works that are devoted to compressing DNA sequences alone. To have a clear understanding, and to supply different levels of economy trade-off, we break the process of economical representation into three independent and optional components: lossy transformation, lossless transformation and coding (or compression). Rather than advocating a single method for each component, we will explore various options.

2 BACKGROUND

NGS generates raw data as images from which the sequence bases are obtained. In practice, each *read* consists of a sequence of DNA bases, the estimated probabilities P that the respective bases was called incorrectly, plus other supporting information. The probabilities, or *error probabilities*, are stored as quantized integers that are the object of our study.

Standard representation: public repositories such as the SRA normally store reads in a human-readable format called FASTQ, as illustrated in Figure 1. In this format, an error probability P is first transformed to its respective *PHRED quality score* Q in a logarithmic manner so as to give more attention to the higher quality bases (Ewing and Green, 1998):

$$Q = -10 \times \log_{10} P.$$

#	FASTQ Data
1	`@SRR032209.2000 length=36`
2	`GTTGTGGCTGAGATGGGATGTAAACTTGANGANANN`
3	`+SRR032209.2000 length=36`
4	`B=A@?@BBB<285:<?8%3;#########!##!#!!`

Fig. 1. Sample NGS data in FASTQ format (SRA's `SRR032209`), with parts being shortened and numbered: (1) read identifiers; (2) sequence of bases; (3) '+' followed by optional comments; and (4) `Q-scores`.

Then, Q is truncated and limited to only integers from 0 to 93 (hence the maximum 93 includes all the values P of 5×10^{-10} or less). Lastly, each integer is offsetted by 33 to make it in the range from 33 to 126 which are ASCII codes of printable characters. This final format is referred to as the *Sanger-FASTQ format*, which is the *de facto* standard (Cock *et al.*, 2010) for representing the error probabilities. For the clarity of presentation, we use the special term `Q-scores` to refer to the scores in this format, and note that all of our input error probabilities are in this format.

Concise representations: representing each PHRED score in one byte as dictated by the Sanger-FASTQ format is not the best method in terms of storage economy. Data compression can be used to obtain a more space-saving representation.

Compression has been an important component of computational biology, as surveyed by Giancarlo *et al.* (2009). Generally, short DNA sequences are compressed by encoding them with respect to other data such as a larger genome through 'edit operations'—instructions that indicate the mismatches needed to map the short sequences to the genome [see, for example, Daily *et al.* (2010)].

In recent years, NGS data in FASTQ format has attracted some attention from the compression field. Tembe *et al.* (2010) encode both the DNA sequences and the `Q-scores` *together*, treating each distinct pair of DNA base and its respective `Q-score` as a new symbol, and then applying Huffman coding (Huffman, 1952). Deorowicz and Grabowski (2011) encode all parts of the FASTQ data through a combination of creating independent blocks, `LZ77` compression (Ziv and Lempel, 1977), run-length coding and Huffman coding. These two compression schemes are classified as *lossless* since the compressed data can be decoded to get back the original data, without any change.

In addition to lossless, there exists *lossy compression*, for which the data can be restored, not as the original, but with some changes that can be regarded as acceptable for applications. Lossy methods can be considered for quality scores—an option that has been advocated by others (Leinonen *et al.*, 2011). For example, Kozanitis *et al.* (2011) encodes DNA sequences with respect to a reference genome. The `Q-scores` are compressed using a combination of taking the gaps between adjacent scores and lossy compression. They noted that the gaps between adjacent scores in a sample dataset were usually small (often zero). Because of this, they randomly permuted `Q-scores` so that they were either just above or just below the previous score. Amortized over the entire dataset, the effect of such randomization balances out. Using SNP calling as an example, they showed that such a lossy transformation has a minimal effect on downstream applications. More recently, Hsi-Yang Fritz *et al.* (2011) defined 'quality budget' as a filter to determine what `Q-scores` are stored. Basically, only the scores associated with sequence bases of interest are retained.

A common theme of the above described methods is that when coding, they treat the `Q-scores` or the gaps between adjacent `Q-scores` as plain symbols, without considering the integral values of the symbols. For Huffman and `LZ77` methods, symbols (or sequence of symbols) are processed so that those with higher frequencies will be assigned shorter code words. We refer to this way of compression as *by-symbol*; we will also consider *by-value* compression methods—methods that assign shorter code words to lower integers, regardless of their frequencies.

3 DATA ANALYSIS

We first analyze some properties of `Q-scores`. Three datasets, summarized in Table 1, were randomly selected from SRA for that and the subsequent experimentation. The last two columns of the table shows the size of the `Q-scores` and the entire FASTQ data (with both the meta-information and sequences) in MiB. The values in parentheses is their compression ratio as a percentage of the original data when the standard tool `gzip` is used. The *compression ratio* is defined as the ratio between the size of the compressed data and the size of the original data, expressed in bits per quality score.

Following the lead of Kozanitis *et al.* (2011) on using score gaps, in addition to statistics for the population of reads and the population of individual `Q-scores`, we also gather additional statistics for another form of `Q-scores`, produced through *gap transformation*. We transform the sequence $(q_1, q_2, q_3, \ldots q_{n-1}, q_n)$ of n `Q-scores`, in the following manner. First, they are translated to the equivalent form of score gaps $(q_1-33, q_2-q_1, q_3-q_2, \ldots q_n-q_{n-1})$. These gaps are in the range of $[-(|\Sigma|-1)..(|\Sigma|-1)]$, with $2|\Sigma|-1$ different values, where Σ is the set of all possible `Q-scores` (hence $|\Sigma|=94$). However, we further transform each of the gaps to a positive integer by using the bisection $(0, 1, -1, 2, -2, \ldots |\Sigma|-1, -(|\Sigma|-1)) \longrightarrow (1, 2, 3, \ldots, 2|\Sigma|-1)$. Each resulting value will be further referred to as a `T-gap`.

Some important statistics of `Q-scores` and `T-gaps` are presented in Figure 2. From the figure, we underline the following three properties, which will be explored in the next section:

Absolute `Q-score` bounds are rarely reached: according to Figure 2a, a large number of reads have their own local minimal `Q-score` different from the absolute minimum $\Sigma_{\min}$ of Σ, and in many cases, with a gap of at least 20—a considerable amount if we consider that the number of possible distinct score values is

Table 1. Three SRA datasets employed in this article

Accession	Species	No. of reads ($\times10^6$)	Read length	Size (MiB)	
				`Q-scores`	Total
`SRR032209`	*M. musculus*	18.8	36	662.8 (40.9%)	3488.2 (21.6%)
`SRR070788_1`	*H. sapiens*	24.8	100	2393.2 (33.3%)	8092.1 (24.9%)
`SRR089526`	*H. sapiens*	23.9	48	1115.7 (33.7%)	4814.2 (22.4%)

The last two columns give the size of the `Q-scores` and the original FASTQ data in MiB. Percentages in parentheses indicate the compression ratio when `gzip` is applied to them.

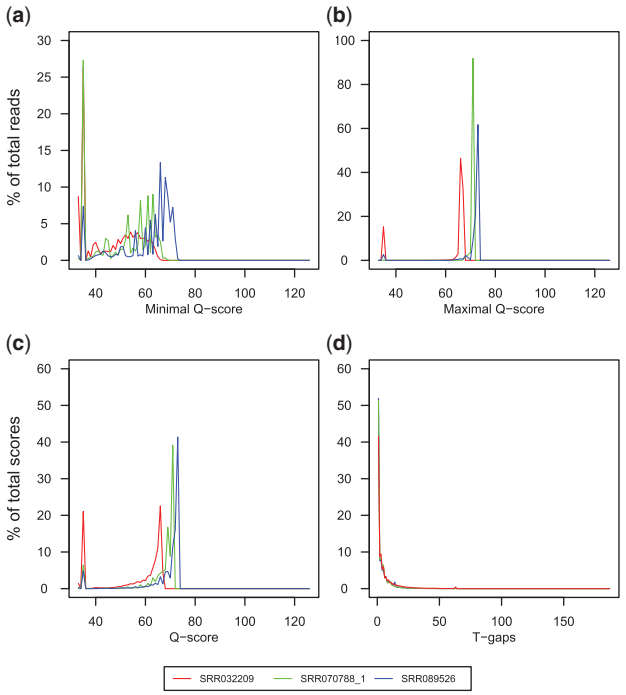

Fig. 2. Some statistics for each of the three described datasets: distribution of minimal and maximal `Q-scores` over the population of reads (**a** and **b**), and distribution of `Q-scores` (**c**) and `T-gaps` (**d**).

just 94. On the other hand, Figure 2b shows that the majority of reads have their own maximal scores being much lower than the respective absolute value $\Sigma_{\max}$ of Σ. In fact, for most of the reads, the maximal score lies approximately in between 65 and 75. This is far less than $\Sigma_{\max}$ (126)—a consequence of the limitations of current NGS technologies.

Non-uniform distribution of `Q-scores`: Figure 2c shows that `Q-scores` are non-uniformly distributed, with a large number of values having very low frequencies, and a few values that occur with exceptionally high frequencies. There also appears to be a bimodal distribution, with most values centered around either 33 or 75.

Near-geometric distribution of `T-gaps`*, but not of* `Q-scores`: judging by the shapes of the distribution curves in Figure 2d, we can speculate that the `T-gaps` for each dataset have a distribution which is close to geometric, where the values of `T-gaps` negatively correlate with their frequencies. Unfortunately, that is not the case for the respective curves for `Q-scores`, as can be seen from Figure 2c. In fact, there is neither a positive nor a negative correlation between the `Q-score` values and their frequencies.

4 TRANSFORMATION AND COMPRESSION

Our investigation follows the diagram in Figure 3. The FASTQ data is first decomposed to separate the `Q-scores` data from the rest. The `Q-scores` then can go through the lossy and lossless transformation and finally the encoding processes to reach point **(A)**, where the compressed representation is attained, and hence

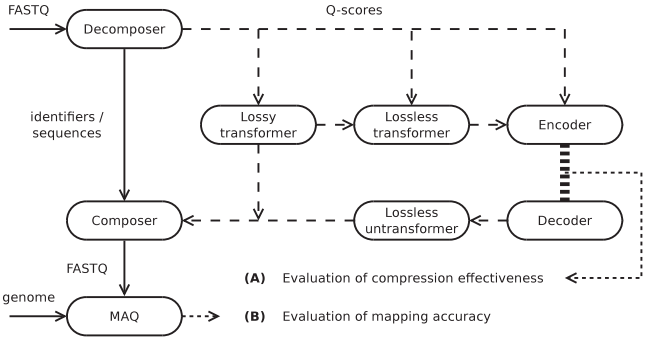

Fig. 3. Workflow of our investigation. Dashed arrows depict a representation of `Q-scores` alone.

compression ratio can be measured. From this point, the reverse process can take place. The decoding and lossless untransforming processes are applied to the compressed data to reproduce the `Q-scores`, which might or might not be the same as the original `Q-scores` data, depending on whether or not any lossy transformation has been selected. The 'Composer' will then recreate the FASTQ-formatted data. After that, at point **(B)** in the diagram, we can evaluate the effect of lossy transformations (if any) by mapping these data to a reference genome using the `MAQ` software (Li *et al.*, 2008).

4.1 Blocking

Most of the compression schemes collect some information over the input data before encoding. They normally segment the input stream into non-overlapping blocks and process each block independently in order to limit and optimize peak memory usage. To accommodate that, we explicitly perform the blocking of `Q-scores` right after they have been produced by the 'Decomposer' component, before any of the lossy and lossless transformations. We define the *block size* k as the number of *reads* instead of the number of elementary `Q-scores` to ensure that no read spans across block boundaries, which in turns guarantees that blocks could be compressed or decompressed independently from each other.

In general, block creation incurs only a small cost in the form of local block information at the beginning of each block—the *block header*. However, as will be seen, some compression schemes might add much more data to the block header, and the relative volume of this data is higher with smaller k. To keep these schemes effective, it is essential to set k large enough. On the other hand, smaller k means more flexibility in decoding, and $k=1$ allows random access to each block in the compressed data—a plus or even a critical feature for some applications. Throughout our investigation, we will consider both small and large block sizes in order to accommodate a potential diversity of applications.

4.2 Lossy transformation

As stated earlier, `Q-scores` is an irreversible quantization of some original (floating point) probabilities, and can have $|\Sigma|=94$ distinct values. The quantization can be viewed as the process of partitioning the probability interval $[0,1]$ into $|\Sigma|$ subintervals, or *bins*, so that the lengths of the bins grow in a logarithmic manner (the closer to the left of $[0,1]$, the shorter the bins). Probabilities falling into the same bin are deemed to share the same quantized `Q-score`.

For clarity, we will refer to this method of quantizing the error probabilities as `Standard`.

As our aim is space saving, it is natural then to think about transformations that are more aggressive than `Standard` with regard to reducing the number of partitioned bins. For that, we introduce three lossy transformations that are each defined by a unique way to partition the probability interval into bins. We abuse our earlier terminology by unifying their parameters as $|\Sigma|$, which represents the number of distinct bins. After partitioning, each bin is associated with the lowest `Q-scores` that falls into the bin. Note that since the mapping from bin order to the associated `Q-score` is determined by the transformation, we can actually store the bin orders instead of the associated `Q-scores`.

`UniBinning`: in this transformation, the interval $[0,1]$ is uniformly divided into $|\Sigma|$ equal bins. For example, if $|\Sigma|=5$, the leftmost bin would have the error probabilities from 0.8 to 1.0, and all `Q-scores` in this bin will be transformed to 33.

`Truncating`: the `Standard` transformation is characterized by the logarithmic partitioning of the error probability interval into 94 bins. Here, we apply the same partitioning scheme, with the only exception that a number of rightmost bins is combined into one single bin. That number is set to be $94-|\Sigma|+1$ to accommodate the parameter $|\Sigma|$. To put it simply, `Truncating` is the same as `Standard`, except that all `Q-scores` higher than $94-|\Sigma|$ are deemed to be $|\Sigma|$.

`LogBinning`: in this transformation, we apply the same logarithmic partitioning as in `Standard`, but using $|\Sigma|$ instead of 94 bins. That is equivalent to grouping a fixed number of `Standard` bins into a new bin. For example, with $|\Sigma|=19$, each bin has five `Q-scores`. However, the rightmost bin has the four last `Q-scores` (that is, from 123 to 126 inclusive), which is assigned the `Q-score` of 123.

The three new lossy transformations aim to improve the space savings at the cost of some loss in mapping accuracy when the transformed values are used. To evaluate this loss, we employ the `MAQ` tool (Li *et al.*, 2008). This tool maps reads from a FASTQ dataset to the corresponding genome. Besides being able to obtain the overall percentage of reads that could be mapped, a mapping quality score is returned for each successfully mapped read.

In our case, we use the number of unambiguously mapped reads using the unaltered `Q-scores` as our baseline. We then repeat the mapping process after substituting the standard `Q-scores` with those that have been lossy transformed. We consider the mapping of the read to be successful if the position, strand and chromosome are identical to that of the baseline. Moreover, the read's mapping quality score may have decreased but it must be non-zero—this indicates it was unambiguously mapped. We define the percentage of reads mapped to the baseline as the *relative mapping accuracy* for that transformation to indicate the quality of the transformation.

Figure 4 compares the relative accuracy of different lossy transformations for one of our datasets. It can be seen that all three lossy transformations can achieve nearly perfect relative accuracy when $|\Sigma|$ is about 30 or higher. However, as expected, the relative accuracy decreases as $|\Sigma|$ drops.

Of the three transformations, `LogBinning` seems to be the best as it is the only one that retains high relative accuracy even with a very low value of $|\Sigma|$. Indeed, with $|\Sigma|=5$, it achieves the relative accuracy of around 99% —the level that the other two can do only at $|\Sigma|$ of 20 or so. Of the three transformations, given that `LogBinning` resembles `Standard` the most, its superiority is understandable. In the rest of our investigation, `LogBinning` with varying values of $|\Sigma|$ will be chosen to represent various levels of lossy transformation.

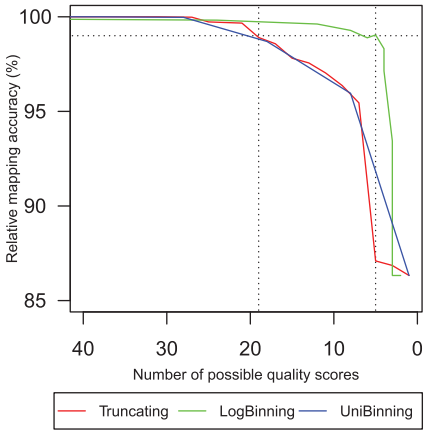

Fig. 4. Relative mapping accuracy of the three lossy transformations, measured for the dataset `SRR032209`. The 100% baseline employs standard `Q-scores`, resulting in 12.1 (out of a total of 18.8) million reads being mapped successfully by `MAQ`. The horizontal dotted line shows where relative mapping accuracy is 99%. The two vertical dotted lines indicate the lowest values for $|\Sigma|$ at which this accuracy is still attainable ($|\Sigma|=19$ for `Truncating` and `UniBinning`; $|\Sigma|=5$ for `LogBinning`).

4.3 Lossless transformation

The lossy transformations benefit compression by means of reducing the number of distinct quantized quality scores. On the other hand, a *lossless transformation* is a reversible mathematical mapping over the quantized scores so that the scores can be untransformed back without any change. A transformation can potentially improve compression ratio if it yields a distribution of transformed values that is more suited for compression than the untransformed ones. Judging by the analysis in Section 3, we investigate the following three lossless transformations.

`MinShifting`: this method converts each `Q-score` q to $q-q_{\min}$, where $q_{\min}$ is the minimal score in the respective block. This requires $q_{\min}$ to be stored in the block header.

`FreqOrdering`: this technique remaps the `Q-scores` that appear in the block to the set of contiguous positive integers starting from 1 in the order of decreasing frequency. The reverse mapping needs to be stored in the header.

`GapTranslating`: for a block of n quantized scores, this transformation is the combination of taking gaps between adjacent scores and translating these gaps to positive integers, exactly as described earlier in Section 3.

It is clear that all the three lossless transformations have a common theme—to map the quantized scores to lower values. Naturally, they will mainly benefit the by-value compression schemes. However, as a special case, the `GapTranslating` transformation can potentially affect the by-symbol schemes, since

it dramatically changes the symbol set as well as the distribution curves. Based on these definitions and the distribution curves of Figure 2, it seems unnecessary to apply more than one lossless transformation at any one time.

4.4 Compression

For clarity of presentation, we assume that the input block of the coding process, that is, the *encoding stream* is $X = x_1 x_2 \ldots x_n$. These integers, or *symbols*, do not necessarily have to be standard `Q-scores` due to the lossy and lossless transformations. More generally, they are drawn from a known set of ℓ distinct integers $\Omega = \omega_1, \omega_2, \ldots, \omega_\ell$, which can be easily computed for each combination of lossy and lossless transformations.

Four groups of compression methods are considered, which differ in the algorithmic complexity of the encoding process. This complexity largely correlates with the amount of computing resources needed to perform coding and, in many cases, decoding. We select a few representatives for each group.

Static codes: a code of this group associates each positive number with a distinguished and fixed code word. Hence, the code word for a symbol $\omega \in \Omega$ is fixed in advance, regardless of neighboring symbols or the overall frequency of that symbol in the encoding stream. Each static code is designed for a particular distribution of the integer numbers, and is in fact a minimum-redundancy code for that distribution. The simplest code is `unary`, which encodes a value $\omega \geq 1$ as a sequence of $\omega - 1$ one-bits followed by a single zero-bit. Another code, `gamma`, which is more effective than `unary` for large numbers, codes a value $\omega \geq 1$ in two parts: the value of $\upsilon = \lfloor \log_2 \omega \rfloor$ in `unary`, and υ-bit binary representation of $\omega - 2^\upsilon$. For larger numbers, a reasonable code is `delta`, which is similar to `gamma`, but with υ coded using `gamma` rather then `unary`.

Parameterized codes: these codes are not context-free, as they each employ a parameter which is derived from making a pass over the encoding stream. The simplest code of this family is `binary`, that employs the parameter $b = \max(x_1, x_2, \ldots, x_n)$. It uses the binary representation of $x - 1$, in either $\lfloor \log_2 b \rfloor$ or $\lceil \log_2 b \rceil$ bits, as the code word for $x \in X$.

One interesting code of this group is `golomb` (Golomb, 1966), which uses the parameter b, computed from n and $N = \sum_{i=1..n} x_i$. It codes a value $x \in X$ as the pair $(x/b,\ x \mod b)$. Gallager and van Voorhis (1975) show that `golomb` is a minimum redundancy code when the distribution of the alphabet Ω is geometric. That makes `golomb` attractive for compressing `T-gaps`. A related coding scheme that we also considered is `rice` (Rice, 1979), that differs from `golomb` only by forcing b to be a power of 2.

When the input stream has some sort of clustering, interpolative coding (`interp`) by Moffat and Stuiver (2000) is a good choice. It first converts the input stream X to the equivalent sequence of cumulative sums X', and separately codes the first and last elements of the later. Then, it codes the middle element of X' using `binary`, whose parameter is computed from the values of the first and last elements. It continues recursively to the left and right halves of X'.

Minimum redundancy codes: minimum redundancy coders employ the minimum number of bits to encode the data stream based on the actual distribution of the input symbols. More frequent symbols are assigned shorter code words in an optimized manner. Huffman code (`huffman`), invented by Huffman (1952), is an excellent representative of this family due to its simplicity and efficiency.

Complex codes: under 'complex codes', we refer to compression systems (as opposed to single codes) that do not generate code words for single symbols, but for sequences of symbols instead. They normally spend a considerable amount of time to build the set of symbol sequences and then apply some form of minimum redundancy coding as the final stage. Representatives include `zlib` (`http://www.zlib.net/`)—the library version of the widely available compression system `gzip`, and `libbzip2`—the library version of `bzip2` (`http://www.bzip.org/`).

It can be easily seen that the above four coding families were presented in the increasing order of algorithmic complexity, at least in terms of the encoding process. Moreover, the first two families have the by-value nature, while the other two are by-symbol. It can be expected that as we move down the above list of four families, we can get better compression ratios, and get poorer encoding times. And it is no doubt that the further we progress down this list, the method will request more internal memory to perform their work.

5 RESULTS

This section reports the main experiment results for the dataset `SRR032209`. Additional results, including those for the dataset `SRR070788_1` are provided in the Supplementary Material. All experiments were conducted on a set of 2.53 GHz 8-Core Intel Xeon E5540 with 12 GB of RAM and hyper-threading.

5.1 Space

Effects of lossless transformations: Figure 5 demonstrates the compression effectiveness, as compression ratios, achieved by the described lossless transformations with various block sizes.

The most noticeable feature in the figure is the near immobility of the by-symbol curves, and the variability of the by-value curves across the four graphs. As anticipated, in general, the lossless transformations improve compression ratio for the by-value methods, and have almost no effects on the by-symbol compression schemes. Moreover, all the by-symbol compression schemes are not effective when the block size k is small.

The effect of lossless transformation on the by-value compression schemes is tremendous. In fact, these compression schemes are unsuitable for the original `Q-scores`, as shown in Figure 5a. It is understandable, as the minimal `Q-scores` of 33 is already a large value for these methods which reserve short code words to small values. While we can take off 33 from all `Q-scores` before encoding to improve compression effectiveness, that simple operation is not considered here because it cannot do better than the `MinShifting` transformation.

Figure 5b and c clearly show the difference in performance between `MinShifting` and `FreqOrdering`. The former has a very low cost in terms of block headers, and performs well when $k = 1$, but gets worse when k grows since the impact of the minimal value lessens. On the contrary, while `FreqOrdering` spends more for the block headers, the absolute cost is stabilized when k is large enough, making `FreqOrdering` the clear winner. It should

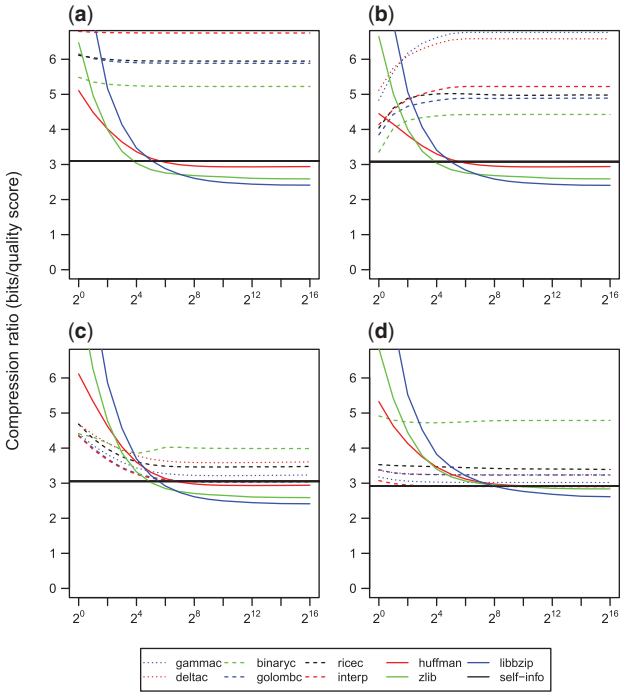

Fig. 5. Compression effectiveness for `SRR032209` using various block sizes and different lossless transformations: (**a**) No transformation; (**b**) `MinShifting`; (**c**) `FreqOrdering`; (**d**) `GapTranslating`. The x-axis represents block size. As reference, the zero-order self-information is indicated as solid black lines.

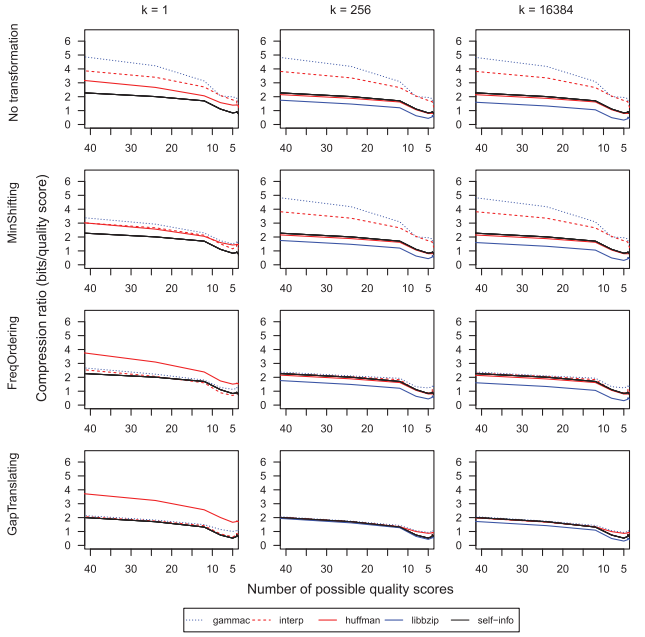

Fig. 6. Compression effectiveness achieved with `LogBinning` for `SRR032209`, with k and $|\Sigma|$ varying on the horizontal, and lossless transformations varying on the vertical dimensions. Black lines represent the self-information, which is unaffected by block size.

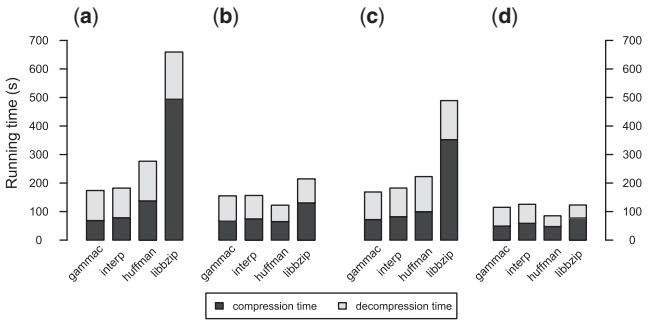

Fig. 7. Compression and decompression (including transformation) times averaged over three trials for `SRR032209`. Lossless transformation `GapTranslating` and no lossy transformation for (**a**) $k=1$ and (**b**) $k=256$; and with no lossless transformation and lossy transformation `LogBinning` at $|\Sigma|=5$ for (**c**) $k=1$ and (**d**) $k=256$.

be noted, however, that the combination ($k=1$, `MinShifting`, `binary`) is the simplest from Figure 5b and c, but achieves a very impressive compression ratio. This combination can be a good choice when random access is a need.

Finally, Figure 5d shows that `GapTranslating` is the best overall lossless transformation for the by-value coding schemes, with the only exception of `binary`. It is interesting to see that the compression ratio is almost stable across different k. Moreover, the compression ratio attained by `interp` is very close to that achieved by the by-symbol compression schemes with large k. The solid black lines indicate the zero-order self-information for each transformed dataset—a limit that can be improved upon through block creation or more complex models.

Based on Figure 5, we will further consider four representative coding schemes: `gamma`, `interp`, `huffman` and `libbzip2`.

Effects of lossy transformations: compression ratio achieved with different levels of granularity of the lossy transformation `LogBinning` for the dataset `SRR032209` is shown in Figure 6. As one would expect, the more stringent the lossy transformation, the better the compression ratio. With $k=1$, the combination of (`GapTranslating`, `interp`) is the clear winner. With $k=256$ or 16 384, although `libbzip2` with no lossless transformation is the best, `interp` and `gamma` with `GapTranslating` can achieve fairly close compression ratios.

In general, when a relative mapping accuracy of 99% is acceptable, the lossy transformation `LogBinning` with $|\Sigma|=5$ can be applied. At this setting, various combinations of lossless transformations and coding methods can achieve compression ratios of around 1 bit per quality score—a dramatic progress in comparison to the level of 2.5 bits per quality score achieved in the absence of any lossy transformation.

5.2 Speed

When our main goal is space economy, processing (transformation and coding) time is also important. Both compression and decompression times are of concern—throughout the day, compression is initiated by a data archiver many times, whereas decompression is performed just a few times, but by many users.

Figure 7 samples the time for the four representative cases across block sizes and coding methods, with either the lossless transformation set at `GapTranslating` or the lossy

transformation fixed at LogBinning. The figure shows that when $k=1$ the order of both compression and decompression time reflect well the algorithmic complexity of the coding methods. However, while it takes less time to compress than to decompress with the by-value methods, the order is reversed for the by-symbol methods. In particular, the compression time of libbzip2 is dramatically higher than that of all the other methods.

When k changes from 1 to 256, all running times improve with the times for libbzip2 improving the most due to the higher running times for $k=1$. With $k=256$ and the LogBinning lossy transformation, the by-symbol methods clearly outperform the by-value methods in terms of both compression and decompression times. This is explained partly by the reduction of the alphabet size. While this reduction has little meaning to the by-value methods since they do not rely on either symbol frequencies or pattern of symbols, it does help the by-symbol methods in reducing the size of the frequency table and creating more and longer repeated patterns.

6 CONCLUSION

In this work, we have considered how to economically represent quality scores in NGS data as a combination of three main components: lossy transformation, lossless transformation and coding. Of them, the first two are optional, but they can considerably affect the whole process. By separating these components, we were able to see the effects each of them can bring, and also to identify the best settings in order to achieve good system performance.

In our study, we proposed three lossy transformations, introduced or made use of three lossless transformations, investigated several by-value coding schemes and considered more conventional by-symbol methods. Moreover, we demonstrated how data blocking can affect both compression ratios and running times. Finally, we also proposed a method to assess the usability, or worthiness, of any lossy transformation by employing the read mapping tool MAQ.

We found that while full-fledged compression systems such as libbzip2 are widely accepted for economical representation of NGS, they are not the best choice for quality scores. Here, simple codes such as the static code gamma and parameterized codes interp and golomb, when accompanied by the lossless transformation GapTranslating, are highly competitive: they can achieve similar levels of compression while using less time.

In particular, unlike their counterparts that are effective only with large block sizes, the simple codes have the distinguished feature of being unaffected by this factor. This is an important point because it essentially removes the block size parameter from the process, and allows simple codes to greatly outperform their counterparts when small block sizes are in use. Note that small block sizes offer a number of advantages which we have not explored. First, peak memory usage is reduced. Second, since blocks are coded independently, errors in the compressed data stream can be isolated easily. Third, random access in compressed data can be supported at the additional low cost of an index before each block. Finally, independent blocks mean that our findings would apply to higher coverage datasets.

As expected for the lossy transformations, compression ratios positively correlate with the number of distinct quality scores. Of the three proposed transformations, LogBinning is the most effective—it achieves excellent relative mapping performance even when employing only a few distinct quality scores. With this choice, compression ratio is improved by around six times, while processing time, when coupled with by-symbol coding schemes, is reduced significantly—all suggesting that lossy transformations are useful.

Our results for SRR032209 are supported by those of SRR070788_1, which appear in the Supplementary Material. Even though the reads are longer, the *relative* performance of the transformation and compression methods remain the same.

Our view is that as NGS data continues to grow, lossy and lossless transformations can work in tandem. For example, NGS data can be compressed losslessly and kept in off-line storage (such as tape backup) while lossy versions of the data can be shared between users and research laboratories for daily use. Employing lossy transformations requires consideration since such changes are more easily noticeable and difficult to assess compared with images and video data (Witten *et al.*, 1994)—our evaluation with MAQ is meant to address this issue.

In the future, we plan to reorder the reads, as was done by Wan and Asai (2010) for the sequence bases, so that each block possesses reads that have similar quality score patterns. With respect to running time, we intend to parallelize some of our methods across blocks. Furthermore, the effect of lossy transformations on other applications of NGS data, such as RNA-Seq and SNP calling, also needs to be evaluated.

Our implementation, dubbed QScores-Archiver, is available from http://www.cb.k.u-tokyo.ac.jp/asailab/members/rwan under the Lesser General Public License version 3 or later to allow users to combine it with their FASTQ compression systems.

ACKNOWLEDGEMENTS

The authors are grateful to members of the Computational Biology Research Center (AIST) for fruitful discussions and to the reviewers for helpful comments. Implementations of most coding schemes are based on the pseudocode from Moffat and Turpin (2002).

Funding: Grant-in-Aid for Scientific Research on Innovative Areas (221S002) to R.W. and K.A.; National Cancer Center Research and Development Fund (to K.A.); National Institute of Advanced Industrial Science and Technology (AIST) (to R.W. and K.A.); Australian Research Council (to V.N.A.).

Conflict of Interest: none declared.

REFERENCES

Blankenberg,D. *et al.* (2010) Manipulation of FASTQ data with Galaxy. *Bioinformatics*, **26**, 1783–1785.

Cock, P.J.A., Fields, C.J., *et al.* (2010). The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Res.*, **38**, 1767–1771.

Daily,K. *et al.* (2010) Data structures and compression algorithms for high-throughput sequencing technologies. *BMC Bioinformatics*, **11**, 514.

Deorowicz,S. and Grabowski,S. (2011) Compression of genomic sequences in FASTQ format. *Bioinformatics*, **27**, 860–862.

Ewing,B. and Green,P. (1998) Base-calling of automated sequencer traces using Phred. II. error probabilities. *Genome Res.*, **8**, 186–194.

Gallager,R. and van Voorhis,D. (1975) Optimal source codes for geometrically distributed integer alphabets. *IEEE Trans. Informat. Theory*, **21**, 228–230.

Giancarlo,R. *et al.* (2009) Textual data compression in computational biology: a synopsis. *Bioinformatics*, **25**, 1575–1586.

Golomb,S.W. (1966) Run-length encodings. *IEEE Trans. Informat. Theory*, **12**, 399–401.

Hsi-Yang Fritz,M. *et al.* (2011) Efficient storage of high throughput DNA sequencing data using reference-based compression. *Genome Res.*, **21**, 734–740.

Huffman,D.A. (1952) A method for the construction of minimum-redundancy codes. *Proc. Inst. Radio Eng.*, **40**, 1098–1101.

Kozanitis,C. *et al.* (2011) Compressing genomic sequence fragments using SlimGene. *J. Comput. Biol.*, **18**, 401–413.

Leinonen,R. *et al.* (2011) The Sequence Read Archive. *Nucleic Acids Res.*, **39**, D19–D21.

Li,H. *et al.* (2008) Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome Res.*, **18**, 1851–1858.

Moffat,A. and Stuiver,L. (2000) Binary interpolative coding for effective index compression. *Inform. Retr.*, **3**, 25–47.

Moffat,A. and Turpin,A. (2002) *Compression and Coding Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA.

Rice,R.F. (1979) Some practical universal noiseless coding techniques. *Technical Report 79–22*, Jet Propulsion Laboratory, Pasadena, California.

Tembe,W. *et al.* (2010) G-SQZ: compact encoding of genomic sequence and quality data. *Bioinformatics*, **26**, 2192–2194.

Wan,R. and Asai,K. (2010) Sorting next generation sequencing data improves compression effectiveness. In *Proceedings of the 2010 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)—Workshops and Posters*. Hong Kong, China, pp. 567–572.

Witten,I.H. *et al.* (1994). Semantic and generative models for lossy text compression. *Comput. J.*, **37**, 83–87.

Ziv,J. and Lempel,A. (1977) A universal algorithm for sequential data compression. *IEEE Trans. Inform. Theory*, **23**, 337–343.