Bioinformatics, 31(19), 2015, 3122–3129
doi: 10.1093/bioinformatics/btv330
Advance Access Publication Date: 28 May 2015
Original Paper

OXFORD

Sequence analysis

QVZ: lossy compression of quality values

Greg Malysa*, Mikel Hernaez*, Idoia Ochoa*, Milind Rao, Karthik Ganesan and Tsachy Weissman

Department of Electrical Engineering, Stanford University, Stanford, CA 94305, USA

*To whom correspondence should be addressed.

Associate Editor: Inanc Birol

Received on November 19, 2014; revised on April 7, 2015; accepted on April 9, 2015

Abstract

Motivation: Recent advancements in sequencing technology have led to a drastic reduction in the cost of sequencing a genome. This has generated an unprecedented amount of genomic data that must be stored, processed and transmitted. To facilitate this effort, we propose a new lossy compressor for the quality values presented in genomic data files (e.g. FASTQ and SAM files), which comprise roughly half of the storage space (in the uncompressed domain). Lossy compression allows for compression of data beyond its lossless limit.

Results: The proposed algorithm *QVZ* exhibits better rate-distortion performance than the previously proposed algorithms, for several distortion metrics and for the lossless case. Moreover, it allows the user to define any quasi-convex distortion function to be minimized, a feature not supported by the previous algorithms. Finally, we show that QVZ-compressed data exhibit better performance in the genotyping than data compressed with previously proposed algorithms, in the sense that for a similar rate, a genotyping closer to that achieved with the original quality values is obtained.

Availability and implementation: QVZ is written in C and can be downloaded from https://github.com/mikelhernaez/qvz.

Contact: mhernaez@stanford.edu or gmalysa@stanford.edu or iochoa@stanford.edu

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

1 Introduction

There has been a recent explosion of interest in genome sequencing, driven by advancements in the sequencing technology. Although early sequencing technologies required years to capture a 3 billion nucleotide genome (Schatz and Langmead, 2013), genomes as large as 22 billion nucleotides are now being sequenced within days (Zimin *et al.*, 2014) using next-generation sequencing technologies (Metzker, 2010). Further, the cost of sequencing a human-length genome has dropped from billions of dollars to merely $4000 (http://systems.illumina.com/systems/hiseq-x-sequencing-system.ilmn) within the past 15 years (Hayden, 2014). These developments in efficiency and affordability have allowed many to envision whole-genome sequencing as an invaluable tool to be used in both personalized medical care and public health (Berg *et al.*, 2011). In anticipation of the storage challenges that increasingly large and ubiquitous genomic datasets could present, compression of the raw data generated by sequencing machines has become an important topic.

The output data of the sequencing machines is generally stored in the widely accepted FASTQ format (Metzker, 2010). A FASTQ file dedicates four lines to each fragment of a genome (a 'read') analyzed by the sequencing machine. The first line contains a header with some identifying information, the second lists the nucleotides in the read, the third is similar to the first one and the fourth lists a 'quality value' (also referred to as quality score) for each nucleotide. The quality values are generally stored using the *Phred score*, which corresponds to the particular number $Q = -10\log_{10}P$, where P is an estimate (calculated by the base calling software running on the sequencing machine) of the probability that the corresponding nucleotide in the read is in error. These scores are commonly represented in the FASTQ file with an ASCII alphabet [33 : 73] or [64 : 104], where the value corresponds

to $Q+33$ or $Q+64$, respectively. In addition, the information contained in the FASTQ files is also found in the SAM files (Li *et al.*, 2009), which store the information pertaining to the alignment of the reads to a reference.

Quality values, which comprise more than half of the compressed data, have proven to be more difficult to compress than the reads (Bonfield and Mahoney, 2013). Thus, generating better compression tools for quality values is crucial for reducing the storage required for large files. Unlike nucleotide information, the quality values generated by sequencing machines tend to exhibit predictable behavior within each read. Strong correlations exist between adjacent quality values as well as the trend that quality values degrade drastically as a read progresses (Kozanitis *et al.*, 2011). There is also evidence that quality values are corrupted by some amount of noise introduced during sequencing (Bonfield and Mahoney, 2013). These features are well explained by imperfections in the base-calling algorithms, which estimate the probability that the corresponding nucleotide in the read is in error (Das and Vikalo, 2012). Further, applications which operate on reads (referred to as 'downstream applications') often make use of the quality values in a heuristic manner. This is particularly true for sequence alignment algorithms (Langmead *et al.*, 2009; Li and Durbin, 2009) and single-nucleotide polymorphism (SNP) calling (DePristo et al., 2011; Li, 2011), the latter having been shown to be resilient to changes in the quality values (in the sense that, in general, little is compromised in performance when quality values are modified (Ochoa *et al.*, 2013; Yu *et al.*, 2014) (http://www.illumina.com/documents/products/whitepapers/whitepaper_datacompression.pdf).

Based on these observations, lossy (as opposed to lossless) compression of quality values emerges as a natural candidate for significantly reducing storage requirements while maintaining adequate performance of downstream applications. While rate-distortion theory provides a framework to evaluate lossy compression algorithms, the criterion under which the goodness of the reconstruction should be assessed is a crucial question. It makes sense to pick a distortion measure by examining how different distortion measures affect the performance of downstream applications, but the abundance of applications and variations in how quality values are used makes this choice too dependent on the specifics of the applications considered.

These trade-offs suggest that an ideal lossy compressor for quality values should not only provide the best possible compression and accommodate downstream applications, but it should provide flexibility to allow a user to pick a desired distortion measure and/or rate.

In this work, we present such a scheme which we call *QVZ* ('Quality Values Zip'), which achieves significantly better rate-distortion performance than any of the existing algorithms. Specifically, the proposed algorithm obtains up to four times better compression than previously proposed algorithms for the same average distortion. In addition, *QVZ* achieves lossless compression. Moreover, we analyze the effect of *QVZ* on the genotyping and show that better results are obtained than with the previously proposed algorithms. Finally, we present some preliminary results that suggest that lossy compression could potentially improve the genotyping with respect to the uncompressed data. This may be due to the inherently noisy nature of the quality values, in ways that will be thoroughly investigated in future work.

1.1 Survey of lossy compressors for quality values

Lossy compression for quality values has recently started to be explored. *Slimgene* (Kozanitis *et al.*, 2011) fits fixed-order Markov encodings for the differences between adjacent quality values and compresses the prediction using a Huffman code (ignoring whether or not there are prediction errors). *Q-Scores Archiver* (Wan *et al.*, 2012) quantizes quality values via several steps of transformations and then compresses the lossy data using an entropy encoder.

Fastqz (Bonfield and Mahoney, 2013) uses a fixed-length code, which represents quality values above 30 using a specific byte pattern and quantizes all lower quality values to 2. *Scalce* (Hach *et al.*, 2012) first calculates the frequencies of different quality values in a subset of the reads of a FASTQ file. Then the quality values which achieve local maxima in frequency are determined. Anytime these local maximum values appear in the FASTQ file, the neighboring values are shifted to within a small offset of the local maximum, thereby reducing the variance in quality values. The result is compressed using an arithmetic encoder.

QualComp (Ochoa *et al.*, 2013) applied rate-distortion theory as a framework for designing a lossy compression algorithm when mean-squared error (MSE) is the distortion measure. Quality value data are first clustered using a k-means algorithm and then an optimization problem is solved to minimize MSE of the compressed output with respect to a rate constraint. *BEETL* (Janin *et al.*, 2013) first applies the Burrows–Wheeler Transform to reads and uses the same transformation on the quality values. Then, the nucleotide suffixes generated by the Burrows–Wheeler Transform are scanned. Groups of suffixes which start with the same k bases while also sharing a prefix of at least k bases are found. All the quality values for the group are converted to a mean quality value, taken within the group or across all the groups. *RQS* (Yu *et al.*, 2014) first generates off-line a dictionary of commonly occurring k-mers throughout a population-sized read dataset of the species under consideration. It then computes the divergence of the k-mers within each read to the dictionary and uses that information to decide whether to preserve or discard the corresponding quality values. *PBlock* (Cánovas *et al.*, 2014) allows the user to determine a threshold for the maximum per-symbol distortion. The first quality value in the file is chosen as the first 'representative'. Quality values are then quantized symbol-by-symbol to the representative if the resulting distortion would fall within the threshold. If the threshold is exceeded, the new quality value takes the place of the representative and the process continues. The algorithm keeps track of the representatives and run-lengths, which are compressed losslessly at the end. *RBlock* (Cánovas *et al.*, 2014) uses the same process, but the threshold instead sets the maximum allowable ratio of any quality value to its representative as well as the maximum value of the reciprocal of this ratio. (Cánovas *et al.*, 2014) also compared the performance of existing lossy compression schemes for different distortion measures.

Finally, Illumina proposed a new binning scheme for reducing the size of the quality values. This binning scheme has been implemented in the state-of-the-art compression tools *CRAM* (Fritz *et al.*, 2011) and *DSRC2* (Roguski and Deorowicz, 2014).

To our knowledge, and based on the results of Cánovas *et al.* (2014), *RBlock, PBlock* and *QualComp* provide the best rate-distortion performance among existing lossy compression algorithms for quality values that do not use any extra information. For this reason, in Section 3 we use *RBlock, PBlock* and *QualComp* as a representation of the existing state-of-the-art when comparing with *QVZ*, together with *CRAM* and *DSRC2*, which apply Illumina's binning scheme. For completeness, we also compare the lossless performance of *QVZ* with that of *CRAM, DSRC2* (in their lossless mode) and *gzip*.

2 Methods

As described previously, we seek to compress the quality scores presented in the genomic data. Let N be the number of quality score sequences to be compressed. The proposed algorithm assumes that all the quality score sequences are of the same length L (for trimmed or hard-clipped reads, please refer to the Supplementary Data). Each sequence consists of ASCII characters representing the scores, belonging to an alphabet $\mathcal{X}$, for example $\mathcal{X} = [33 : 73]$. These quality score sequences are extracted from the genomic file (e.g. FASTQ and SAM files) prior to compression.

We model the quality score sequence $\mathbf{X} = [X_1, X_2, \ldots, X_L]$ by a Markov chain of order one: we assume the probability that X_i takes a particular value depends on previous values only through the value of X_{i-1}. We further assume that the quality score sequences are independent and identically distributed (i.i.d.). We use a Markov model based on the observation that quality scores are highly correlated with their neighbors within a single sequence, and we refrain from using a higher order Markov model to avoid the increased overhead and complexity this would produce within our algorithm.

The Markov model is defined by its transition probabilities $P(X_i|X_{i-1})$, for $i \in 1, 2, \ldots, L$, where $P(X_1|X_0) = P(X_1)$. QVZ finds these probabilities empirically from the entire dataset to be compressed and uses them to design a codebook. The codebook is a set of quantizers indexed by position and previously quantized value (the context). These quantizers are constructed using a variant of the Lloyd–Max algorithm (Lloyd, 1982). After quantization, a lossless, adaptive arithmetic encoder is applied to achieve entropy-rate compression.

In summary, the steps taken by QVZ are as follows:

1. Compute the empirical transition probabilities of a Markov-1 Model from the data.
2. Construct a codebook (Section 2.2) using the Lloyd–Max algorithm (Section 2.1).
3. Quantize the input using the codebook and run the arithmetic encoder over the result (Section 2.3).

2.1 Lloyd–Max quantizer

Given a random variable X governed by the probability mass function $P(\cdot)$ over the alphabet $\mathcal{X}$ of size K, let $D \in R^{K\times K}$ be a distortion matrix where each entry $D_{x,y} = d(x, y)$ is the penalty for reconstructing symbol x as y. We further define $\mathcal{Y}$ to be the alphabet of the quantized values of size $M \leq K$.

Thus, a Lloyd–Max quantizer, denoted hereafter as $\mathrm{LM}(\cdot)$, is a mapping $\mathcal{X} \to \mathcal{Y}$ that minimizes an expected distortion. Specifically, the Lloyd–Max quantizer seeks to find a collection of boundary points $b_k \in \mathcal{X}$ and reconstruction points $y_k \in \mathcal{Y}$, where $k \in \{1, 2, \ldots, M\}$, such that the quantized value of symbol $x \in \mathcal{X}$ is given by the reconstruction point of the region to which it belongs (Fig. 1). For region k, any $x \in \{b_{k-1}, \ldots, b_k - 1\}$ is mapped to y_k, with b_0 being the lowest score in the quality alphabet and b_M the highest score plus one. Thus, the Lloyd–Max quantizer aims to minimize the expected distortion by solving

$$\{b_k, y_k\}_{k=1}^{M} = \underset{b_k, y_k}{\operatorname{argmin}} \sum_{j=1}^{M} \sum_{x=b_{j-1}}^{b_j - 1} P(x) d(x, y_j). \tag{1}$$

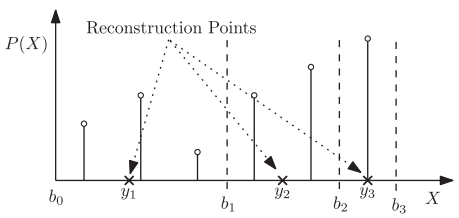

Fig. 1. Example of the boundary points and reconstruction points found by a Lloyd–Max quantizer, for $M = 3$

To approximately solve Equation (1), which is an integer programming problem, we employ an algorithm which is initialized with uniformly spaced boundary values and reconstruction points taken at the midpoint of these bins. For an arbitrary D and $P(\cdot)$, this problem requires an exhaustive search. We assume that the distortion measure $d(x, y)$ is quasi-convex over y with a minimum at $y = x$, i.e. when $x \leq y_1 \leq y_2$ or $y_2 \leq y_1 \leq x$, $d(x, y_1) \leq d(x, y_2)$. If the distortion measure is quasi-convex, an exchange argument suffices to show the optimality of contiguous quantization bins and a reconstruction point within the bin. The following steps are iterated until convergence:

1. Solving for y_k: We first minimize Equation (1) partially over the reconstruction points given boundary values. The reconstruction points are obtained as,
$$y_k = \underset{y=\{b_{k-1},\ldots,b_k-1\}}{\operatorname{argmin}} \sum_{x=b_{k-1}}^{b_k - 1} P(x) d(x, y), \; \forall k = 1, 2, \ldots, M. \tag{2}$$
2. Solving for b_k: This step minimizes Equation (1) partially over the boundary values given the reconstruction points. b_k could range from $\{y_k + 1, \ldots, y_{k+1}\}$ and is chosen as the largest point where the distortion measure to the previous reconstruction value y_k is lesser than the distortion measure to the next reconstruction value y_{k+1}, i.e.
$$b_k = \max \{x \in \{y_k + 1, \ldots, y_{k+1}\} : P(x) d(x, y_k) \leq P(x) d(x, y_{k+1})\} \; \forall k = 1, 2, \ldots, M - 1. \tag{3}$$

Note that this algorithm, which is a variant of the Lloyd–Max quantizer, converges in at most K steps.

Given a distortion matrix D, the defined Lloyd–Max quantizer depends on the number of regions M and the input probability mass function $P(\cdot)$. Therefore, we denote the Lloyd–Max quantizer with M regions as $\mathrm{LM}_M^P(\cdot)$ and the quantized value of a symbol $x \in \mathcal{X}$ as $\mathrm{LM}_M^P(x)$.

An ideal lossless compressor applied to the quantized values can achieve a rate equal to the entropy of $\mathrm{LM}_M^P(X)$, which we denote by $\mathrm{H}(\mathrm{LM}_M^P(X))$. For a fixed probability mass function $P(\cdot)$, the only varying parameter is the number of regions M. Since M needs to be an integer, not all rates are achievable. Because we are interested in achieving an arbitrary rate R, we define an extended version of the LM quantizer, denoted as LME. The extended quantizer consists of two LM quantizers with the numbers of regions given by ρ and $\rho + 1$, each of them used with probability $1 - r$ and r, respectively (where $0 \leq r \leq 1$). Specifically, ρ is given by the maximum number of regions such that $\mathrm{H}(\mathrm{LM}_\rho^P(X)) < R$ (which implies $\mathrm{H}(\mathrm{LM}_{\rho+1}^P(X)) > R$). Then, the probability r is chosen such that the average entropy (and hence the rate) is equal to R, the desired rate. More formally,

$$\mathrm{LME}_R^P(x) = \begin{cases} \mathrm{LM}_\rho^P(x), & \text{w.p. } 1 - r, \\ \mathrm{LM}_{\rho+1}^P(x), & \text{w.p. } r, \end{cases}$$
$$\rho = \max \{x \in \{1, \ldots, K\} : \mathrm{H}(\mathrm{LM}_x^P(X)) \leq R\} \tag{4}$$
$$r = \frac{R - \mathrm{H}(\mathrm{LM}_\rho^P(X))}{\mathrm{H}(\mathrm{LM}_{\rho+1}^P(X)) - \mathrm{H}(\mathrm{LM}_\rho^P(X))}.$$

2.2 Codebook generation

Because we assume the data follows a Markov-1 model, for a given position $i \in \{1, \ldots, L\}$ we design as many quantizers $\mathcal{Q}_q^i$ as there were unique possible quantized values q in the previous context $i-1$. This collection of quantizers forms the codebook for QVZ. For an unquantized quality score X_i, we denote the quantized version as Q_i, so $\mathbf{Q} = [Q_1, Q_2, \ldots, Q_L]$ is the random vector representing a quantized sequence. The quantizers are defined as

$$\mathcal{Q}^1 = \mathrm{LME}_{\alpha \mathrm{H}(X_1)}^{\mathrm{P}(X_1)} \tag{5}$$

$$\mathcal{Q}_q^i = \mathrm{LME}_{\alpha \mathrm{H}(X_i|Q_{i-1}=q)}^{\mathrm{P}(X_i|Q_{i-1}=q)}, \text{ for } i = 2, \ldots, L \tag{6}$$

where $\alpha \in [0, 1]$ is the desired compression factor. $\alpha = 0$ corresponds to 0 rate encoding, $\alpha = 1$ to lossless compression and any value in between scales the input file size by that amount. Note that the entropies can be directly computed from the corresponding empirical probabilities.

Next we show how the probabilities needed for the LMEs are computed.

2.2.1 Computation of the probability P

To compute the quantizers defined above, we require $\mathrm{P}(X_{i+1}|Q_i)$, which must be computed from the empirical statistics $\mathrm{P}(X_{i+1}|X_i)$ found earlier. The first step is to calculate $\mathrm{P}(Q_i|X_i)$ recursively and then to apply Bayes rule and the Markov Chain property to find the desired probability:

$$\begin{aligned}
\mathrm{P}(Q_i|X_i) &= \sum_{Q_{i-1}} \mathrm{P}(Q_i, Q_{i-1}|X_i) \\
&= \sum_{Q_{i-1}} \mathrm{P}(Q_i|X_i, Q_{i-1}) \sum_{X_{i-1}} \mathrm{P}(Q_{i-1}, X_{i-1}|X_i) \\
&= \sum_{Q_{i-1}} \mathrm{P}(Q_i|X_i, Q_{i-1}) \sum_{X_{i-1}} \mathrm{P}(Q_{i-1}|X_{i-1}, X_i) \mathrm{P}(X_{i-1}|X_i) \\
&= \sum_{Q_{i-1}} \mathrm{P}(Q_i|X_i, Q_{i-1}) \sum_{X_{i-1}} \mathrm{P}(Q_{i-1}|X_{i-1}) \mathrm{P}(X_{i-1}|X_i)
\end{aligned} \tag{7}$$

Equation (7) follows from the fact that $Q_{i-1} \leftrightarrow X_{i-1} \leftrightarrow X_i$ form a Markov chain. Additionally, $\mathrm{P}(Q_i|X_i, Q_{i-1} = q) = \mathrm{P}(\mathcal{Q}_q^i(X_i) = Q_i)$, which is the probability that a specific quantizer produces Q_i given previous context q. This can be found directly from r [defined in Eq. (4)] and the possible values for q. We now proceed to compute the required conditional probability as

$$\begin{aligned}
\mathrm{P}(X_{i+1}|Q_i) &= \sum_{X_i} \mathrm{P}(X_i|Q_i) \mathrm{P}(X_{i+1}|X_i, Q_i) \\
&= \sum_{X_i} \mathrm{P}(X_i|Q_i) \mathrm{P}(X_{i+1}|X_i)
\end{aligned} \tag{8}$$

$$= \frac{1}{\mathrm{P}(Q_i)} \sum_{X_i} \mathrm{P}(Q_i|X_i) \mathrm{P}(X_i, X_{i+1}), \tag{9}$$

where Equation (8) follows from the same Markov chain as earlier. Terms in Equation (9) are: (i) $\mathrm{P}(X_i, X_{i+1})$: joint pmf computed empirically from the data, (ii) $\mathrm{P}(Q_i|X_i)$: computed in Equation (7) and (iii) $\mathrm{P}(Q_i)$: normalizing constant given by

$$\mathrm{P}(Q_i = q) = \sum_{X_i} \mathrm{P}(Q_i = q|X_i) \mathrm{P}(X_i).$$

The steps necessary to compute the codebook are summarized in Algorithm 1. Note that support(X) denotes the support of the random variable X or the set of values that X takes with non-zero probability.

Algorithm 1 Generate codebook

Input: Transition probabilities $\mathrm{P}(X_i|X_{i-1})$, compression factor α
Output: Codebook: collection of quantizers $\{\mathcal{Q}_q^l\}$
$P \leftarrow \mathrm{P}(X_1)$
Compute and store $\mathcal{Q}^1$ based on P using Equation (5)
for all columns $i = 2$ to L **do**
  Compute $\mathrm{P}(Q_{i-1}|X_{i-1} = x)\, \forall x \in \text{support}(X_{i-1})$
  Compute $\mathrm{P}(X_i|Q_{i-1})\, \forall q \in \text{support}(Q_{i-1})$
  for all $q \in \text{support}(Q_{i-1})$ **do**
    $P \leftarrow \mathrm{P}(X_i|Q_{i-1} = q)$
    Compute and store $\mathcal{Q}_q^i$ based on P using Equation (6)
  end for
end for

2.3 Encoding

The encoding process is summarized in Algorithm 2. First, we generate the codebook and quantizers. For each read, we quantize all scores sequentially, with each value forming the left context for the next value. As they are quantized, scores are passed to an adaptive arithmetic encoder, which uses a separate model for each position and context. For a detailed explanation of the arithmetic encoder, we refer the reader to the Supplementary Data.

Algorithm 2. Encoding of quality scores

Input: Set of N reads $\{\mathbf{X}_j\}_{j=1}^N$
Output: Set of quantizers $\{\mathcal{Q}_q^l\}$ (codebook) and compressed representation of reads
Compute empirical statistics of input reads
Compute codebook $\{\mathcal{Q}_q^l\}$ according to Algorithm 1
for all $j = 1$ to N **do**
  $[X_1, \ldots, X_L] \leftarrow \mathbf{X}_j$
  $Q_1 \leftarrow \mathcal{Q}^1(X_1)$
  for all $i = 2$ to L **do**
    $Q_i \leftarrow \mathcal{Q}_{Q_{i-1}}^i(X_i)$
  end for
  Pass $[Q_1, \ldots, Q_L]$ to arithmetic encoder
end for

2.4 Clustering

The performance of the compression algorithm depends on the conditional entropy of each quality score given its predecessor. Earlier we assumed that the data were all i.i.d., but it is more effective to allow each read to be independently selected from one of several distributions. If we first cluster the reads into C clusters, then the variability within each cluster may be smaller. In turn, the conditional entropy would decrease and fewer bits would be required to encode X_i at a given distortion level, assuming that an individual codebook is available unique to each cluster.

Thus, QVZ has the option of clustering the data prior to compression. Specifically, it uses the K-means algorithm (*MacQueen et al.*, 1967), initialized using C quality value sequences chosen at random from the data. It assigns each sequence to a cluster by means of Euclidean distance. Then, the centroid of each cluster is computed as the mean vector of the sequences assigned to it. Because of the lack of convergence guarantees, we have incorporated a stop criterion that avoids further iterations once the centroids of the clusters have moved

less than U units (in Euclidean distance). The parameter U is set to 4 by default, but it can be modified by the user. Finally, storing which cluster each read belongs to incurs a rate penalty of at most $\log_2(C)/L$ bits per symbol, which allows *QVZ* to reconstruct the series of reads in the same order as they were in the uncompressed input file.

3 Results and discussion

To assess the performance of the proposed algorithm *QVZ*, we compare it with the state of the art lossy compression algorithms *PBlock, RBlock* (Cánovas *et al.*, 2014) and *QualComp* (Ochoa *et al.*, 2013). We also consider *CRAM* (Fritz *et al.*, 2011), *DSRC2* (Roguski and Deorowicz, 2014) and *gzip*. In this assessment, we focus on two aspects that we believe are important: the rate-distortion curve and the behavior in genotyping. The rate-distortion curve provides a framework for comparison that is independent of the downstream applications, which vary significantly in their use of quality scores. It also gives a measure of fidelity for each of the algorithms: how similar are the reconstructed quality scores to the original values? On the other hand, examining the behavior in genotyping aims to provide a comparison on how the different lossy compressors affect the downstream applications, which are widely used in practice. Specifically, we focus on SNP calling, because analyzing the effects of lossy compression on this application is of significant importance in practice.

The dataset used for our analysis is the *NA12878.HiSeq.WGS.bwa.cleaned.recal.hg19.20.bam*, which corresponds to the chromosome 20 of a *Homo sapiens* individual. We downloaded it from the GATK bundle (http://tiny.cc/3i49tx). This dataset pertains to one of the most studied human individuals in the literature (DePristo et al., 2011; Zook et al., 2014), making it a suitable baseline for comparison. We generated the SAM file from the BAM file and then extracted the quality score sequences from it. The dataset contains 51, 585, 658 sequences, each of length 101. We consider four more datasets for our study, namely, the chromosome 20 of the *H.sapiens* dataset *SRR622461*, the whole genome of a *Saccharomyces cerevisiae* (*SRR1179906*) and two ChIP-Seq datasets from a *Mus musculus* (*SRR32209*) and a *Drosophila melanogaster* (*ERR011354*). Because of space constraints, their analyses are presented in the Supplementary Data.

The machine used to perform the experiments has the following specifications: 39 GB RAM, Intel Core i7-930 CPU at 2.80 GHz x 8 and Ubuntu 12.04 LTS.

The next two subsections report on the results of our study as they pertain to rate-distortion and genotyping, respectively.

3.1 Rate-Distortion analysis

First, we describe the options used to run each algorithm. *QVZ* was run with the default parameters, multiple rates and different number of clusters. *PBlock* and *RBlock* (http://tiny.cc/kg49tx) were run with different values of p and r, respectively, and with $m = 1$ (the default value). *QualComp* (http://tiny.cc/9b49tx) was run with three clusters and multiple rates, and *CRAM* and *DSRC2* were run with the lossy mode that implements Illumina's proposed binning scheme. Finally, we also run each of the mentioned algorithms in the lossless mode, except *QualComp*, since it does not support lossless compression. We refer the reader to the Supplementary Data for more details.

QVZ can minimize any quasi-convex distortion, if the corresponding matrix is provided, or any of the following three built-in distortion metrics: (i) the average MSE, where $d(x,y) = |x-y|^2$; (ii) the average L1 distortion, where $d(x,y) = |x-y|$ and (iii) the average Lorentzian distortion, where $d(x,y) = \log_2(1+|x-y|)$. Hereafter, we refer to each of them as *QVZ-M*, *QVZ-A* and *QVZ-L*, respectively. *QVZ* can also perform clustering prior to compression—similar to *QualComp*– using a user-specified number of clusters, so we ran it with 1, 3 and 5 clusters for each distortion metric to examine the effects of clustering on the rate-distortion curve.

Assuming N reads of length L each, the distortion D used to compare the different algorithms is computed as

$$D = \frac{1}{NL}\sum_{k=1}^{N}\sum_{i=1}^{L} d(x_i(k), y_i(k)), \tag{10}$$

where $x_i(k)$ denotes the quality score value of read k at position i, $y_i(k)$ the corresponding reconstructed value (after lossy compression) and $d(\cdot,\cdot)$ the distortion metric under consideration. Since *QVZ* can select to optimize for MSE, L1 or Lorentzian distortions, we provide results for all three. Any other distortion metric can be used for comparison, but we limit our attention to these three due to space constraints and refer the reader to the Supplementary Data for results on other distortion metrics. As a measure of rate, we use the final size of the quality score sequences after compression. The results are presented in Figure 2.

As can be seen in Figure 2, *QVZ* outperforms the previously proposed algorithms for all three choices of distortion metric. Furthermore, although *QualComp* reconstructs the quality score sequences in a different order, *QVZ* maintains the original order. This is achieved by storing the cluster to which each quality score

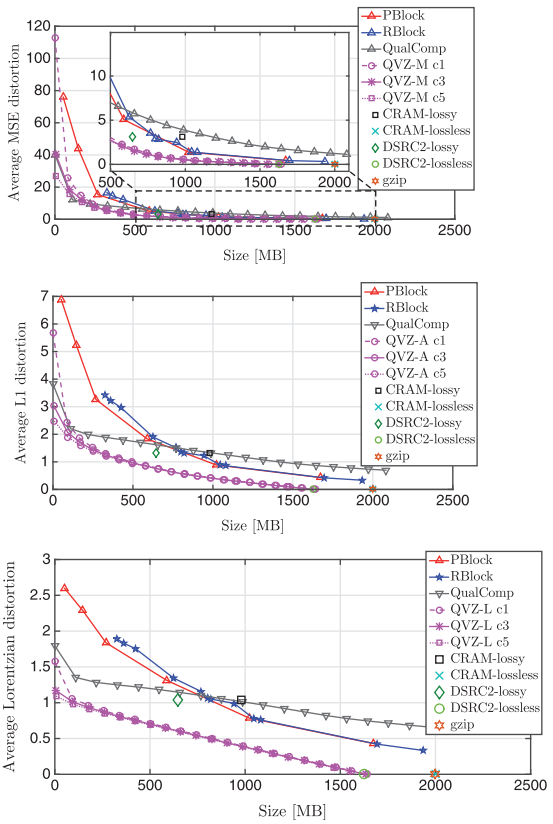

Fig. 2. Rate-distortion curves of *PBlock, RBlock, QualComp* and *QVZ*, for MSE, L1 and Lorentzian distortions. In *QVZ*, c1, c3 and c5 denote 1, 3 and 5 clusters, respectively

Table 1. Lossless results of the different algorithms for the *NA12878* dataset

	QVZ (3 clusters)	*PBlock RBlock*	*DSRC2*	*CRAM*	*gzip*
Size (MB)	1632	3229	1625	2000	1999

sequence belongs, in contrast to *QualComp* which produces one file per cluster. Note that storing this information for C clusters would incur a cost of approximately $N\log_2 C$ bits, assuming uniform distribution of sequences across the clusters, which is not included for *QualComp* in Figure 2.

The lossy modes of *CRAM* and *DSRC2* can each achieve only one rate-distortion point, and both are outperformed by *QVZ*. We further observe that although *QualComp* outperforms *RBlock* and *PBlock* for low rates (in all three distortions), the latter two achieve a smaller distortion for higher rates. *QVZ*, however, outperforms all previously proposed algorithms in both low and high rates. *QVZ*'s advantage becomes especially apparent for distortions other than MSE.

It is also significant that *QVZ* achieves a zero distortion at a rate at which the other lossy algorithms exhibit positive distortion. In other words, *QVZ* achieves lossless compression faster than *QualComp*, *RBlock* or *PBlock*. In fact, due to its design, *QualComp* cannot achieve lossless compression, even for very high rates. Moreover, *QVZ* also outperforms the lossless compressors *CRAM* and *gzip* and achieves similar performance to that of *DSRC2* (Table 1).

Finally, we observe that applying clustering prior to compression in *QVZ* is especially beneficial at low rates. For higher rates, the performance of 1, 3 and 5 clusters is almost identical. Therefore we recommend using multiple clusters at low rates for better distortion and 1 cluster at high rates for faster compression.

The results obtained from this analysis are in line with the ones presented in the Supplementary Data for the other studied datasets.

QVZ compares favorably with the other schemes insofar as running times are concerned. For example, it requires approximately 13 min to compress the analyzed dataset with one cluster and 12 min to decompress it. If three clusters are used instead, the compression time increases to 18 min. *QualComp*, on the other hand, takes more than 1 h to cluster the data (if more than one cluster is used): around 90 min to compute the necessary statistics and 20 min to finally compress the quality scores. The decompression is done in 15 min. *DSRC2* requires 20 min to compress and decompress, whereas *CRAM* employs 14 min to compress and 4 min to decompress. Finally, both *Pblock* and *Rblock* take around 4 min to compress and decompress, being the algorithms with the least running times among those that we analyzed. The running times of *gzip* to compress and decompress are 7 and 30 min, respectively.

In terms of memory usage, *QVZ* uses 5.7 GB to compress the analyzed dataset and less than 1 MB to decompress, whereas *QualComp* employs less than 1 MB for both operations. *Pblock* and *Rblock* have more memory usage than *QualComp*, but this is still below 40 MB to compress and decompress. *DSRC2* uses 3 GB to compress and 5 GB to decompress, whereas *CRAM* employs 2 GB to compress and 3 GB to decompress. Finally, *gzip* uses less than 1 MB for both operations.

3.2 Genotyping analysis

To perform the genotyping analysis, and following a similar analysis to the one presented in Cánovas *et al.* (2014), we compare the SNP calling of the original SAM file with that obtained when the quality values are replaced with the reconstructed quality values. Note that we replace the quality scores directly in the SAM file: we do not regenerate the SAM file by running an alignment program. The reason is that similar to SNP calling, the alignment program uses quality values (in general) to generate the alignment, and thus by running both it will become impossible to separate the effect that quality scores have on SNP calling from alignment. Note that if the alignment program does not use the quality values [e.g. BWA (Li and Durbin, 2009)], modifying them in the original SAM file is equivalent to re-running the alignment program.

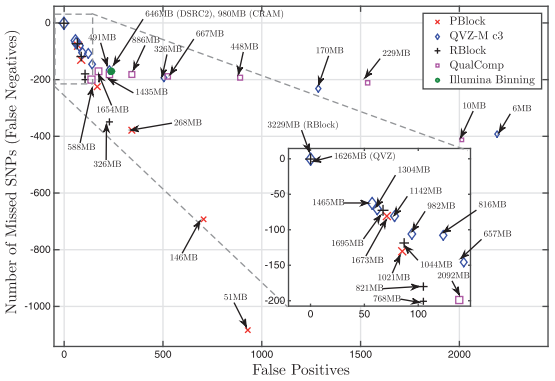

Fig. 3. SNP calling results of the original SAM file (*NA12878*), denoted as uncompressed, and those generated with the different lossy compression algorithms. Note that the *y*-axis refers to the FNs times minus one

We use the programs provided by the HTS library (http://www.htslib.org) to perform SNP calling, with the parameters and commands suggested by the SNP calling workflow therein (exact commands can be found in the Supplementary Data). Any SNP calling program could have been used for this purpose.

Figure 3 shows the number of false negatives (FN) versus the number of false positives (FP) with respect to the uncompressed version. The point (0, 0) corresponds to lossless compression. We chose to show the performance of *QVZ-M* with three clusters for the sake of clarity, although similar performance was obtained for the other configurations of *QVZ* (see the Supplementary Data). As shown previously in the rate-distortion analysis, *QVZ* achieves lossless compression and thus same genotyping as the uncompressed version, with a file size of only 1626 MB, while *RBlock* needs 3229 MB. On the other hand, *QualComp* behaves similarly to *QVZ*, although its files are generally larger for the same genotyping results. Moreover, *QualComp* cannot achieve the same genotyping as the uncompressed version as it cannot generate a lossless file. When comparing with Illumina's binning, we observe that *QVZ* achieves a similar point in the genotyping with 491 MB, whereas *DSRC2* and *CRAM* need 646 MB and 980 MB, respectively.

The differences in convergence to the lossless genotyping between *QVZ* (and *QualComp*) and both *PBlock* and *RBlock* for this dataset are very interesting. While the variant calling of *PBlock*- and *RBlock*-reconstructed data does not generate many FP, it misses several SNPs (i.e., it generates more FN) before achieving perfect genotyping. On the other hand, using *QVZ*- and *QualComp*-reconstructed data seems to result in more calls with higher compression ratios. This behavior makes the number of FP increase as the file size decreases, while the number of true positives remains almost constant for different sizes. Even with a high compression ratio (small size), the observed number of true positives is nearly identical to the uncompressed version. Similar results are observed in the extra analyses provided in the Supplementary Data.

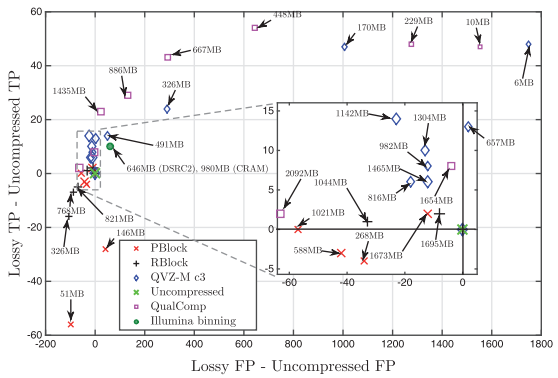

Fig. 4. SNP calling results of the original SAM file (*NA12878*), denoted as uncompressed, and those generated with the different lossy compression algorithms

This observation provided the motivation for our next experiment. Specifically, we wanted to explore whether among the FP called with *QVZ* (especially for low rates), there were actually true positives that were missed with the original SAM file. This situation is conceivable, as the quality scores are inherently noisy, so their lossy compression may serve to denoise them as well, thereby boosting the inferential power of the downstream applications. To verify if this was the case, we compared the SNP calling generated with the modified SAM files with what we refer to as the 'ground truth'. In particular, the 'ground truth' corresponds to the SNPs called over the same individual after following the Best Practices workflow for SNP calling provided by the Broad Institute. The corresponding VCF file containing the SNPs can be found in the Broad Institute Resource Bundle. Note that the SAM file used for this purpose has been previously pre-processed according to the Best Practices provided by the Broad Institute, thus removing most of the FP introduced by duplicates and bad alignments around indels.

Figure 4 shows the difference between the number of TPs and FPs called with the uncompressed version and the different lossy versions with respect to the 'ground truth'. A similar convergence to the lossless case can be seen, just as before when comparing to the unmodified SAM file. In the case of *PBlock* and *RBlock*, no new TPs are found. This seems to be a consequence of the fact that fewer SNPs are called than with the uncompressed version. Moreover, fewer FP are also called than with the uncompressed version, as shown in the lower-left quadrant of the figure. We also observe that with *QVZ*, fewer FPs and more TPs are obtained than those obtained with the Illumina's binning, while achieving more compression.

The upper-left quadrant deserves special attention. It contains those cases where not only are more true positives achieved, but there are also fewer FN. This means that in these cases the genotyping improves over the uncompressed version. It is intriguing to observe that all the files above 700 MB generated with our proposed algorithm *QVZ* are in this quadrant. A similar behavior is also observed when the 'ground truth' is chosen as the one provided by the NIST Proposed Standard, as shown in the Supplementary Data. This is a very interesting finding, as it seems to suggest that the proposed lossy compressor can potentially be used not only as a means to reduce the storage requirements but also for improving the downstream analysis performed on the data. These preliminary findings are admittedly anecdotal. However, they provide a glance of the potential of applying lossy compression for genotype improvement. Further analysis in this direction is left for future research.

4 Conclusion

In this work, we have presented *QVZ*, a new lossy compression algorithm for quality scores in genomic data. The proposed algorithm can work for several distortion metrics, including any quasi-convex distortion metric provided by the user, a feature not supported by the previously proposed algorithms. Moreover, it exhibits better rate-distortion performance. Unlike some of the previously proposed algorithms, *QVZ* also allows for lossless compression and a seamless transition from lossy to the lossless with increasing rate. Moreover, we have shown that in comparison to previously proposed lossy algorithms, using *QVZ*-compressed data achieves genotyping performance closer to that obtained with uncompressed quality values, for similar compression rates.

Finally, we have obtained some preliminary and promising results which suggest that lossy compression could be beneficial not only for storage and transmission but also for boosting performance in downstream applications. The extent of this phenomenon, the relation between the distortion criterion, the compression rate, the characteristics of the noise in the quality values and the resulting performance boosts are due further investigation.

Acknowledgements

The authors would like to thank Golan Yona for providing initial motivation for this work and the anonymous reviewers for helpful suggestions.

Funding

This work was partially supported by a Stanford Graduate Fellowships Program in Science and Engineering, a fellowship from the Basque Government, a grant from the Center for Science of Information (CSoI) and the 1157849-1-QAZCC NSF grant.

Conflict of Interest: none declared.

References

Berg,J. *et al.* (2011) Deploying whole genome sequencing in clinical practice and public health: meeting the challenge one bin at a time. *Genet. Med.*, **13**, 499–504.

Bonfield,J.K. and Mahoney,M.V. (2013) Compression of FASTQ and SAM format sequencing data. *PloS One*, **8**, e59190.

Cánovas,R. *et al.* (2014) Lossy compression of quality scores in genomic data. *Bioinformatics*, **30**, 2130–2136.

Das,S. and Vikalo,H. (2012) Onlinecall: fast online parameter estimation and base calling for illumina's next-generation sequencing. *Bioinformatics*, **28**, 1677–1683.

DePristo,M.A. *et al.* (2011) A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nat. Genet.*, **43**, 491–498.

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

Hach,F. *et al.* (2012) Scalce: boosting sequence compression algorithms using locally consistent encoding. *Bioinformatics*, **28**, 3051–3057.

Hayden,E.C. (2014) Technology: the $1 000 genome. *Nature*, **507**, 294–295.

Janin,L. *et al.* (2013) Adaptive reference-free compression of sequence quality scores. *Bioinformatics*, **30**, 24–30.

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

Langmead,B. *et al.* (2009) Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol.*, **10**, R25.

Li,H. (2011) A statistical framework for SNP calling, mutation discovery, association mapping and population genetical parameter estimation from sequencing data. *Bioinformatics*, **27**, 2987–2993.

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

Li,H. *et al.* (2009) The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, **25**, 2078–2079.

Lloyd,S. (1982) Least squares quantization in PCM. *IEEE Trans. Inf. Theory*, **28**, 129–137.

MacQueen,J. *et al.* (1967) Some methods for classification and analysis of multivariate observations. In: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, **Vol. 1**, Oakland, CA, USA, pp. 281–297.

Metzker,M.L. (2010) Sequencing technologies the next generation. *Nat. Rev. Genet.*, **11**, 31–46.

Ochoa,I. *et al.* (2013) Qualcomp: a new lossy compressor for quality scores based on rate distortion theory. *BMC Bioinformatics*, **14**, 187.

Roguski,Ł. and Deorowicz,S. (2014) DSRC 2—industry-oriented compression of FASTQ files. *Bioinformatics*, **30**, 2213–2215.

Schatz,M.C. and Langmead,B. (2013) The DNA data deluge. *IEEE Spectr.*, **50**, 28–33.

Wan,R. *et al.* (2012) Transformations for the compression of FASTQ quality scores of next-generation sequencing data. *Bioinformatics*, **28**, 628–635.

Yu,Y.W. *et al.* (2014) Traversing the k-mer landscape of NGS read datasets for quality score sparsification. In: *Research in Computational Molecular Biology*. Springer, pp. 385–399.

Zimin,A. *et al.* (2014) Sequencing and assembly of the 22-gb loblolly pine genome. *Genetics*, **196**, 875–890.

Zook,J.M. *et al.* (2014) Integrating human sequence data sets provides a resource of benchmark SNP and indel genotype calls. *Nat. Biotechnol.*, **32**, 246–251.