Published online 12 October 2012 Nucleic Acids Research, 2013, Vol. 41, No. 1 e27
doi:10.1093/nar/gks939

NGC: lossless and lossy compression of aligned high-throughput sequencing data

Niko Popitsch* and Arndt von Haeseler

Center for Integrative Bioinformatics Vienna, Max F. Perutz Laboratories, University of Vienna, Medical University of Vienna, Dr. Bohr Gasse 9, Vienna, A-1030, Austria

Received July 10, 2012; Revised September 14, 2012; Accepted September 15, 2012

ABSTRACT

A major challenge of current high-throughput sequencing experiments is not only the generation of the sequencing data itself but also their processing, storage and transmission. The enormous size of these data motivates the development of data compression algorithms usable for the implementation of the various storage policies that are applied to the produced intermediate and final result files. In this article, we present NGC, a tool for the compression of mapped short read data stored in the wide-spread SAM format. NGC enables lossless and lossy compression and introduces the following two novel ideas: first, we present a way to reduce the number of required code words by exploiting common features of reads mapped to the same genomic positions; second, we present a highly configurable way for the quantization of per-base quality values, which takes their influence on downstream analyses into account. NGC, evaluated with several real-world data sets, saves 33–66% of disc space using lossless and up to 98% disc space using lossy compression. By applying two popular variant and genotype prediction tools to the decompressed data, we could show that the lossy compression modes preserve >99% of all called variants while outperforming comparable methods in some configurations.

INTRODUCTION

Current high-throughput sequencing (HTS) technologies enable the fast, accurate and affordable sequencing of long stretches of DNA, which adds genome scientific approaches, such as RNA-sequencing (RNA-seq) or whole genome sequencing to the set of standard laboratory methods. These technologies result in huge amounts of digital data that have to be processed, transferred, stored and archived, which includes ‘raw’ sequencing data and an even larger number of intermediate and final result files that are produced by pipelines of data analysis and manipulation tools. Such files store HTS data in different (pre-) processing states and associated metadata describing these data, such as read names or mapping quality values, using various file formats, for example, unmapped reads stored in FASTQ format, mapped reads stored in SAM/BAM or called variations stored in the VCF format.

In general, all such files are subject to differing data handling and storage policies that define, for example, where and how long these files are stored, how fast they have to be accessible or how secure this access has to be. The emerging field of personal genomic sequencing, for example, will result in large amounts of data with high security demands, but not necessarily fast access times. Note that such policies are influenced not only by practical considerations (such as available storage space) but also, for example, by legal constraints and privacy issues. It is the costs associated with processing, storage and transmission of these data, rather than the generation of sequencing data itself, that constitute a major challenge to HTS experiments today [compare with (1–3)]. This motivates the development of data compression algorithms specialized for the discussed file formats, and recent research in this field may be divided into three major, but overlapping, categories: (i) compression of genomic sequences as generally produced by re-sequencing experiments (1,4,5); (ii) compression of unmapped short reads (6–8); and (iii) compression of aligned read data (9–12).

This article falls in the latter category, namely by compression of aligned short reads stored in the popular SAM text file format (13). This data format stores not only short sequences of DNA characters (read bases) but also a lot of associated metadata, such as per-base quality values (*q*-values), read names or mapping positions. Along with this easily processable text format goes a compressed binary variant (BAM) that basically comprises a blocked gzipped version of SAM. Our tool, NGC, allows a more efficient compression of the data stored in SAM/BAM files by handling each contained data stream individually,

*To whom correspondence should be addressed. Tel: +43 1 4277 24024; Fax: +43 1 4277 24098; Email: niko.popitsch@univie.ac.at

using value transformations and compression algorithms that pay attention to the respective value distributions. An overview of our solution is depicted in Figure 1. In this article, we mainly discuss the used compression approaches for two of these data streams, namely read bases and *q*-values and briefly sketch our strategies for encoding read names and alignment positions.

Our proposed method for the (lossless) compression of read bases builds on the wide-spread idea to store such data relative to some reference sequence (5,9,10,15,16). However, we propose to traverse the bases in an alignment of reads in a per-column way that exploits common features of multiple mapped reads rather than handling each read individually as done in previous research. This leads, ultimately, to a reduction of required code words and, in consequence, to a more efficient data compression. We measured the achieved compression rates (the required bits per sample) and overall compression ratios (the ratio between compressed and uncompressed size. The smaller this ratio, the better) and compared them with related tools.

Regarding the compression of *q*-values, we contribute a detailed discussion of several possibilities for their lossy compression and analyse the impact of the associated information-loss on subsequent data analysis pipelines. We propose a novel way for lossy q-value compression that distinguishes between different categories of *q*-values and is able to preserve the original qualities of bases in selected columns, which are the main targets of variant-calling and genotype prediction algorithms. We have evaluated our lossy per-base quality value compression using variant calling pipelines composed of state-of-the-art analysis tools and found that our proposed methods may preserve 99–100% of all called variants on average while outperforming comparable methods. Our evaluation included *Homo sapiens*, *Mus musculus*, *Escherichia coli* and *Arabidopsis thaliana* data from exome, whole genome, ChIP and RNA sequencing experiments.

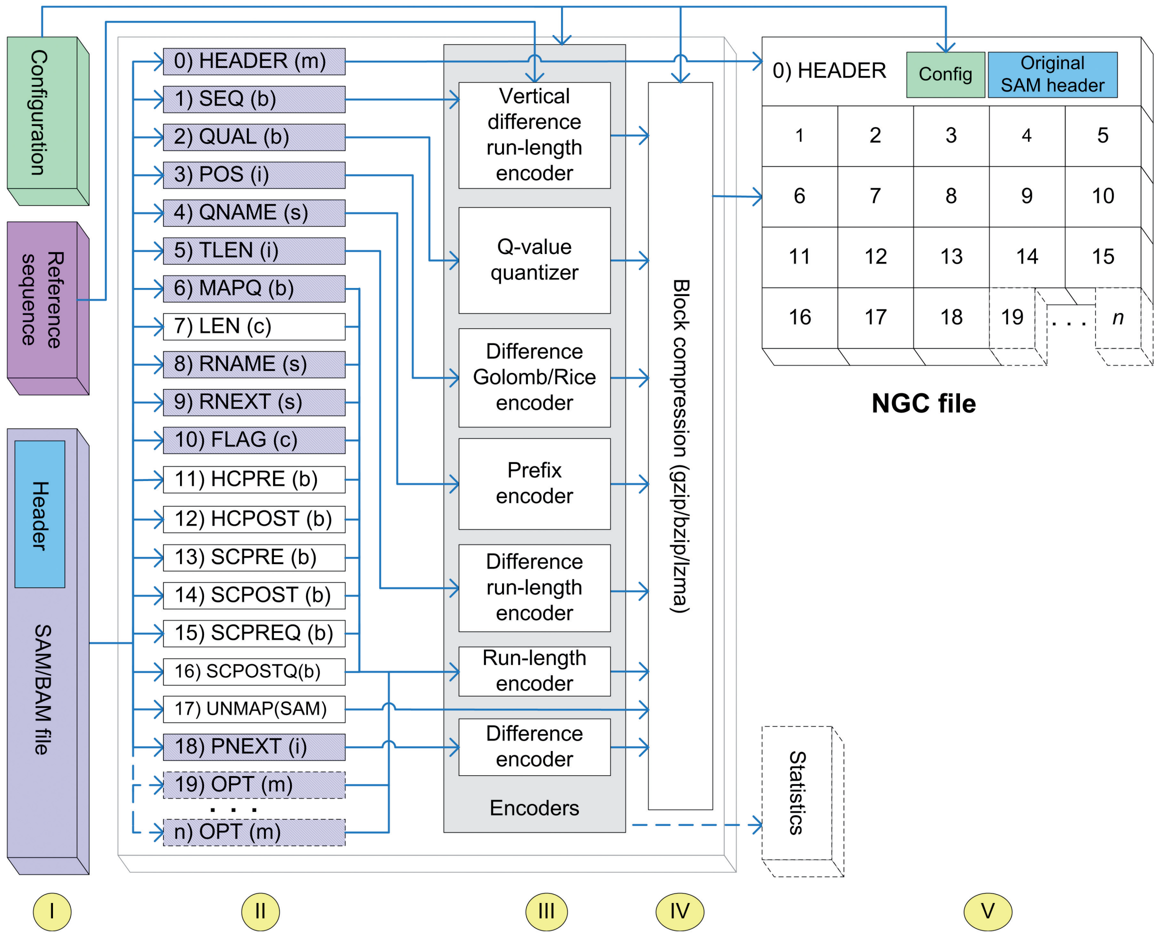


Figure 1. Schematic overview of the NGC compression approach. NGC takes a SAM/BAM file, a reference sequence and a set of configuration parameters as input (step I) and generates an NGC file and an optional statistics file (optional components drawn with dashed lines). First, NGC de-multiplexes the various data streams and adds some additional streams (step II). These streams are then passed to our various encoders that transform their values and prepare them for the subsequent block-compression (step III). In step IV, the data is compressed using a general-purpose compression algorithm (gzip, bzip2 or lzma). Finally, the compressed data blocks, the original SAM file header, and the required configuration information are combined to one single output file (step V). Streams with dark background (step II) encode the information described in (14), although with different encoding schemas. Streams with white background encode: 7: read lengths, 11–16: data required for reconstructing clipped bases/*q*-values, 17: unmapped reads. The basic data types of the streams are written in parentheses: (s)tring, (i)nteger, (c)har, (b)yte, (m)ixed or SAM format.

MATERIALS AND METHODS

Data sets and software availability

The used evaluation data sets are deposited in the Sequence Read Archive (3) under study numbers/run accession numbers: ChIP-Seq (mouse): SRX014899/SRR032209, Reseq/hm (human): SRX000376/SRR001471, RNA-Seq (*E. coli*): ERX007969/ERR019653, Reseq (*E. coli*, paired end): ERX008638/ERR022075, Reseq (*E. coli*): SRX118029/SRR402891, Reseq (*A. thaliana*): SRX011868/SRR029316. The human exome-sequencing data set was kindly given to us by B. Streubel. The Reseq/chr20 (human) data set is a resequencing data set of human chromosome 20, available from the GATK resource bundle (compare with Supplementary Materials).

The data sets were mapped using BWA v0.6.1 (17), with standard parameters for single paired end data. Unmapped reads were pruned from the data sets; variants were called with GATK v1.4 (18) and SAMTOOLS v0.1.18 (13), using the parameter settings given in the Supplementary Materials. NGC was implemented in Java 1.7 and is available for non-commercial use at http://purl.org/lsdv/ngc.

NGC

We have developed a tool (NGC) that enables the complete lossless and lossy compression of alignment data stored in SAM/BAM files. The NGC compressor takes an alignment file, a reference sequence and several configuration parameters as input and outputs a compressed file (Figure 1). On the other hand, the NGC decompressor reverses this operation. When the lossless mode of NGC is used, the resulting file is semantically equal to the original file in the sense that it contains the exact same information, although it might not be byte-equal as (i) the order of optional SAM fields is not preserved, which results in different byte streams in the resulting BAM file and (ii) the SAM fields MD ('mismatching positions') and NM ('number of mismatches') are dropped at compression time and automatically recalculated at decompression time, which may result in slightly different values because of ambiguities in the SAM specification. When one of the multiple lossy modes is chosen, the quality values in the resulting file additionally differ (partly) from the original ones.

NGC treats each read in the original file as an n-tuple of values of differing data type. Read names, for example, are of type string, alignment coordinates are of type integer, mapping quality values are of type byte and so forth. Such an n-tuple consists in principle of the 11 mandatory and all optional SAM fields [compare with (14)]; however, for algorithmic reasons, there are some deviations, for example, we do not store the CIGAR information, but reconstruct it from the other data and store some additional metadata, such as read lengths (Figure 1). NGC treats each element in these n-tuples individually, that is, each data field may undergo independent steps of value transformations and compression. Individual fields may even be dropped during the compression phase (e.g. NGC enables users to prune the read names from the data to save space. New read names will then be auto-generated at decompression).

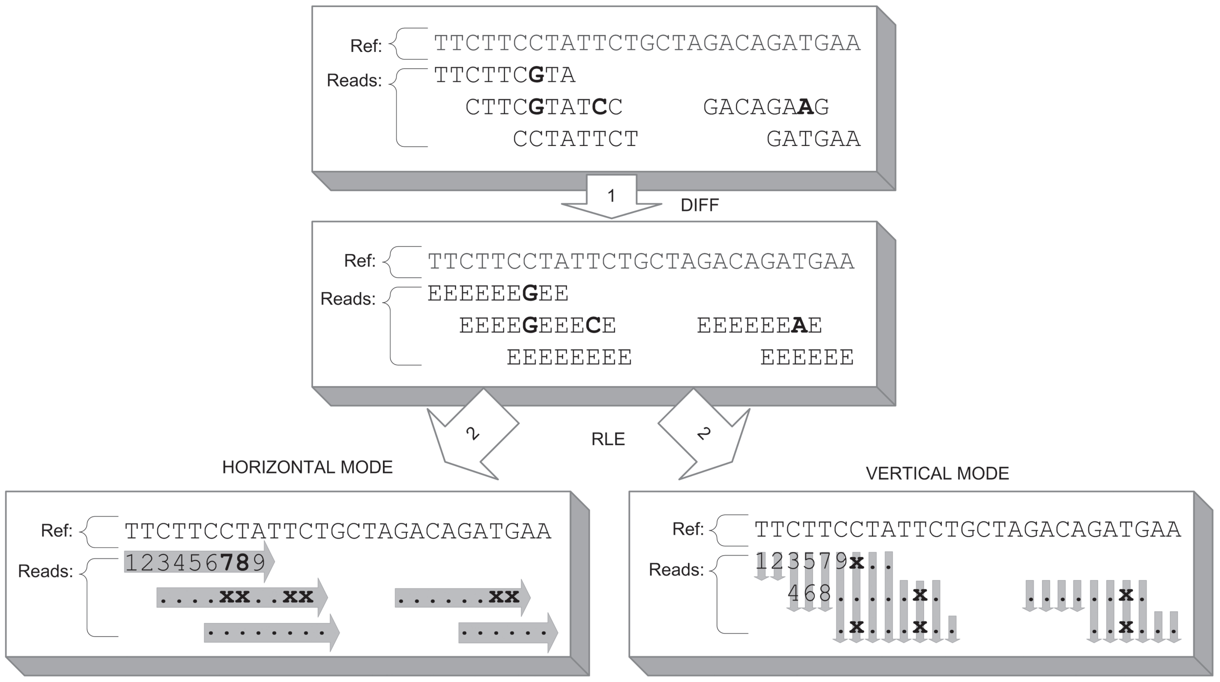
Figure 1 gives an overall picture of our software and shows the default encoding of the different data streams. In this paper we describe the four streams that take up most of the space in compressed data files in more detail: read base sequence information, per-base quality values, read positions and read names.

Read base compression

We call our idea for the compression of read bases 'vertical difference run-length encoding' (VDRLE). Generally, run-length encoding (RLE) is a simple encoding scheme for sequences of characters from some alphabet $\mathcal{A}$ (in our case the IUPAC nucleotide single-letter codes). A run-length (RL) is basically a pair of a single character from this alphabet and a positive integer number indicating the run length of this character in the sequence, an RLE is then an ordered sequence of such RLs. For example, a DNA sequence $S=$ '$AACTTT$' is encoded by the RLE $\{(A,2),(C,1),(T,3)\}$. The general compression idea of RLE is that long stretches of identical characters can be represented by one single RL that requires storing only the two RL symbols. Obviously, this simple encoding strategy is not effective for DNA sequences, as most RLs would be short, and a large number of such pairs would be needed to encode the sequence. The situation changes, however, when the encoded sequence contains long stretches of identical characters, as in this case, much less RL pairs than characters would be required. Our proposed compression algorithm for read bases is based on reducing the number of required RLs that may then be effectively encoded using well-known coding schemas and compression algorithms, respectively.

The first step of VDRLE is similar to other reference-based compression approaches: we do not encode a read sequence S itself but rather its differences to some reference sequence R. For this, let us formally introduce a new character 'E' to a now extended alphabet $\mathcal{A}_e = \mathcal{A}+\{$ 'E'$\}$ that represents that a character in S is unchanged in comparison to R. Now, we calculate a sequence 'diff', using the function $\Delta : A^n \times A^m \rightarrow A_e^n$ that replaces all characters in the first sequence that map to the same characters in the second sequence with this special character. Thus, $S' = \Delta(S,R)$ is constructed by replacing all characters in S that map to equal characters in the reference sequence R with 'E' characters (compare with Figure 2, step 1). It is easy to see that one may expect more and longer runs of 'E' characters in S' the higher the similarity between S and R.

Our improvement to existing RLE-based approaches is to encode such diff-ed read bases in an alignment 'vertically' [i.e. position after position or 'columnwise', compare with (19)] instead of 'horizontally' (i.e. read after read). This means that the stream of base characters that is encoded is not simply a concatenation of read bases but first lists all base characters that are mapped to position 1, then the ones mapped to position 2 and so forth (similar to the SAMTOOLS *mpileup* command). Figure 2 summarizes our approach. The figure shows how Δ(S,R) is



	Reseq/chr20 (human)	Reseq/hm (human)	Reseq (*A. thaliana*)	Reseq (*E. coli*)	Reseq (*E. coli*, PE)	Exome-seq (human)	RNA-seq (*E. coli*)	ChiP-seq (mouse)
Horiz. DRLE counts	43,532,619	900,546	13,341,776	4,303,863	191,384,090	3,386,907	56,295,485	8,127,608
Vert. DRLE counts	28,855,452	824,683	8,912,294	3,587,923	48,162,138	2,123,039	1,348,494	6,364,152
Ratio [%]	0.66	0.92	0.67	0.83	0.25	0.63	0.02	0.78

Figure 2. Vertical and horizontal difference RLE. The figure shows the difference for a simple alignment (top of figure). In step 1, all bases that match the reference are replaced by the 'E' character. In step 2, the resulting read bases are run-length encoded. The left box shows a 'horizontal' way to do this: the bases of each read (sorted by alignment position) are enumerated (we show the indices of the first nine bases in the figure), and a new RL is started whenever a base differing from the previous one is encountered (we have marked such positions boldface). The lower-right box shows our vertical approach: the read bases are enumerated column-after-column. Again, the start positions of RLs are marked boldface. Vertical encoding saves two RLs in this toy-example, the table below the figure shows the counts for horizontal and vertical difference run-length encoding (DRLE) in our test data sets. Figure 3 shows an excerpt of one of these data sets.

calculated in the first step by replacing all bases in the individual reads that match the respective reference bases. Our algorithm then generates a stream of base characters by traversing such a diff-ed alignment columnwise and encodes this stream by RLE. A new RL is generated whenever a different base than the last encoded one is encountered. For actually storing these RLs, we simply map all characters in our alphabet to byte values and write them, followed by one byte representing the RL length, to a standard byte stream. This stream is finally compressed using a general-purpose compression algorithm (users may choose between bzip2, gzip, lzma and no block compression).

Note that the described VDRLE method exploits common features between reads mapped to the same genomic region to decrease the number of required code words to be encoded. It exploits the fact that short-read mapping software tries to minimize differences between reference sequence and mapped reads, which often results in the same bases differing from the reference sequence being stacked above each other rather than next to each other, regardless whether these bases are real variants from the reference sequence or just sequencing artefacts.

Quality value compression

As others have reported before, we also found per-base quality values (*q*-values) hard to compress because of their quasi-random distributions, the high entropy of these data that results in rather high (undesirable) data compression ratios (7,11,12,20). Although we have tried several approaches to improve lossless q-value compression, including an adaptive arithmetic coding approach (21) that builds a statistical model per read position, we did not reach significantly better compression rates over all data sets in comparison with the general-purpose bzip2 algorithm (data not shown). We, therefore, currently do not apply any value transformation [not even diff-encoding or GapTranslating as advocated in (20)] in our lossless mode, but we encode the raw data using bzip2 with parameter settings that result in its best compression (i.e. using the '-9' switch). With our data sets, bzip2 achieved compression rates of 1.8–3.7 bits/q-value, which compares well with other reported numbers [compare with (11,20)].

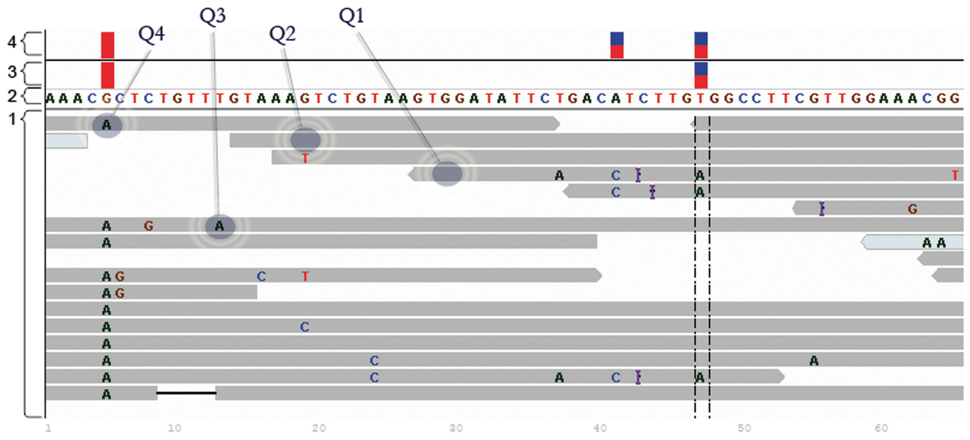
This rather bad compressibility of *q*-values is, however, the current main challenge for our compression method, which becomes clear when considering their large fractions (82–97%) of the overall file sizes of losslessly compressed data sets as shown in Table 1. It was, however, proposed by several authors in the recent past that it might be feasible to store *q*-values in a so-called lossy manner, that is, to discard some information in favour of better compression ratios (11,10,20).

Lossy *q*-value compression

A central step in most lossy data compression methods is quantization. Quantization of *q*-values maps the set of possible Phred quality values [e.g. 0–93 in Sanger format (23)] to a smaller set of values, usually by binning. Although this does not reduce the number of code words that have to be encoded, it greatly enhances the effect of subsequent probabilistic or dictionary-based compression methods, as there are much smaller ranges of possible code words. However, this loss of information also affects the results of downstream applications that make use of *q*-values, such as software for the removal of polymerase chain reaction (PCR) duplicates, for variant calling or for genotype prediction. It is a common goal of lossy compression approaches to find a good trade-off in this regard. There are many possible q-value quantization schemas that differ in complexity and in their influence on compression ratios and downstream effects. NGC uses a simple binning strategy that maps all *q*-values that lie within an interval to some single value within this interval (e.g. its upper or lower border). Such possibly non-uniform quantization intervals should be disjoint and should cover the whole range of input values. Note that extreme cases of this approach are to use a single interval that spreads the whole input value range and, thus, assigns a single value to all *q*-values or to use as many intervals as there are possible *q*-values, which basically results in no quantization at all.

Our quantization method does not treat all *q*-values in an alignment equally. It rather distinguishes between *q*-values of bases that (i) match or mismatch the reference sequences and (ii) reside in single- or multiallelic alignment columns (i.e. columns that contain one multiple different base characters in the read sequences). The resulting four possible q-value categories (annotated in Figure 3) are as follows:

- Q1: *q*-values of bases that match the reference and occur in columns where all bases match.
- Q2: *q*-values of bases that match the reference and occur in multiallelic columns.


Q4
Q3
Q2
Q1
4
3
2
1
AAACGCTCTGTTTGTAAAGTCTGTAAGTGGATATTCTGACATCTTGTGGCCTTCGTTGGAAACGG
1
10
20
30
40
50
60

	Reseq/chr20 (human)	Reseq/hm (human)	Reseq (*A. thaliana*)	Reseq (*E. coli*)	Reseq (*E. coli*, PE)	Exome-seq (human)	RNA-seq (*E. coli*)	ChiP-seq (mouse)
Q1 counts	3,873,164,564	61,831,496	303,452,695	491,077,181	42,556,735	256,925,783	31,453,173	472,315,173
Q2 counts	807,325,823	119,513	109,224,800	204,412,364	**4,388,981,896**	65,240,735	**188,284,057**	21,246,748
Q3 counts	22,271,926	155,855	5,702,544	2,158,208	102,563,275	1,641,199	29,657,344	2,348,243
Q4 counts	1,582,569	324,201	1,089,650	6,578	294	113,933	3,625	1,769,643

Figure 3. Excerpt of an alignment to illustrate our VDRLE approach for read base compression. The figure is a modified screenshot of region chr1:121484997–121485088 in the Reseq/hm (human) data set made with IGV 2.0.17 (22). It shows (from bottom to top): the reads with only the bases that differ the mapping reference shown, INDELs are drawn as I-characters or horizontal lines [1], the mapping reference [2], variants called by SAMTOOLS [3] and GATK [4], respectively. Our read sequence compression creates RLs by traversing this alignment columnwise (see text). The present alignment could be encoded with 75 horizontal, but only 55 vertical, RLs. Note that in NGC, however, the reads would be sorted strictly by coordinates; thus, the results would be slightly different. The marked bases/*q*-values are: Q1—bases that match the reference in columns that contain no alternate allele; Q2/Q3—bases that match/mismatch the reference in multiallelic columns; Q4—bases that mismatch the reference in columns that contain only one kind of alternate allele. The table below the figure lists the counts of the respective *q*-values in our evaluation data sets. It is notable that in two data sets, the number of Q2 values exceeds the number of Q1 values significantly (printed boldface), which is ascribable to the respective experimental settings.

Table 1. Evaluation data sets

	Reseq/chr20 (human)	Reseq/hm (human)	Reseq (*A. thaliana*)	Reseq (*E. coli*)	Reseq (*E. coli*, PE)	Exome-seq (human)	RNA-seq (*E. coli*)	ChiP-seq (mouse)
Mapped reads	50 663 069	487 201	11 651 942	19 379 287	44 938 891	3 239 217	6 927 728	13 824 441
BAM size (MB)	5868	53	504	655	2945	199	177	637
Base/q-value count	5.21×10^9	6.24×10^7	4.19×10^8	6.98×10^8	4.54×10^9	3.24×10^8	2.49×10^8	4.98×10^8
Average read length	101	128	36	36	101	100	36	36
Coverage	81.19	0.02	3.51	150.37	978.26	0.10	53.75	0.18
q-value % of total file size (BAM)	0.41	0.35	0.49	0.64	0.66	0.64	0.71	0.44
q-value % of total file size (NGC lossless)	0.53	0.82	0.88	0.97	0.89	0.95	0.97	0.86
Lossless compression ratios								
NGC	0.60	0.32	0.44	0.50	0.57	0.52	0.55	0.40
Cram	0.69	0.45	0.47	0.52	0.78	0.65	0.59	0.44
Goby	[a]	[a]	0.72	0.75	0.85[b]	0.75	0.83	0.66
Comp./decomp. times (lossless) (min)								
NGC	150/95	8/7	26/8	41/10	89/55	12/4	17/72	38/14
Cram	26/65	1/1	3/6	4/9	20/50	1/3	1/3	3/8
Goby	[a]	[a]	8/13	11/20	135/170[b]	3/5	4/7	9/16
Lossy compression ratios								
NGC m4	0.29	0.07	0.07	0.02	0.08	0.03	0.02	0.06
Cram-lossy	0.35	0.08	0.10	0.03	0.22	0.05	0.03	0.09
Comp./decomp times (lossy) (min)								
NGC m4	94/71	8/7	21/6	32/6	43/37	9/2	14/137	32/11
Cram-lossy	27/56	1/1	2/5	3/6	16/34	1/2	1/2	3/6

The table provides some data describing our evaluation data sets and compares the compression ratios (i.e. compressed divided by uncompressed size) and compression/decompression times of NGC, cram and goby for lossless and lossy compression (commandline parameters for the various modes are given in the Supplementary Materials). Read names were not included in these data sets; all BAM files contained tags that were preserved during the decompression/compression. The bottom section of the table compares compression ratios and times for NGC's m4 mode and cram's lossy configuration (see text). Absolute numbers for all lossy modes are provided in Supplementary Tables S1 and S2.

[a]These data sets could not be properly de-/compressed with goby.

[b]We had to increase the maximum heap size for compressing this data set with goby, see Supplementary Materials.

- Q3: *q*-values of bases that mismatch the reference and occur in multiallelic columns.
- Q4: *q*-values of bases that mismatch the reference and occur in columns where only this allele occurs.

Our general idea is that *q*-values of distinct categories have also differing impacts on downstream analyses. For this reason, our approach allows the application of different quantization schemas to *q*-values of differing categories. We propose, for example, to use more fine-grained sets of quantization intervals for *q*-values of substituted bases (Q3, Q4) rather than for the ones that match the mapping reference, as the latter ones have much less impact on the results of downstream applications as described later in the text. Our tool further provides the possibility to 'lock' an arbitrary number of columns in the alignment, so that the original *q*-values will be retained in these positions. This is useful, for example, in re-sequencing experiments where users already know the positions of some (expected) variations beforehand. The counts of the four q-value categories in our evaluation data sets are shown in the table in Figure 3. These numbers reveal that Q1 and Q2 values of the categories are naturally much more frequent than Q3 and Q4 values, which makes them the primary targets of our quantization strategies.

NGC supports two modes for storing quantized *q*-values. The first mode considers *q*-values in a horizontal way and stores them to a simple byte stream. The second mode traverses *q*-values vertically and stores them RLE (comparable with the described VDRLE approach, however, without the 'diff-ing' step). This latter mode is more efficient when the majority of *q*-values are quantized, which leads to long stretches of equal *q*-values. Note that the resulting byte streams are subsequently compressed by some general-purpose block-compression method in both cases.

The final compression ratio of our lossy compression approach consequently depends on the present *q*-value distribution, the chosen interval-sets for the four q-value categories, the partitioning of *q*-values into these categories, an optional list of 'locked' alignment columns and the chosen block compression mode.

Evaluation

To find reasonable trade-offs between quantization strategies, resulting compression ratios and possible downstream effects, we conducted a comprehensive experiment. In this experiment, we compressed the data sets listed in Table 1 using different combinations of q-value quantization intervals. Then we decompressed the data and used a pipeline of state-of-the-art tools to call variants (SNPs, INDELS, etc.) and their predicted genotypes in the obtained BAM files. This pipeline contained steps for INDEL realignment, PCR duplicate removal and various variant filtering steps (for details, see Supplementary Materials). We further used two wide-spread variant calling tools, namely GATK (18) and SAMTOOLS (13) for variant and genotype prediction. The resulting variant sets were then compared with the ones found in the respective unquantized data sets that served as our 'gold standard'. We counted the number of recovered (true positive), lost (false negative) and additional (false positive) variants and the number of variants that differed in their predicted genotype. We further measured the compression ratio of the particular approach, which is defined as the ratio between compressed and uncompressed file size. Additionally, we compared our tool with cram (http://www.ebi.ac.uk/ena/about/cram_toolkit) and goby (http://campagnelab.org/software/goby/), two tools that also compress SAM/BAM files. Goby (we used its latest version v2.1) supports only lossless compression. Cram is based on the method described in (10) but was further developed, in the meantime, and we used its latest available version (v0.85) in our evaluation, treating it as an alternative lossy

Table 2. Evaluation modes

Evaluation mode	Q1	Q2	Q3	Q4	RLE?	Low MAPQ filter	Known variants?	Average % of recovered variants
Lossless					No			100.0
m1	Std				Yes			99.7
m2	Std	Std			Yes			98.9
m3	30	Std			Yes			98.7
m4	30	30			Yes			96.1
Cram-lossy	30	30[a]						96.1
m5	30	30	Std		Yes	20		95.8
Recovery	Std	Std			Yes		Yes	99.7
Drop all	30	30	30	30	Yes			92.7

The columns 'Q1–Q4' list which quantization strategy was used for which kind of *q*-values. An entry 'Std' refers to the proposed standard binning scheme that is explained in the text. The value '30' means that the respective *q*-values were dropped and assigned the constant value 30. The column 'RLE?' shows whether RLE was used for q-value compression; the 'Low MAPQ filter' column shows whether *q*-values of reads with a mapping quality lower than the given value were dropped and the 'Known variants' column shows whether a set of known variants (the ones called in the lossless data set) was used to lock *q*-values in the respective columns. The last column lists the achieved overall percentage of recovered variants. Note that this measure neglects the false positive rates of the respective methods and refers to the recovery precision and sensitivity measures in Figure 4 and to the Supplementary Tables S3–S5 for details in this regard. The 'Cram-lossy' mode in row 6 refers to the lossy configuration of cram we evaluated against. All other modes were realized by configuring our NGC tool. Note that the configuration of the recovery mode in row 8 was derived from the m2 mode.

[a]Note that cram-lossy preserves qualities of Q2 bases if there are two or more reads that differ from the reference at a particular column, which is a slightly different behaviour than in NGC's m4 mode where all Q2 *q*-values are quantized.

compression method. Cram enables users to preserve q-values of various categories selectively. It allows, for example, preserving only q-values of substituted bases or of insertion regions.

This leads to a number of quantization strategies that were evaluated by us as summarized in Table 2. We basically configured our tool to either quantize q-values by simply assigning a single value to them (maximum compression) or to use a simple 'standard' binning scheme:

$$q' = f_{quant}(q) = \begin{cases} q, & q < 2 \\ 2, & q < 10 \\ 15, & q < 20 \\ 30, & \text{otherwise} \end{cases}$$

This schema was derived by studying the descriptions of various SNP calling software that incorporate per-base quality values in their calculations. Note, however, that multiple other schemas might be useful and that an extensive study of this regard is beyond the scope of this work. For a direct comparison, we configured a mode (m4, see Table 2) that quantizes q-values as similar as possible to a lossy configuration of cram (parameters given in Supplementary Materials), called 'cram-lossy' in the remainder of this article. We further configured a mode 'recovery', where NGC was provided the VCF file obtained by calling variants with GATK in the unquantized data sets. NGC then 'locks' the respective columns and preserves its original q-values, and we were curious whether this could significantly improve variant recovery. We additionally configured one mode (m5) that maximized compression by using the most restrictive quantization scheme and additionally dropping all q-values of reads with a low mapping quality.

Other streams

For read positions (stream POS, Figure 1), we store the differences to the previous position Golomb/Rice-encoded as also proposed by other authors [compare with (9)]. We estimate the Golomb-parameter by first calculating the mean difference between two neighbouring read alignment positions by sampling and then applying the method discussed in (24). It is noticeable that Golomb-codes may result in long code words when encoding numbers that are much larger than this parameter because of the unary coding scheme used for encoding the quotient. Such large 'gaps' between reads occur, for example, between adjacent covered regions in exome sequencing data. Long unary prefixes are, however, effectively compressed by the subsequent block compression methods (e.g. bzip2).

Read names (stream QNAME, Figure 1) are usually systematic names that contain information, such as instrument and flowcell identifiers, run numbers or x/y coordinates. For these names, we have developed a simple encoder that looks for the longest common, stable prefix of the last n reads; this prefix is then encoded only once and for all read names, only the variable rest is stored until a new prefix is found. Our simple method requires ~3–5 bytes per read name (see Supplementary Table S2); however, this could be further optimized [compare with (12) for an alternative approach]. We believe, however, the information stored in these strings is rarely used after the mapping phase, and consequently our tool enables users to drop them. Decompressed files will then contain automatically generated read names. When not dropped, read names take up 8–19% of the file size in the losslessly compressed evaluation data sets that increases up to 30–85% when using lossy compression.

RESULTS

To verify that it results in less RLs when traversing an alignment 'vertically' rather than 'horizontally', we have counted the RLs in our evaluation data (see Figure 2). We found considerably less code words when compared with a 'horizontal' approach (up to 50X less). Further, we compressed all data sets with NGC and calculated how many bits per encoded read base were used by its VDRLE method. The results varied between 0.12 and 0.03 bits/base (Supplementary Table S2), and we believe that there is some potential left when optimizing the RLE encoder. As the read stream accounts for only ~1–2% of our total compressed file size; however, we focused on a different stream that accounts for ~35–71% in the original BAM files and for ~82–97% in our losslessly compressed files: the per-base quality values.

Data compression, decompression and variant recovery

We evaluated how well the different q-value quantization modes listed in Table 2 perform at preserving the variants called in the nonquantized data sets with our pipelines. For this, we compressed the data using the respective quantization schemas, decompressed them and called their variants in comparison with their mapping reference sequence. The resulting variant sets were then compared with the ones called in the respective unquantized data sets. The results of our evaluation are summarized in Figure 4 that contrasts the achieved compression ratios with the precision and the sensitivity of the chosen quantization strategy. It can be seen that our tool provides lossless compression ratios between ~0.3–0.6, which corresponds to space savings of 40–70%. We compared these measures with the lossless compression ratios of cram and goby, absolute numbers and a comparison are given in Table 1.

The compression ratios of the lossy modes depend mainly on the used quantization schemas for Q1–Q4 values. A comparison of m4 and cram-lossy reveals that we provide a more efficient lossy compression than the cram tool (Table 1) when we configure both tools to basically quantize the same q-values (although it has to be noted that cram-lossy preserves qualities of Q2 bases if there are two or more reads that differ from the reference at a particular column, which is a slightly different behaviour than in NGC's m4 mode where all Q2 q-values are quantized). The precision and sensitivity analysis also reveals to what extent q-value quantization has to be paid in terms of false positives and negative calls. Precision (aka positive predictive value) measures the

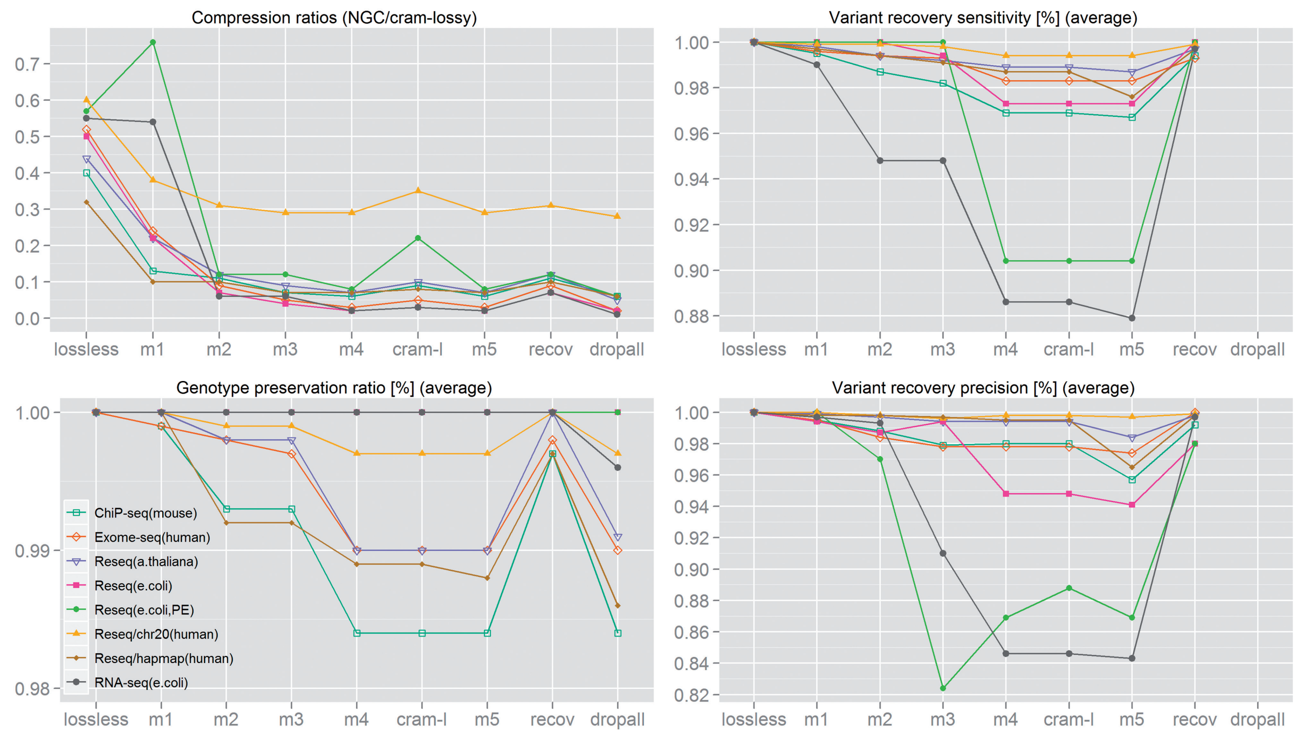


Figure 4. Evaluation data. The upper-left figure compares the compression ratios of the different modes for the used evaluation data sets (cram-l refers to the used cram-lossy mode). The absolute byte sizes and the compression ratios of the decompressed BAM files can be found in the Supplementary Table S2. Note that the Reseq/chr20 (human) BAM files are less compressible, as they contain large sections of optional BAM tags (compare with Supplementary Materials). The figures on the right side show the variant recovery precision and sensitivity averaged over the data we obtained by applying two different variant-calling tools (GATK and SAMTOOLS). We omitted the data for the ‘drop all’ mode in these two figures for readability, as the corresponding values would require a much larger scale. The lower-left figure shows how many of the re-found variants (i.e. the true positives) did not change their predicted genotype classification, either from homozygous to heterozygous or *vice versa*. All absolute data values and the used command-line parameters are given in the Supplementary Materials. Please note the differing scales of the *y*-axes in the four figures.

ratio between true positives and positive calls, sensitivity (aka recall rate or true positive rate) measures the ratio between true positives and the sum of true positive and false negatives (see Supplementary Materials). One extreme is the ‘drop all’ mode that simply replaces all *q*-values by a fixed value (data given in Supplementary Tables S3–S5). This mode primarily leads not only to a large number of false positive variant calls but also to much higher false positive rates than all other modes and consequently to low precision/sensitivity measures. Our proposed mode m1 (quantizing only Q1 values) results in high and stable (over all data sets) precision and sensitivity rates while considerably lowering the achieved compression ratios in comparison with lossless compression. The outliners for the m1 compression ratios can be explained when considering that the number of Q2 bases exceeds the number of Q1 bases for the RNA-seq (*E. coli*) and the Reseq (*E. coli*, PE) data sets (compare with Figure 2). This means that there are few columns that completely match the reference sequence, which is unfavourable for RLE that was used for this mode.

Better compression with slightly higher counts of false positives/negatives is achieved with m2, which uses our proposed quantization scheme f_{quant} for Q1 and Q2 bases. M4 basically quantizes the same *q*-values as cram-lossy but results in better compression, m3 lies between m2 and m4 in most measures. M5 provides maximum compression, which is, however, near the ratios achieved by m4. The recovery mode was configured equal to m2 but here we additionally passed the set of variants called from the respective lossless data set as parameter and did not quantize *q*-values at these positions. This resulted in only slightly worse compression but could increase precision and sensitivity for most data sets. For some, however, precision slightly dropped [e.g. Reseq (*E. coli*)].

Finally, we were interested in how well the individual quantization modes ‘preserved’ the predicted genotype of called variants. For this, we used GATK’s method to classify each found variant as homozygous, heterozygous or unknown. Then we counted how many of the variants that were redetected after compression/decompression with the respective quantization method (i.e. the true positives) changed their classification from homozygous to heterozygous or *vice versa*. In Figure 4, we plotted the percentage of these ‘preserved’ variants in the true positives. Not surprisingly, methods with the lowest compression ratios lead not only to more false positives and negatives but also to more variants with wrongly predicted genotypes, which might be an issue for use cases where these data is considered (e.g. when searching for loss-of-heterozygosity regions). It can also be observed that the recovery mode performed well in preserving these classifications, and it outperforms its nearest comparable mode (m2).

Computing times

We compared the compression and decompression times of NGC with cram and goby and list the times in Table 1. NGC is considerably slower than cram in both categories, although the differences are smaller when considering decompression. We attribute this mainly to the following two reasons: first, the VDRLE approach is much more time consuming when compared with traversing an ordered alignment read-by-read. Second, we have not yet optimized our tool in this regard, as we consider it as an archiving solution where computing times may not be the primary issue when they stay within practically useful limits. Although efficiently computing the compressed data may not be an issue, it is certainly helpful if the decompression of data is fast, and here, NGC is not substantially worse than cram and even faster than goby except for the RNA-Seq (*E. coli*) data set. The lossy compression/decompression modes of NGC and cram are commonly faster when compared with the respective lossless modes [Table 1, again, RNA-Seq (*E. coli*) being a considerable exception where the decompression times nearly doubled for NGC]. For some data sets, NGC's lossy modes compressed twice as fast and decompressed 66% faster (absolute times are given in the Supplementary Tables S1 and S2).

DISCUSSION

Continuously dropping costs per sequenced genome and technology advances, such as longer read lengths and shorter sequencing durations, will result in enormous amounts of data directly associated with HTS experiments. This issue cannot be ignored by referring to the also dropping costs per byte of secondary/tertiary storage as (i) the rate at which sequencing data is produced already overtook the rate of storage production (2,3); (ii) it is not only the data storage but also their increasing transmission, for example, between nodes in a cluster-/cloud-computing environment, between storage servers/long-term archives in a research facility or between research centres through the Internet, which imposes a challenge; (iii) data transmission leads in many cases also to data redundancies (e.g. cached parts, data copies) that further enlarge this 'sequence data heap'. All this will constitute a major bottleneck for HTS experiments in the future that can partly be widened when reducing file sizes by data compression.

Our proposed solution for the compression of SAM/BAM files contributes two major improvements to the state of the art. First, we propose a way to exploit common features of reads mapped to the same position to reduce the required number of code words. Second, we propose a model for the controlled lossy compression of per-base quality values that enables a comprehensive configuration of the trade-off between low compression ratios and high variant recovery rates.

Lossless compression with NGC may save between one-third and two-thirds of the disc space required for such alignment files, depending on characteristics of the input data. With lossy compression, however, we reach space savings up to 98%. In this article, we contribute a novel method for the compression of read bases in mapped alignments that encodes read bases losslessly using 0.12–0.03 bits/base. We achieve this by representing read bases in a vertical (per-column) way as also done by previous data formats, such as CALF (25) or the Ensembl Multi Format (EMF). However, different from those, we exploit common features between reads mapped to the same genomic position to reduce the number of required code words.

We further contribute a novel approach for the lossy compression of per-base quality values. These quality values account for up to 97% of the used disc space in our losslessly compressed files, which can be reduced to 2–33% with our proposed lossy modes. Note that our idea of VDRLE could also be used to compress whole sets of alignments that were mapped to the same genome (e.g. stemming from multiple resequencing experiments or from metagenomics analyses), and we plan to elaborate on this in future research. One possible improvement of our VDRLE approach is to further enhance the clustering of equal bases in (deep) alignment columns by applying a Burrows–Wheeler transform to them, compare with (26).

Lossy compression means a controlled loss of information that usually affects downstream data handling methods. Kozanitis *et al.* (11) have already shown that their model of lossy q-value representation shows only little impact on SNP calling with CASAVA, and somewhat comparable, yet in a more elaborated manner, we evaluated various q-value quantization strategies to find optimal trade-offs between compression rates and variant recovery. Different from this study, we used a more complex variant calling pipeline for evaluation that included INDEL realignment, duplicate removal and two widely adopted variant calling tools, namely GATK and SAMTOOLS. The idea of our proposed lossy modes is to preserve maximum precision in those parts of the data that are of high relevance to downstream applications. Variant-calling or genotype prediction software, for example, considers an alignment and its *q*-values also positionwise and will benefit from the maintained q-value characteristics in such columns. Note, however, that variant recovery in our evaluation depends not only on the actual variant calling tool but also on upstream processes that are influenced by *q*-values, such as the PCR duplicate removal.

Our evaluation showed that NGC outperforms the comparable cram and goby approaches with respect to lossless and lossy compression. Note that the BAM files resulting from decompression of our lossy compressed alignments are also considerably (up to 5X) smaller than the originals. This is because quality quantization will result also in improved compression of BAMs gzip blocks. These BAM files are, however, still considerably larger than the corresponding NGC files (see Supplementary Table S2).

The configurability of our tool further allows for lossy q-value quantization approaches that outperform cram-lossy with respect to higher variant recovery and higher genotype preservation rates (e.g. m1 or m2). Furthermore, cram currently suffers from not being able to reconstruct original read names (as of August 2012, the latest version of cram (v0.9) is also able to compress read names), whereas our tool can reconstruct BAM alignments completely, including the original file headers and all optional tags, and is independent of additional data, such as BAM index files. The current version of cram is, however, considerably faster than our tool when we compress data. Further, our data format does not allow efficient random access to portions of its data, as proper index structures are currently missing. As our tool was designed for archiving of BAM files, we do not consider this as a major obstacle for its application; we do, however, plan to improve NGC in this regard. Our q-value quantization methods also outperform the value transformation proposals in (20), who report compression rates of ~1 bit/q-value for a data set we also used in our evaluation. Other related work to NGC is the cSRA format that stores compressed-by-reference alignments in the Sequence Read Archive (3), SAMZIP (12), a tool that is specialized to compress whole SAM/BAM files but does not allow lossy compression and SlimGene (11) that compresses reads in the Illumina Export format.

Unmapped reads

As other reference based encoding methods, our proposed approach for sequence data compression cannot be applied to unmapped reads. We, therefore, currently store those in an own file block encoded in the original BAM format. We do, however, quantize their *q*-values in lossy compression modes (for this purposes, an own quantization scheme can be defined, i.e. an additional fifth q-value category was added). Fritz *et al.* (10) proposed the assembly of contiguous sequences in such unmapped reads to which these could then be mapped; however, their method resulted in additional mapping of only ~15% of those reads and consequently achieves only limited improvement of compression ratios.

CONCLUSIONS

The current development in the area of HTS data compression might, to some extent, be comparable with the situation of multimedia research shortly before audio, video and image capturing devices became available to the mass market; the application of the whole arsenal of available data compression algorithms led to the development of novel, specialized, lossless and lossy (e.g. JPEG or MP3) compression algorithms that ultimately enabled further progress in the development of capturing devices. It is our belief that efficient HTS data compression algorithms will play a comparable role in the field of nucleic acid sequencing. In this article, we presented one such solution for reducing the enormous data heap that is associated with today's HTS experiments.

SUPPLEMENTARY DATA

Supplementary Data are available at NAR Online: Supplementary Tables 1–5 and Supplementary Materials.

ACKNOWLEDGEMENTS

The authors wish to thank Heng Li and one anonymous reviewer for their valuable comments and suggestions that helped to improve the manuscript.

FUNDING

Wiener Wissenschafts-, Forschungs- und Technologiefonds (WWTF) in part. Funding for open access charge: CIBIV household budget.

Conflict of interest statement. None declared.

REFERENCES

1. Pinho,A., Pratas,D. and Garcia,S. (2011) GReEn: a tool for efficient compression of genome resequencing data. *Nucleic Acids Res.*, **40**, e27.
2. Kahn,S. (2011) On the future of genomic data. *Science*, **331**, 728–729.
3. Kodama,Y., Shumway,M. and Leinonen,R., International Nucleotide Sequence Database Collaboration (2012), The sequence read archive: explosive growth of sequencing data. *Nucleic Acids Res.*, **40**, D54–D56.
4. Cao,M., Dix,T.I., Allison,L. and Mears,C. (2007) *A Simple Statistical Algorithm for Biological Sequence Compression*, Data Compression Conference (DCC '07), pp. 43–52.
5. Wang,C. and Zhang,D. (2011) A novel compression tool for efficient storage of genome resequencing data. *Nucleic Acids Res.*, **39**, e45.
6. Tembe,W., Lowey,J. and Suh,E. (2010) G-SQZ: compact encoding of genomic sequence and quality data. *Bioinformatics*, **26**, 2192–2194.
7. Deorowicz,S. and Grabowski,S. (2011) Compression of genomic sequences in FASTQ format. *Bioinformatics*, **27**, 860–862.
8. Bhola,V., Bopardikar,A., Narayanan,R., Lee,K. and Ahn,T. (2011) *No-Reference Compression of Genomic Data Stored in FASTQ Format*. Proceedings of the 2011 IEEE International Conference on Bioinformatics and Biomedicine. IEEE Computer Society, Los Alamitos, CA, USA, pp. 147–150.
9. Daily,K., Rigor,P., Christley,S., Xie,X. and Baldi,P. (2010) Data structures and compression algorithms for high-throughput sequencing technologies. *BMC Bioinformatics*, **11**, 514.
10. Fritz,M., Leinonen,R., Cochrane,G. and Birney,E. (2011) Efficient storage of high throughput DNA sequencing data using reference-based compression. *Genome Res.*, **21**, 734–740.
11. Kozanitis,C., Saunders,C., Kruglyak,S., Bafna,V. and Varghese,G. (2011) Compressing genomic sequence fragments using SlimGene. *J. Comput. Biol.*, **18**, 401–413.
12. Sakib,M.N., Tang,J., Zheng,J. and Huang,C.T. (2011) Improving transmission efficiency of large sequence alignment/map (SAM) files. *PLoS One*, **6**, e28251.
13. Li,H., Handsaker,B., Wysoker,A., Fennell,T., Ruan,J., Homer,N., Marth,G., Abecasis,G. and Durbin,R. (2009) The sequence alignment/map format and SAMtools. *Bioinformatics*, **25**, 2078–2079.
14. *SAM*: *SAM Format Specification* (*v1.4-r985*) http://samtools.sourceforge.net/SAM1.pdf. (10 June 2012, date last accessed).
15. Brandon,M., Wallace,D. and Baldi,P. (2009) Data structures and compression algorithms for genomic sequence data. *Bioinformatics*, **25**, 1731–1738.
16. Christley,S., Lu,Y., Li,C. and Xie,X. (2009) Human genomes as email attachments. *Bioinformatics*, **25**, 274–275.
17. Li,H. and Durbin,R. (2010) Fast and accurate long-read alignment with Burrows-Wheeler transform. *Bioinformatics*, **26**, 589–595.

18. McKenna,A., Hanna,M., Banks,E., Sivachenko,A., Cibulskis,K., Kernytsky,A., Garimella,K., Altshuler,D., Gabriel,S., Daly,M. *et al.* (2010) The genome analysis toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Res.*, **20**, 1297–1303.
19. Giancarlo,R., Scaturro,D. and Utro,F. (2009) Textual data compression in computational biology: a synopsis. *Bioinformatics*, **25**, 1575–1586.
20. Wan,R., Anh,V. and Asai,K. (2012) Transformations for the compression of FASTQ quality scores of next generation sequencing data. *Bioinformatics*, **28**, 628–635.
21. Witten,I., Neal,R. and Cleary,J. (1987) Arithmetic coding for data compression. *Commun. ACM*, **30**, 520–540.
22. Thorvaldsdóttir,H., Robinson,J. and Mesirov,J. (2012) Integrative Genomics Viewer (IGV): high-performance genomics data visualization and exploration. *Brief. Bioinform.*, April 19 (doi:10.1093/bib/bbs017; epub ahead of print).
23. Cock,P., Fields,C., Goto,N., Heuer,M. and Rice,P. (2010) The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Res.*, **38**, 1767–1771.
24. Kiely,A. (2004) *Selecting the golomb parameter in rice coding*. IPN Progress Report. Jet Propulsion Laboratory.
25. Green,P. (2008) CALF (Compact ALignment Format), Version 0.081113. University of Washington, http://www.phrap.org/phredphrap/calf.pdf. (23rd July 2012, date last accessed).
26. Cox,A., Bauer,M., Jakobi,T. and Rosone,G. (2012) Large-scale compression of genomic sequence databases with the Burrows-Wheeler transform. *Bioinformatics*, **28**, 1415–1419.