GeneComp, a new reference-based compressor for SAM files

Reggy Long*, Mikel Hernaez*, Idoia Ochoa*†, and Tsachy Weissman*

*Department of Electrical Engineering
Stanford University, CA
and †Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign, IL
{reglong, mhernaez}@stanford.edu, idoia@illinois.edu, tsachy@stanford.edu

Abstract

The affordability of DNA sequencing has led to unprecedented volumes of genomic data. These data must be stored, processed, and analyzed. The most popular format for genomic data is the SAM format, which contains information such as alignment, quality values, etc. These files are large (on the order of terabytes), which necessitates compression. In this work we propose a new reference-based compressor for SAM files, which can accommodate different levels of compression, based on the specific needs of the user. In particular, the proposed compressor GeneComp allows the user to perform lossy compression of the quality scores, which have been proven to occupy more than half of the compressed file (when losslessly compressed). We show that the proposed compressor GeneComp overall achieves better compression ratios than previously proposed algorithms when working on lossless mode.

1 Introduction

Recently, there has been growing interest in genome sequencing, driven by advancements in the sequencing technology. Although early sequencing technologies required several years to capture a 3 billion nucleotide genome, genomes as large as 22 billion nucleotides are now being sequenced within days using next-generation sequencing technologies. Furthermore, the cost of sequencing a whole human genome has dropped from billions of dollars to merely $1000 within the past 15 years.[1]

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. As a result, increasingly large and ubiquitous genomic datasets are being generated. These datasets need to be stored, transmitted, and analyzed, which poses significant challenges.

In particular, there has been a lot of effort to ease the storage and distribution of these data sets. These data are mainly composed of the raw FASTQ files [1], and the aligned SAM files [2]. The FASTQ files are generated at the output of the sequencing machine, and they contain the nucleotide sequences, called "reads", the associated quality score sequences that indicate the reliability of each of the bases in a read, and the identifiers for each read. The SAM files are generated after aligning/mapping the reads present in the FASTQ file against a reference genome, and they contain the same information as the FASTQ file, together with the alignment information for

[1] https://www.genome.gov/sequencingcostsdata/

each read. Consequently, the size of a SAM file is generally up to twice that of the corresponding FASTQ file.

In this work we focus on the compression of SAM files, as there is a pressing need for compression of these files due to their prohibitively large size.

The SAM format is a TAB-delimited text format consisting of a header section, which is optional, and an alignment section. If present, the header must appear prior to the alignment section. Each alignment line has 11 mandatory fields and an arbitrary number of auxiliary fields. The mandatory fields always appear in the same order and must be present, but their values can be "0" or "*" (depending on the field) if the corresponding information is unavailable. We refer the reader to [2] for a comprehensive overview of the format.

Most of the proposed compressors in the literature focus on the compression of the SEQ and QUAL fields, as those are the fields that dominate the space requirement of compressed SAM files [3]. In addition, some of the remaining fields can potentially be reconstructed from the sequence itself, like the CIGAR string and some of the auxiliary fields, if present.

The SAM compressors can be divided into two main categories, those that rely on a reference sequence for compression and decompression, and those that do not. The state-of-the-art SAM compressors belonging to the first category are Scramble [4], NGC [5] and DeeZ [6], and the ones belonging to the second category are BAM [2] and CSAM [3]. Scramble and DeeZ also support reference-free compression, although the results are poorer in this case. Note that Scramble is a later version of the well known CRAM compressor [7], thus in the following when we refer to CRAM we actually refer to its later version Scramble. Finally, the recently proposed CARGO [8] framework supports of-the-self compressors for SAM files with or without the need of a reference sequence. Other than the use of a reference sequence, some of the main differences between the SAM compressors is the support of random access, complete lossless compression, and lossy compression for the quality scores. Table 1 summarizes the differences between the aforementioned algorithms. CRAM and NGC are not lossless because some fields in the SAM file are not compressed, and instead they are estimated during the decompression. As a result, those fields may differ from the original file. In addition, NGC does not preserve the order of the optional (auxiliary) SAM fields.

	CRAM	NGC	DeeZ	BAM	CSAM
Supports reference-based compression	Yes	Yes	Yes	No	No
Complete lossless compression	No	No	Yes	Yes	Yes
Lossy compression of quality scores	Yes	Yes	Yes	No	Yes
Random access	No	No	Yes	Yes	Yes

Table 1: Main features of the state-of-the-art SAM compressors.

Regarding lossy compression of the quality scores, only BAM does not support it. In CRAM, quality scores are binned according to a fixed table such that quality scores can take only 8 different values instead of the actual ones. NGC predefines four different binning tables, according to the different effects that their assigned quality

scores may have in downstream applications; then, NGC assigns each quality score to a binning table and bins them accordingly. Finally, DeeZ and CSAM perform a smoothening of neighboring quality scores, but while in DeeZ the smoothening is based on the global (or partial) statistics of the quality scores of the file, in CSAM it is based on the local statistics.

In this work we propose a new compressor for SAM files. The proposed method GeneComp uses a reference for compression and decompression, and offers lossless compression as well as lossy compression for the quality scores. Some of the main differences with previously proposed algorithms lie in the use of specialized models for compression of not only the reads and the quality scores, but the other fields as well, including the optional ones.

In particular, the compression of the reads is based on the work presented in [9], which benefits from a thoughtful modeling of the aligned reads. However, we use a different approach for compression. One of the caveats of the method presented in [9] is the use of the auxiliary field MD, which may not always be present. In addition, even if present, the MD field may be inaccurate, which leads to an incorrect decompressed file. To overcome this issue we propose an approach that does not rely on the MD field, and instead generates itself the mismatches between the read and the segment of the reference sequence where it maps.

For the quality scores, we use the compressed method QVZ [10], which offers both lossless and lossy compression. In addition, QVZ has been shown to produce variant calling results that are similar – and sometimes superior – to those obtained with the original file [11], making it a natural choice to compress quality scores. Note that QVZ was presented in isolation, that is, as an algorithm to solely compress the quality scores, and as a result, its practicality was limited. The proposed method GeneComp is the first one to incorporate QVZ in a complete SAM compressor.

We show that the proposed SAM compressor GeneComp achieves overall better lossless compression ratios than its competitors. Part of the advantage of GeneComp relies on the use of specialized models for all the fields that compose the SAM file. This is in contrast to other algorithms like BAM, CSAM, NGC, or DeeZ, which use general compression algorithms such as gzip to compress some of the fields present in the SAM file. In addition, GeneComp is the first algorithm to include QVZ for the lossy compression of the quality scores, which has been shown to produce variant calling results comparable to those obtained with the original file [11].

2 Proposed Compression Scheme

We describe the proposed method GeneComp for compression of SAM files, which assumes a reference sequence is available for the compression and the decompression. In addition, we consider the SAM file is sorted by position. As stated in the introduction, the bulk of the space occupied by compressed SAM files is generally due to the reads and the quality scores. Thus we focus mainly on describing the method used to compress the reads and the quality scores, and briefly describe the methods used to compress the remaining fields, the unmapped reads, and the headers.

2.1 Compression of the aligned reads and associated fields

Next we describe the method used to compress the reads, which are specified in the field SEQ. Recall that GeneComp assumes the reference sequence used for the alignment is available at the time of compression. Thus the reads can be represented by specifying the location at which they map to the reference, together with the edit information needed to perfectly reconstruct the read from the segment where it maps in the reference. Note that part of this information is specified in the fields RNAME, POS, and CIGAR, and in the auxiliary field MD, if present.

We noticed that the CIGAR field and the MD field, which allow to directly extract the edit information, are sometimes inaccurate. As a result, the decompressed file may not correspond to the original file, making it not lossless. In addition, some of these inaccuracies may lead in some cases to the compression algorithm crashing (for example, if a basepair in the reference sequence is different from the one expected based on the MD field).[2] Another consequence of these fields being inaccurate is that if not stored at the time of compression, and rather estimated at the time of decompression, the resulting reconstructed fields may be incorrect (thus leading to a reconstructed file that is not lossless).[3]

Thus to ensure lossless compression, the proposed method GeneComp does not rely in the CIGAR field and the MD field, if present, to extract the edit information. Instead, we compute the edit distance between a substring of the reference sequence and the read by using the Wagner-Fischer algorithm [12] with uniform costs for SNPs (Single Nucleotide Polymorphisms) and INDELS (i.e., insertions and deletions). We can then backtrack through the edit distance matrix to produce a sequence of edits (insertions, deletions, substitutions) that transform the reference sequence substring into the read. Note that this information, together with the location where the read maps to the reference, suffices to perfectly reconstruct the read. In addition, at the time of compression, we estimate the CIGAR field based on the edit information. If the estimated CIGAR field corresponds to the actual one, we encode a bit to indicate that. Otherwise, we encode the complement bit followed by the actual value of the CIGAR field. In this way, the decoder knows if its estimation is correct, or if instead, it needs to decode the actual CIGAR value. The MD field, is present, is compressed as it is (see below for how the compression of the auxiliary fields is performed).

Once the edit information is computed, we need to encode it together with the location where the read maps to the reference sequence. To this end, we use a context based approach, as suggested in [9], where they demonstrate that significant compression gains are possible thanks to the use of specialized models. The aim of the proposed models is to make use of the information of previously proposed reads to estimate the location where the current read maps in the reference, as well as the edit information. In other words, the idea is to exploit the redundancy that exists in the reads by using the previously compressed ones to predict the information needed to reconstruct the following read. These models are thus fed into an arithmetic encoder, which generates the bit stream. As one would expect, the benefits of this approach become more apparent as the coverage increases, since more coverage implies more redundancy present in the reads.

[2]This is the case, for example, of the aligned read compressor [9].

[3]This is the case of algorithms such as CRAM and NGC.

Next we describe how we store the information necessary to reconstruct the read (i.e., the position where the read maps and the edit information), and the models fed into the arithmetic encoder.

- An integer that indicates the position where the read maps in the reference. To encode it, we first compute the delta between the current position and the previous one. This transformation considerably reduces the size of the alphabet and reshapes the distribution, strongly biasing it towards low numbers. One of the challenges in compressing the delta values is the unknown alphabet. We use a modified order-0 Prediction by Partial Match (PPM) [13] scheme that contains an escape symbol used for previously unseen symbols.

- A bit that indicates the strand of the read. We use a binary model with no context, as the directions of the strands are almost i.i.d.

- A bit that indicates if the read maps perfectly to the reference or not. To compress it, we use as context the mapping position of the previous read, and a bit that indicates if the match was perfect or not. The rationale behind this approach is that we expect regions of the reference where no variations are found. In those regions, note that no SNPs or INDELS are likely to be present in the reads that map there, whereas the opposite is true for regions where variations occur.

- In case of a non-perfect match, we store either a "0", if the edit information includes INDELS, or the number of SNPs, if no INDELS are found. We use the previously seen number of SNPs as context, since closely mapping reads are expected to have a similar number of SNPs.

- If there are INDELS, we store the number of insertions, deletions, and SNPs. We model them similarly to the previous case.

- Positions of the insertions, deletions, and SNPs, within the read, if any. In this case we use several contexts for compression. First, we use a global vector that indicates, for each position of the genome, if a variation has been previously seen. We also use the position of the previous variation within the read, and the direction of the strand of the read, as contexts. The use of this context is intended to have a good estimate of where the current variations are going to occur.

- The new base pair for each insertion, if any. We use a model with no context to compress the new base pairs.

- The reference base pair and the new one for the substitutions, if any. We use as context the base pair observed in the reference, as the probability of having a SNP between non-conjugated base-pairs is higher than between conjugated base-pairs.

The models described above use previously seen information to estimate the next values to be compressed. As mentioned above, these models rely on the assumption that the reads contain redundant information, a fact particularly apparent in high coverage data sets.

2.2 Compression of the quality scores

The quality scores are compressed with the algorithm QVZ [10], which supports lossless and lossy compression. In particular, the user can specify a parameter $\alpha \in [0, 1]$, where $\alpha = 0$ is equivalent to using zero bits per quality score, and $\alpha = 1$ correspond to lossless compression. Any value in between scales the output file size by that amount. Thus the user can completely control the distortion level of the output file.

We chose QVZ as it has demonstrated superior rate-distortion performance than previously proposed algorithms [10]. In addition, for the recommended rate of 1 bit per quality score, QVZ has been shown to produce variant calling results that are comparable – and sometimes superior – to those obtained with the lossless compressed data. For a detailed analysis, we refer the reader to [11].

QVZ assumes the quality scores follow a temporal Markov model or order 1, based on the observation that quality scores are highly correlated with their neighbors within a single sequence. That is, given a quality score sequence $\mathbf{Q} = [Q_1, Q_2, \ldots, Q_l]$ of length l, QVZ assumes that the probability that the quality score Q_i takes a particular value depends on previous values only through the value of Q_{i-1}. Further, QVZ assumes that the quality score sequences are independent and identically distributed (i.i.d.).

The Markov model is defined by its transition probabilities $P(Q_i|Q_{i-1})$, for $i \in 1, 2, \ldots, l$, where $P(Q_1|Q_0) = P(Q_1)$. *QVZ* finds these probabilities empirically from the data to be compressed. If the quality scores are lossless compressed, then those probabilities are directly fed to an arithmetic encoder, which compresses the quality scores sequentially. Otherwise, these probabilities are used 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 [14]. After quantization, a lossless, adaptive arithmetic encoder is applied to achieve entropy-rate compression.

In summary, the steps taken by QVZ are to first compute the empirical transition probabilities of a Markov-1 model from the data. Then, use the Lloyd-Max algorithm to construct a codebook. Finally, use the codebook to quantize the input and feed into an arithmetic encoder.

Due to space constraints, we refer the reader to [10] for a detailed description of the codebook generation.

2.3 Compression of the remaining fields

The remaining fields correspond to QNAME, FLAG, MAPQ, RNEXT, PNEXT, TLEN, and the auxiliary fields, if any. Next we briefly describe how each of these fields is compressed.

- QNAME: This field is tokenized prior to compression, and each token is treated independently. We also check if a given token appeared in the QNAME of the previously compressed read, as some tokens tend to repeat across reads. Thus if a token is found in the previously compressed read, we compress a pointer to that location. Otherwise, the current token is compressed independently by means of an arithmetic encoder.

- FLAG: Compressed with an arithmetic encoder that uses a model based on the statistics from previously compressed values of the FLAG field.

- MAPQ: Compressed with an arithmetic encoder that uses a model based on the statistics from previously compressed values of the MAPQ field.

- RNEXT: It is common for the field RNEXT to take values "=" and "*", and thus we first indicate if one of these values occurred. Otherwise, we encode the value by means of an arithmetic encoder.

- PNEXT: We first indicate if the value is "0" or not, and if it is not, we encode the absolute distance between the value of PNEXT and that of POS, and a bit indicating the sign with an arithmetic encoder.

- TLEN: Similarly to PNEXT, we first indicate if the value is equal to "0". If not, we encode the value of TLEN with an arithmetic encoder.

- AUX: To encode the auxiliary fields, if any, we first create a list of most common auxiliary fields that we observe in the SAM file to be compressed. Thus when compressing an auxiliary field, we first indicate if it is contained in this list or not. If it is, we indicate what position in the list matches the auxiliary field, and encode it with an arithmetic encoder. Otherwise, the whole content of the auxiliary field is encoded, by separating it into TAG, TYPE and VALUE, and compressing each of them with an arithmetic encoder.

2.4 Compression of unmapped reads and headers

There are two types of unmapped reads, which we call type 1 and type 2. The first type of unmapped read has an RNAME that corresponds to a chromosome, and the second has an RNAME of "*". For the first type, we compress it using the same approach outlined above. We compress the latter type of reads using *gzip*.

The headers are stored uncompressed at the beginning of the compressed file.

3 Results and Discussion

In order to assess the performance of the proposed SAM compressor GeneComp, we compare it with the state-of-the-art compressors CRAM [4], NGC [5], and DeeZ [6]. We did not consider the algorithm CSAM [3], as it does not use a reference for compression. However, we do compare against BAM [2], even though it does not use a reference for compression, for being the current preferred method for storing SAM files.

We used two machines to perform the experiments because we were unable to compile DeeZ on the first machine. The first machine has 32 Intel Xeon E5-2658's clocked at 2.10 Ghz and 256 GB of RAM. The second machine is a 2015 Macbook Pro with an Intel i7 clocked at 2.5Ghz and 16 GB of RAM. All experiments that used the second machine are marked with an asterisk (*).

We selected for our study several datasets from the *Database for Evaluation of Genomic Information Compression and Storage* proposed by the Moving Picture Expert Group (MPEG) [15]. In particular, we selected the datasets *9799.sam*, *9827.sam*, *NA21144.chrom11* and *DH10B*.

3.1 Lossless results

Dataset	SAM	BAM	CramTools	NGC	DeeZ*	Scramble	GeneComp	Gain
9799	11538	2478	1583	-	1527	1609†	**1412**	8%
9827	21059	6548	3434	3829	3594	3398	**3198**	6%
NA21144	4504	1051	617	683	625	**622**†	630	-1%
DH10B	5579	1411	877	1070	870	873	**828**	5%

Table 2: Compression sizes for each dataset, rounded to the nearest Megabyte (base 10). The last column represents how our compressor compares to the smallest compressed size of the remaining compressors. We calculate gain using the expression $1 - (\text{our compressed size})/(\text{min compressed size of competitors})$.
†These results are taken from [4], since we were unable to run Scramble on these datasets due to the program crashing during the compression.

The compression sizes are outlined in Table 2. All four of the compressors outperform the BAM format, with a reduction in size up to 44%. In all datasets, our algorithm produces files whose sizes are comparable to or smaller than the all of the other methods. Importantly, our results show that we can preserve more information than the CRAM-based compressors, while still attaining better compression ratios.

When comparing the true lossless compressors, BAM, DeeZ, and GeneComp, we see that our compressor also performs comparably or outperforms the others on all datasets tested. The performance of the proposed method can be attributed to the use of specialized models, as well as to the good performance of QVZ for lossless compression of quality values (see [10]).

To further analyze these results, we show the relative and absolute compression performance for each part of the SAM file for our method. This breakdown is outlined in Table 3. As expected, quality scores occupy more than fifty percent of the compressed file. We also observe that auxiliary fields, although optional, can take a significant portion of the resulting file. Finally, it is worth noting that the biological information (that is, the reads themselves, the alignment information plus the pair-end information) only occupy about ten percent of the total size.

In terms of compression/decompression speed, the SAM to BAM conversion and Scramble compressors perform the best (see Table 4). The SAM to BAM conversion requires relatively less computation since it uses *gzip* along with some preprocessing. The version of Scramble used in the experiments uses the range-coder variant of the Asymmetric Numerical System, which has been shown to be faster than arithmetic

Category	Percentage	Absolute
QUAL	59	1884
AUX	18	585
QNAME	9	272
READ + ALIGN	6	288
INFO + MATE INFO	5	155
UNCATEGORIZED	4	14

Table 3: Absolute (in MB) and relative breakdown of each part of 9827.sam for GeneComp. The READ + ALIGN category includes the RNAME, POS, READ, and CIGAR, and the INFO + MATE INFO category includes TLEN, PNEXT, MAPQ, RNEXT, and FLAG. The last category contains all type 2 unmapped reads.

encoding [16]. Our compression and decompression times are middle-of-the-pack. However, our implementation is currently single-threaded, and our approach could be easily parallelized (i.e., compress each part of the SAM file in parallel).

Dataset	BAM	CramTools	NGC	DeeZ*	Scramble	GeneComp
9799	**0.5 (0.1)**	0.9 (2.2)	- (-)	**0.5** (0.6)	- (-)	0.9 (1.1)
9827	1 **(0.4)**	4.1 (35.8)	5.2 (3.4)	1.9 (1.2)	**0.4** (1.0)	2.3 (2.5)
NA21144	**0.1 (0.1)**	1.5 (1.8)	0.8 (-)	0.2 (0.16)	- (-)	0.5 (0.5)
DH10B	**0.1** (0.4)	0.4 (2.5)	1.3 (-)	0.3 **(0.3)**	**0.1 (0.3)**	0.7 (0.7)

Table 4: Compression and decompression times for each dataset in thousands of seconds. In each column, the first time is the compression time, and the time in parenthese is the decompression time. Times with a hypen (-) indicate that the program crashed while compressing/decompressing.

4 Conclusion

The widespread availability of low-cost sequencing techniques has led to an explosion of interest in personalized medicine and genome analysis, but has also led to overwhelming storage costs. In order to tackle these storage costs, we developed a new compressor for genomic data in the SAM file format. Our compressor is capable of both lossless and lossy compression, whereas other formats, such as NGC and CRAM-based compressors are not fully lossless. On the majority of the datasets tested, our method performs comparably or outperforms the state-of-the-art results in terms of compression ratio.

In the future, we would like to allow for random access of the compressed reads to facilitate genomic analysis without the prohibitive storage costs. This can be done, for example, by resetting the context for the arithmetic encoder every n reads. As shown in Table 3, there is room for improvement in compressing the AUX field.

5 Acknowledgement

The authors would like to acknowledge Carlos Navarro for his assistance and collaboration with improving the code. This work is partially funded by a fellowship from the Basque Government, a grant from the Center for Science of Information (CSoI), the 1 U01 CA198943-01 NIH grant, and the Stanford Data Science Initiative (SDSI).

References

[1] P. J. Cock, C. J. Fields, N. Goto, M. L. Heuer, and P. M. Rice, "The sanger FASTQ file format for sequences with quality scores, and the solexa/illumina FASTQ variants," *Nucleic acids research*, vol. 38, no. 6, pp. 1767–1771, 2010.

[2] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin *et al.*, "The sequence alignment/map format and SAMtools," *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, 2009.

[3] R. Cánovas, A. Moffat, and A. Turpin, "CSAM: Compressed SAM format," *Bioinformatics*, p. btw543, 2016.

[4] J. K. Bonfield, "The scramble conversion tool," *Bioinformatics*, p. btu390, 2014.

[5] N. Popitsch and A. von Haeseler, "NGC: lossless and lossy compression of aligned high-throughput sequencing data," *Nucleic acids research*, vol. 41, no. 1, pp. e27–e27, 2013.

[6] F. Hach, I. Numanagic, and S. C. Sahinalp, "DeeZ: reference-based compression by local assembly," *Nature methods*, vol. 11, no. 11, pp. 1082–1084, 2014.

[7] M. H.-Y. Fritz, R. Leinonen, G. Cochrane, and E. Birney, "Efficient storage of high throughput dna sequencing data using reference-based compression," *Genome research*, vol. 21, no. 5, pp. 734–740, 2011.

[8] Ł. Roguski and P. Ribeca, "CARGO: effective format-free compressed storage of genomic information," *Nucleic acids research*, p. gkw318, 2016.

[9] I. Ochoa, M. Hernaez, and T. Weissman, "Aligned genomic data compression via improved modeling," *Journal of bioinformatics and computational biology*, vol. 12, no. 06, p. 1442002, 2014.

[10] G. Malysa, M. Hernaez, I. Ochoa, and et. al., "QVZ: lossy compression of quality values," *Bioinformatics*, p. btv330, 2015.

[11] I. Ochoa, M. Hernaez, R. Goldfeder, T. Weissman, and E. Ashley, "Effect of lossy compression of quality scores on variant calling," *Briefings in bioinformatics*, p. bbw011, 2016.

[12] R. A. Wagner and M. J. Fischer, "The string to string correction problem," *Journal of the ACM*, pp. 21:168–178, 1974.

[13] K. Sayood, *Introduction to data compression*. Newnes, 2012.

[14] S. Lloyd, "Least squares quantization in pcm," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[15] MPEG, "Database for evaluation of genomic information compression and storage," *Joint AhG on Genomic Information Compression And Storage, Geneva, CH*, 2016.

[16] J. Duda, "Asymmetric numeral systems: entropy coding combining speed of huffman coding with compression rate of arithmetic coding," 2013.