INTERNATIONAL ORGANISATION FOR STANDARDISATION
ORGANISATION INTERNATIONALE DE NORMALISATION
ISO/IEC JTC1/SC29/WG11
CODING OF MOVING PICTURES AND AUDIO

ISO/IEC JTC1/SC29/WG11 MPEG2015/M36282
June 2015, Warsaw, Poland

Source	**Leibniz University Hannover (Institute for Information Processing)**
Status	**Input document**
Title	**Approaches to SAM File Compression**
Author	Jan Voges, Dipl.-Ing. Marco Munderloh (Leibniz University Hannover, Institute for Information Processing) / {jvoges, munderl}@tnt.uni-hannover.de

Table of Contents

1 Introduction

Due to novel high-throughput next-generation sequencing (NGS) technologies, the sequencing of huge amounts of genetic information has become affordable. On account of this flood of data, IT costs may become a major obstacle compared to sequencing costs. High-performance compression of genomic data is required to reduce the storage size and transmission costs.

Raw sequencing data (mainly the base sequences – also known as reads - and the quality scores) is stored in so-called FASTQ files, whereas mapped sequence data is stored in so-called SAM files. SAM files are plain-text human-readable files containing the reads and quality scores as well as mapping information with respect to some reference genome. Mapped sequencing data represented in SAM files contains more redundancy, as typically multiple reads are mapped to the same location on the reference genome. The average amount of reads mapping to the same location is referred to as *coverage*.

It has been shown by Tembe et al. [7] and Deorowicz and Grabowski [8], that splitting the data into separate streams for sequence reads, quality scores etc. (and compressing them independently) yields significant gains over general-purpose (dictionary-based [2]) programs like *gzip*. All recent contributions to the topic of FASTQ and/or SAM file compression employ this scheme of separately compressing the sequence reads and quality scores (as well as the other field present in FASTQ or SAM files).

Based on statistical analysis performed on the reference data set issued by this ad-hoc group during the 111th MPEG meeting in Geneva, new proposals for sequence and quality score compression in SAM files have been developed.

1.1 Outline

We propose a sequence compressor which assumes aligned and position-sorted data. Our sequence compressor combines the mapping positions, the CIGAR strings and the actual nucleotide sequences to implicitly assemble local parts of the donor genome and compress the sequence reads. The compressor does not need a reference genome and is able to perform compression “on-the-fly” using solely a *sliding window* as context for the local assembly.

Regarding the compression of quality scores, we propose a couple of compression schemes based on Markov models with a variable context to boost prediction performance. Due to the large alphabet of quality scores (and thus a huge memory consumption of any Markov model), we furthermore propose employing FIR filter-based compression for quality scores.

2 Software Framework

We developed a compression framework called *tsc* (C99-compliant) to evaluate compression mechanisms for the different fields present in a SAM file. *Tsc* splits the SAM file into separate streams for each field [1]. Dedicated compression algorithms can then be employed in a block-by-block manner on each field or on a combination of fields. New compression modules for e.g. quality score compression can easily be mounted.

We propose to use the *tsc* framework as a starting point for a standardization effort. Presumably, a combination of compression algorithms from various sources will yield the best overall compression ratio for SAM files as well as FASTQ files (the *tsc* framework can easily be extended to accept FASTQ files, too).

3 Block-based Compression of Nucleotide Sequences

3.1 Prior Work

Current implementations (e.g. *Quip* [6] and *sam_comp* [3]) use an order-2 arithmetic coder (AC) to compress the nucleotide sequences. In order to exploit the redundancy present in the data, an AC needs *a priori* knowledge. This practically excludes random access to the compressed data as the prediction performance increases with the amount of data the Markov model can be trained on.

The authors of *DeeZ* [5] made a new approach, called "local assembly" (implicit assembly of the donor genome using the mapping information present in SAM files). *DeeZ* nevertheless performs the local assembly with respect to an external reference (given in the FASTA format).

The authors of *Quip* [6] implemented an assembly-based compression scheme for nucleotide sequences, too. They use by default the first 2.5 million reads to assemble so-called contigs which are then used in place of a reference sequence to encode aligned reads. This algorithm does not require an external reference – this means that the resulting compressed files are entirely self-contained – but has the drawback that the best match of a specific read to the previously assembled reference has to be found.

3.2 Proposed Compression Mechanism

We propose a compression scheme using a sliding window to perform a "local block-based implicit assembly". In contrast to *DeeZ*, our compressor does not require a reference genome and is able to work on smaller block granularities as it uses a "sliding window" as context (of course smaller block sizes have a negative impact on compression ratio). In contrast to *Quip*, we do not need any reads to assemble so-called contigs in advance. We perform the compression using only the currently available nucleotide sequences in the sliding window. Our compression scheme therefore is less complex, but should nevertheless yield comparable compression ratios.

Our approach is based on correlations between consecutive reads. As all reads in a SAM file are aligned to their mapping position, all reads mapped to the same position should be similar, except for gene mutations (SNP's) or sequencing errors respectively mapping errors. These aberrations are coded in the CIGAR string.

Figure 1 and figure 2 illustrate the structure of mapped sequence reads. Consecutive reads are plotted line by line. All reads mapped to the same position are aligned. As SAM files are stored sorted by mapping position, a read mapping to a different location than the read before leads to a step in the plot. If plotted over a large block, this leads to the staircase property visible in the plotted figure.

Our approach proposes to encode sequence reads block-wise exploiting the joint coding of the mapping positions, the CIGAR strings and the sequence reads itself, whereas the block size might be fixed or variable and of arbitrary size. The first sequence read in a block is encoded without prediction, as well as the corresponding mapping position and the CIGAR string. However, it might also be coded against some reference. Some subsequent read $i, i > 1$, is then aligned to a previous read $j, j < i$, using its CIGAR string and its mapping position. Thereafter, we compute the difference *d* (containing the differences of the mapping position and the

nucleotide sequence from the aligned read *i* to read *j*) and pass it to an entropy coder, e.g. an order-0 arithmetic coder. At the decoder, read *i* can be reconstructed from read *j* by applying *d* to reconstruct the nucleotide sequence, the CIGAR string, and the mapping position.

The alignment of read *i* to read *j* is done with a function called expand. The CIGAR strings and mapping positions of both reads are used to unwind the sequence such that they would both directly align to some external reference (please refer to the SAM file specification [1] for a detailed explanation of the CIGAR string syntax). Consequently, expand condenses the position, the CIGAR string, and the sequence into a joint representative code word. As an alternative, these SAM fields could be compressed using separate entropy coders.

The previous read *j* is by default read $i-1$. This context read $j=i-1$ might be highly erroneous or not aligned to the reference. We address this issue by keeping track of *N* previous reads (sliding window) to be able to select the best matching read to encode read *i*.

Sequence reads which are not aligned to the reference might be directly passed to the entropy coder or retained and encoded separately later, e.g. at the end of each block.

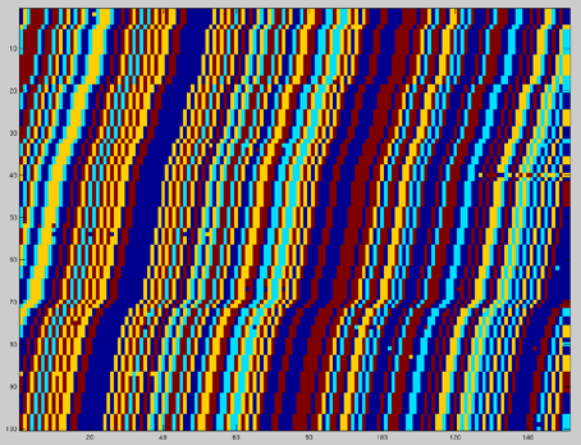

Figure 1: Consecutive nucleotide sequences from a SAM file. The different colors represent the nucleotides.

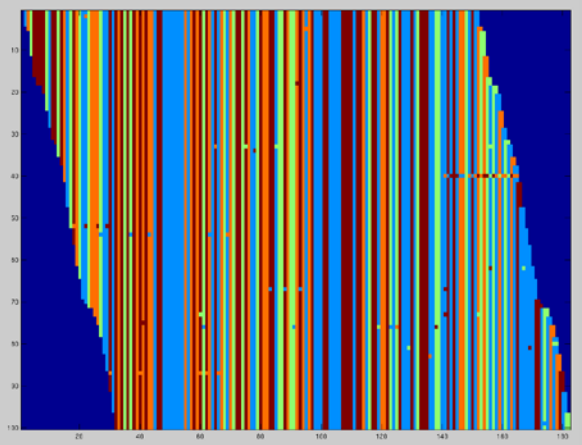

Figure 2: The nucleotide sequences after the `expand` function, stored as a sliding window in the circular buffer. The noisy 'dots' represent genome mutations or sequencing errors described in the CIGAR string.

Note: Per block, the initial read and the trailing substrings of subsequent reads can be regarded as locally assembled genome.

3.3 Results

First results have been obtained using data from the genomic information database issued by this ad-hoc group during the MPEG meeting 111 in Geneva.

The proposed sequence compression algorithm has been integrated into the *tsc* framework from section 2. We used a block size of 100,000 (sequence reads per block) and a sliding window holding 100 reads. Subsequent entropy coding is performed using an order-0 AC. The results are compared against *Quip* and *DeeZ*. *Quip* uses a high-order Markov chain. The nucleotide at a given position is predicted using the preceding twelve positions. *DeeZ* uses its own unique compression algorithm (implicit assembly of the donor genome).

Our algorithm (which does not require a reference genome) generally seems to be on par with *DeeZ*. Employing our algorithm on the MiSeq_Ecoli_DH10B and RNAseq datasets yields even better compression ratios than *DeeZ*. These results seem to be on account of the subsequent AC. The compression of the ERR317482 dataset yields a slightly poorer compression ratio than *DeeZ*. This might be on account of the relatively small sliding window we use in contrast to the full local assembly performed by *DeeZ*.

Dataset	**Compressed SEQ size**		
	tsc	**Quip**	**DeeZ**
MiSeq_Ecoli_DH10B	**2.22%**	33.11%	8.93%
ERR317482	**17.07%**	34.62%	12.21%
RNAseq	**7.58%**	31.94%	9.98%

4 Predictive Coding of Quality Scores

4.1 Prior Work

The quality scores yield the largest first-order entropy among all field present in a typical SAM file, mainly due to their large alphabet.

The authors of *Fqzcomp* [3] state the following dependencies for a sequence of quality scores:

- Any score q_i has a strong correlation to the direct previous quality score $q_{i-1}, q_{i-2}, \dots$, decreasing the further back we go [4].
- A known issue with the Illumina base-caller causes many sequences to end with quality score 2 ("#").
- There is a correlation between position and quality values. Quality values are typically reducing along the length of the sequence.
- Sequences as a whole tend to be good or bad.

The authors of *Fastqz* [3] additionally state that it was observed that the quality values tend to start with a common maximum value.

Current SAM compression implementations exploit this observations by employing a Markov model with the memory ranging over the direct predecessors in the quality score stream.

4.2 Observations

In terms of performing some statistical analysis regarding the quality scores, we converted the quality score lines into a single stream of quality scores.

A snapshot of some typical quality score lines and their histogram is shown in figures 3 and 4.

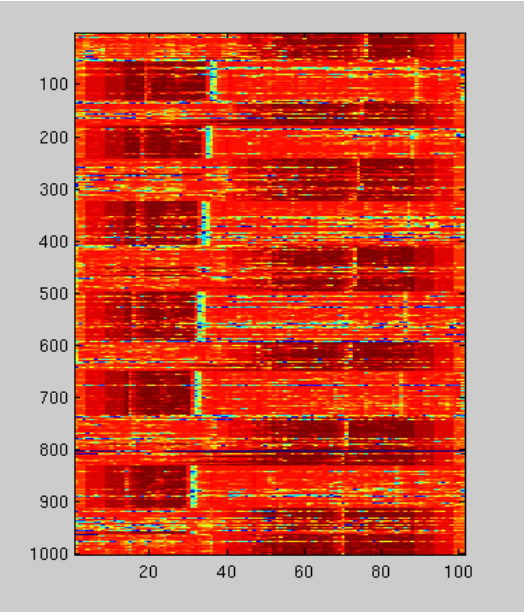

Figure 3: Snapshot of 1,000 quality score line from a SAM file. The data has been produced by an Illumina machine (constant read lengths).

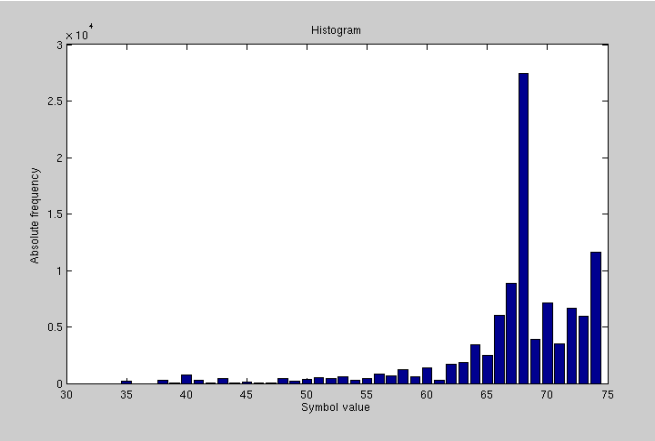

Figure 4: Typical quality score histogram

The autocorrelation function of such a stream is shown in figure 5. The auto-correlation does not resemble a delta impulse and thus the power spectral density is not constant. This means, that the source has some kind of memory. Thus, it is highly suitable to model the source with a Markov chain.

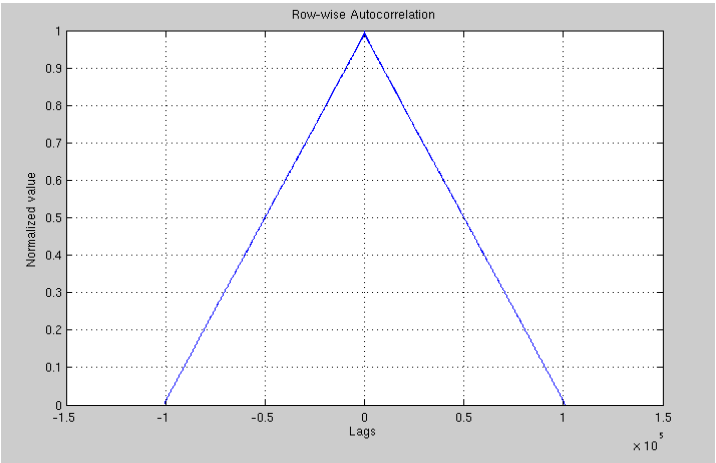

Figure 5: Autocorrelation function of a quality score stream

4.3 Modeling the Source with a Markov Chain

Current SAM file compressors model the quality score source as a Markov chain, predicting the current symbol with a maximum likelihood predictor using the *N* direct predecessors.

The maximum likelihood predictor selects in each state $q_1, \dots q_N$ the symbol a_i with the greatest conditional probability P as prediction value $q_{N+1,p}$:

$$P\left(q_{N+1,p} = a_i \middle| q_1, \dots q_N\right) \geq P(q_{N+1} = a_k | q_1, \dots, q_N) \forall k$$

4.3.1 Drawbacks and Recommendations

The maximum likelihood predictor consumes a huge amount of memory since we have to keep track of $k^N * k$ relative symbol frequencies (having an alphabet of size k). This model also needs a large amount of data to train in order to tightly fit the model to the data. We propose to store the prediction table (generated during compression) in the file header to speed up decompression. Prediction tables for the individual blocks could then be delta-encoded. This would enable random access to the compressed data while retaining the compression performance of the Markov model.

4.4 Proposed Compression Schemes

4.4.1 Dictionary-based Substring Matching Approach

The authors of *Fqzcomp* [3] state, that quality score lines as a whole tend to be "bad" or "good". Additionally, we made the observation, that in some datasets subsequent quality score lines are "similar" in terms of a small Levensthein distance between pairs of quality score lines.

We respond to this using a so-called line context, i.e. a structure holding some number of previous lines from the current block (this can be regarded as *sliding window*).

To encode the quality score $q_{i,j}$ in column j and line i, we propose to select the substring s of N preceding symbols $s = q_{i,j-N}, \dots, q_{i,j-1}$. We then propose to search for a similar substring $\hat{s} = q_{\lambda,\gamma}, \dots, q_{\lambda,\gamma+N-1} \equiv s$ in the line context and to use $q_{\lambda,\gamma+n}$ as the prediction value $q_{i,j}^{(p)}$ for $q_{i,j}$.

Furthermore, we propose to use a prediction value $q_{i,j}^{(p)}$ calculated as a mean or median from M substring matches $\hat{s}_m$, which might be weighted by their prediction error $\left|q_{i,j}^{(m)} - q_{i,j}^{(p,m)}\right|$, whereas any suitable norm might be applied.

4.5 Entropy Coding of Prediction Errors

The histogram of the prediction errors of the Markov or FIR predictor approximately shows a two-sided geometric distribution. We therefore propose to use subsequent Rice coding [9] as an alternative to arithmetic coding to obtain the binary representation of the data. This reduces the encoder complexity, as recent compressors such as *Quip* [6] employ arithmetic coding.

4.6 Results

First results have been obtaining using data from the genomic information database issued by this ad-hoc group during the MPEG meeting 111 in Geneva.

Similar to the proposed sequence compressor in section 3.2, the quality score compressors have been integrated into the *tsc* framework. Again, we used a block size of 100,000 reads per block and a sliding window holding 100 reads. The results are compared against *Quip* and *DeeZ* (using a modified order-2 AC respectively an order-2 AC).

"tsc O0" refers to a simple order-0 AC. "tsc O1" refers to the dictionary-based substring matching approach from section 4.4.1 with a subsequent order-0 AC.

Surprisingly, the order-0 AC shows the best compression performance. Every attempt to predict quality score values (either by substring matching ("tsc O1") or by employing higher-order AC's), yields poorer overall compression ratios.

Dataset	**Compressed SEQ size**			
	tsc O0	**tsc O1**	**Quip**	**DeeZ**
MiSeq_Ecoli_DH10B	**51.98%**	56.97%	57.63%	77.58%
ERR317482	**50.65%**	65.00%	53.59%	71.50%
RNAseq	**46.72%**	54.81%	53.92%	72.80%

5 Recommendations

Due to recent efforts made in the domain of lossy quality score compression, we recommend to further investigate lossy, predictive quality score compression with an emphasize on rate-distortion performance.

6 References

[1] The SAM/BAM Format Specification Working Group. Sequence Alignment/Map Format Specification

[2] Ziv., J. & Lempel, A. A universal algorithm for sequential data compression. IEEE Transactions on information theory 23, 337-343 (1977).

[3] Bonfield, J.K & Mahoney, M. V. Compression of FASTQ and SAM Format Sequencing Data. PloS ONE 8, e59190 (2013).

[4] Christos Kozanitis, Chris Saunders, Semyon Kruglayak, Vineet Bafna, and George Varghese. Compressing genomic sequence fragments using SlimGene. Journal of Computational Biology: A Journal of Computational Molecular Cell Biology, 18(3):401-13, March 2011. ISSN 1557-8666. doi: 10.1089/cmb.2010.0253

[5] Faraz Hach, Ibrahim Numanagic & S. Cenk Sahinalp. DeeZ: reference-based compression by local assembly. Nature Methods, Vol. 11, No. 11, November 2014, pp 1082-1084

[6] Daniel C. Jones, Walter L. Ruzzo, Xinxia Peng & Michael G. Katze. Compression of next-generation sequencing reads aided by highly efficient de-novo assembly. Nucleic Acids Research, 2012, 1-9. doi: 10.1093/nar/gks/754.

[7] Tembe, W., Lowey, J. and Suh, E. G-SQZ: compact encoding of genomic sequence and quality data. Bioinformatics, 26, 2192-2194, (2010).

[8] Deorowicz, S. and Grabowski, S. Compression of DNA sequence reads in FASTQ format. Bioinformatics, 27, 860-862. (2011).