Journal of Bioinformatics and Computational Biology
Vol. 12, No. 6 (2014) 1442002 (17 pages)
© Imperial College Press
DOI: 10.1142/S0219720014420025

Imperial College Press
www.icpress.co.uk

Aligned genomic data compression via improved modeling

Idoia Ochoa*, Mikel Hernaez† and Tsachy Weissman‡
Department of Electrical Engineering, Stanford University
350 Serra Mall, Stanford, CA, USA
**iochoa@stanford.edu*
†mhernaez@stanford.edu
‡tsachy@stanford.edu

Received 14 July 2014
Revised 12 September 2014
Accepted 17 September 2014
Published 13 November 2014

With the release of the latest Next-Generation Sequencing (NGS) machine, the HiSeq X by Illumina, the cost of sequencing the whole genome of a human is expected to drop to a mere $1000. This milestone in sequencing history marks the era of affordable sequencing of individuals and opens the doors to personalized medicine. In accord, unprecedented volumes of genomic data will require storage for processing. There will be dire need not only of compressing aligned data, but also of generating compressed files that can be fed directly to downstream applications to facilitate the analysis of and inference on the data. Several approaches to this challenge have been proposed in the literature; however, focus thus far has been on the low coverage regime and most of the suggested compressors are not based on effective modeling of the data.

We demonstrate the benefit of data modeling for compressing aligned reads. Specifically, we show that, by working with data models designed for the aligned data, we can improve considerably over the best compression ratio achieved by previously proposed algorithms. Our results indicate that the pareto-optimal barrier for compression rate and speed claimed by Bonfield and Mahoney (2013) [Bonfield JK and Mahoneys MV, Compression of FASTQ and SAM format sequencing data, PLOS ONE, **8**(3):e59190, 2013.] does not apply for high coverage aligned data. Furthermore, our improved compression ratio is achieved by splitting the data in a manner conducive to operations in the compressed domain by downstream applications.

Keywords: SAM file; compression; context modeling.

1. Introduction

Sequencing cost has drastically decreased in recent years.[1] While in 2004 the cost of sequencing a full human genome was around $20 million, in 2008 it dropped to a million, and in 2012 to a mere $6000 (www.genome.gov/sequencingcosts). In 2014, the $1000 human genome milestone was reached thanks to Illumina's latest

*Corresponding author.

Next-Generation Sequencing (NGS) machine, the HiSeq X. The rate of this price drop is surpassing Moore's law, which suggests that efficient compression of this sequencing data will become increasing key for sustaining this growth.

NGS data can be classified into two main categories: (i) Raw NGS data — which is usually stored as a FASTQ file and contains the raw output of the sequencing machine and (ii) aligned raw NGS data — which besides the raw data it additionally contains its alignment to a reference. The latter is usually stored as a SAM/BAM file.[2]

We focus here on the aligned (reference-based) scenario, since this data is both the largest (up to terabytes) and the one generally used in downstream applications. Moreover, we believe that the size of these files conveys the major bottleneck for the transmission and handling of NGS data among researchers and institutions and, therefore, good compression algorithms are of primal need.

Although there exist general-purpose compression schemes like gzip, bzip2, and 7zip (www.gzip.org, www.bzip.org and www.7zip.org, respectively) that can be directly applied to SAM files, they do not exploit the particularities of this data, yielding relatively low compression gains.[3–5] For this reason, several algorithms for compression of SAM files have been previously proposed.

The first compression algorithm proposed for these files was **BAM**.[2] BAM was released at the same time as its uncompressed counterpart SAM. However, more than a specialized compression algorithm, it is a binarization of the SAM file that uses general-purpose compression schemes as its compression engine.

Concerned by the exponentially increasing size of the BAM files, Fritz *et al.*[6] proposed **CRAM**, a new compression scheme for aligned data. CRAM is presented as a toolkit, in the sense that it combines a reference-based compression of sequence data with a data format that is directly available for computational use in downstream applications.

In 2013, **Goby** was presented as an alternative to CRAM.[7] Goby is also presented as a toolkit that performs reference-based compression with a data format that allows downstream applications to work over the goby files. The authors also provide software to perform some common NGS tasks as differential expression analysis or DNA methylation analysis over the goby-compressed files.

Alternatively, more compression-oriented proposals have been published recently. These works do not focus on creating a compression scheme that aids downstream applications to work over the compressed data, but on maximizing the compression ratio.

In 2012, **Quip**, a lightweight dependency-free compressor of high-throughput sequencing data was proposed by Jones *et al.*[8] The main strength of Quip is that it accepts both non-aligned and aligned data, in contrast to both CRAM and GOBY.[a] In the case of non-aligned data, if a reference is available, it performs its own

[a] It is in fact the only algorithm appearing in all the categories when comparing compression algorithms by data type in the review paper of Bonfield and Mahoney.[5]

lightweight and fast assembly and then compresses the result. Since here we focus on aligned data, in the rest of the paper, we consider Quip exclusively on its SAM file compressor mode.

Finally, Bonfield and Mohoney[5] proposed **SamComp**, which is, to our knowledge, the best compression algorithm for aligned data. Two versions of the algorithm were proposed, namely, SamComp1 and SamComp2. Throughout the paper, we consider SamComp1, as it provides better compression results and it is the one recommended by the authors.[5] SamComp is more restrictive than Quip in the sense that it only accepts SAM and/or BAM files as input files. However, the same authors also proposed other methods that accept FASTQ files and a reference (optional) as input. If a reference is available they perform their own fast alignment before compression.[5] Note that the compression methods proposed in Ref. 5 do not compress the SAM file itself, but only the necessary information to be able to recover the corresponding FASTQ file.

The compression of NGS aligned raw data can be divided in several sub-problems of different nature; namely, the compression of the reads, the compression of the header and/or the identifiers, the compression of the quality scores, and the compression of the fields related to the alignment. Although all these sub-problems are addressed by the above algorithms,[b] in this paper, we focus exclusively on compressing the necessary information to reconstruct the reads. The reason being that they carry most — if not all — of the information used by downstream applications and thus their compression is of primal importance. Moreover, the compression of quality values can be drastically decreased by the use of lossy compression methods without compromising the performance of the downstream applications (see Refs. 9 and 10), while the identifiers and the headers are generally ignored.

In the context of compressing aligned reads, all the aforementioned algorithms, except SamComp, perform similarly in terms of compression ratio. Moreover, each of them outperforms the others in at least one data set (see the results tables of Refs. 5–8). On the other hand, SamComp clearly outperforms the rest of the algorithms, as shown in Bonfield and Mahoney.[5] It is important to mention that the comparison is not strictly fair as SamComp is a specialized SAM compressor whereas the other algorithms are oriented to provide a stable toolkit, which sometimes includes random access capabilities. However, SamComp shows that judicious design of models for the data yields a significant improvement in compression ratio and thus similar compression techniques should be applied in toolkit oriented programs such as Goby or CRAM.

Following the approach of SamComp, the main contribution of this paper is not to provide a complete compression scheme for aligned data, as done in CRAM or Goby, but to demonstrate the importance of data modeling when compressing aligned reads. That is, the purpose of the paper is to give insight into the potential gains that can be achieved by an improved data model. We believe that the

[b] To the exception of Bonfield and Mahoney[5] where they do not reconstruct a SAM file but a FASTQ file.

techniques described in this paper can be applied in the future to more complete compression toolkits. We show that, by constructing effective models adapted to the particularities of the genomic data, significant improvements in compression ratio are possible. We will show that these improvements become considerable in high coverage data sets.

Next, we introduce the lossless data compression problem, and examine the approach (and assumptions) made by the previously proposed algorithms when modeling the data.

1.1. *The lossless data compression problem*

Information theory states that given a sequence $\mathbf{s} = [s_1, s_2, \ldots, s_n]$ drawn from a probability distribution $P_{\mathbf{S}}(\mathbf{s})$, the minimum number of bits required on average to represent the sequences $\mathbf{s}$ is essentially (ignoring a negligible additive constant term) given by $H(\mathbf{S}) = \mathbb{E}[-\log(P_{\mathbf{S}}(\mathbf{S}))]$ bits,[c] where $H(\cdot)$ is the shannon entropy, which only depends on the probability distribution $P_{\mathbf{S}}$.

Thus, the lossless compression problem comprises two sub-problems: (i) data modeling, that is, selection of the probability distribution (or model) $P_{\mathbf{S}}$ of the sequence, and (ii) code word assignment, that is, given a $P_{\mathbf{S}}$ that models the data, find effective means of compressing close to the "ideal" $-\log(P_{\mathbf{S}}(\mathbf{s}))$ number of bits for any given sequence $\mathbf{s}$, such that the expected length of the compressed sequences can achieve the entropy (that is, the optimum compression). Arithmetic coding provides an effective mean to sequentially assign a codeword of length close to $-\log(P_{\mathbf{S}}(\mathbf{s}))$ to $\mathbf{s}$. There are other codes that achieve the "ideal code length" for particular models, such as Golomb codes for geometric distributions and Huffman codes for D-adic distributions.

The lossless compression problem can therefore equivalently be thought of as one of finding good models for the data to be compressed. The model for a sequence $\mathbf{s}$ can be decomposed as the product of the models for each individual symbol s_t, where at each instant t, the model for s_t is computed using the sequence of previous symbols $[s_{t-1}, \ldots, s_1]$ as context. This characteristic makes it possible to generate the models sequentially, which is particularly relevant for the aligned data compression problem, as the files can be of remarkable size, and more than one pass over the data could be prohibitive. Note that the memory needed to store all the different contexts grows exponentially with the context length and becomes prohibitively large very quickly. To solve this issue, the simple approach is to constrain the context lengths to at most m symbols, while more advanced techniques rely on variable context lengths.[11]

In general, a context within a sequence may refer to entities more general than substrings preceding a given symbol. In this paper, we show how the symbols appearing in the aligned file can be estimated using previously compressed symbols. This modeling of the data ultimately leads to considerable improvements over the state-of-the-art algorithms proposed so far in the literature.

[c] All the logarithms used in this paper are base-2.

1.2. *Data modeling in aligned data compressors*

Following a similar classification as in Campagne *et al.*,[7] we show different approaches for data modeling in the context of aligned data, and then introduce the ones employed by the above mentioned algorithms. Throughout the paper, we assume the SAM files are ordered by position, as this is also the assumption made by Goby, CRAM, and SamComp; moreover, as shown in Bonfield and Mahoney,[5] better compression ratios can be achieved.

The most general approach on data modeling is performed by general compression algorithms, which perform a serialization of the whole file and compute a byte-based model over the serialized file. However, since the SAM file is divided in different fields, the first intuitive approach is to treat each of the fields separately, as each of them is expected to be ruled by a different model. A further improvement can be done by considering cross-field correlation. In this case, the value of some fields can be modeled (or estimated) using the value of other fields, thus achieving better compression ratios. Finally, as shown in Ref. 7, other modeling techniques could be used, as for example, generating a template for a field and then compressing the difference between the actual value and the template.

In this context, CRAM encodes each field separately and extensively uses Golomb coding. By using these codes CRAM implicitly models the data as independent and identically distributed by a Geometric distribution. The main reason for using Golomb codes is their simplicity and speed. However, we believe that the computational overhead carried by the use of arithmetic codes is negligible, whereas the compression penalty paid by assuming geometrically distributed fields may be more significant.

Goby offers several compression modes. In the fastest one, general compression methods are used over the serialized data, yielding the worst compression ratios. In the slower modes a compression technique denoted by the authors as Arithmetic Coding and Template (ACT) is used. In this case, all the fields are converted to a list of integers and each list is encoded independently using arithmetic codes. To our knowledge, no context is used when compressing these lists. Moreover, we believe that the conversion to integer list, although it makes the algorithm more scalable and schema-independent, it damages the compression as it does not exploit possible models or correlations between neighboring integers. In the slowest mode, some inter-list modeling is performed over the integer lists to aid compression. The improvement upon CRAM is that no assumptions are made about the distribution of the fields. Instead, the code learns the distribution of the integers within each list as it compresses.

As a more data specific compressor, Quip uses a different arithmetic coding over each field. However, they do not use any context when compressing the data (thus, implicitly assuming independent and identically distributed data) and they treat each field independently.

SamComp, on the other hand, performs an extensive use of contexts; thus, they do not assume the independence of the data. The results of Ref. 5 show that the use of contexts aids compression significantly.

The aforementioned proposals — with the exception of SamComp — lack accurate data models for the reads and make extensive use of generic compression methods (e.g. byte-oriented compressors), relying on models that do not assume or exploit the possible dependencies across different fields.

In the present paper, we show the importance of data modeling when compressing aligned reads. We show that, by generating data models that are particularly designed for the aligned data, we can improve the compression ratios achieved by the previous algorithms. Moreover, we show that the pareto-optimal barrier for compression rate and speed claimed in Ref. 5 does not hold in general. This fact becomes more notable when compressing aligned reads from high coverage data sets.

Finally, as the proposed scheme is envisaged as being part of a more general toolkit, it is important to aim for a compression scheme that aids the future utilization of the compressed data in downstream applications, as proposed in Refs. 6 and 7. Thus, relevant information, such as the number of Single Nucleotide Polymorphisms (SNPs) and their position within the read, should be easily accessible from the compressed data. Quip and SamComp seem to lack this important feature as they perform, as part of the read compression, a base-pair by base-pair model, and compression. The main drawback of this approach is that in order to find variations of a specific read and the positions of the reference where the variations occur, one must first reconstruct the entire read to then extract the information. This can be computationally intensive for the downstream applications. In this paper, we show that the data can be compressed in a way that could enable downstream applications to rapidly extract relevant information from the compressed files, not only without compromising the compression ratio, but actually improving it.

2. Proposed Compression Scheme

As one of the aims of the proposed compression scheme is to facilitate downstream applications to work over the compressed data, we decompose the aligned information regarding each read into different lists, and then compress each of these lists using specific models. All operations — splitting the read into the different lists, computing the corresponding models and compressing the values in the lists — are done read by read, thus performing only one pass over the data and generating the compressed file as we read the data. Next we show the different lists in which the read information is split and introduce the models used to compress each of them.

For each read in the SAM file, we generate the following lists:

- **List $\mathcal{F}$**: We store a single bit that indicates the strand of the read.
- **List $\mathcal{P}$**: We store an integer that indicates the position where the read maps are in the reference.
- **List $\mathcal{M}$**: We store a single bit that indicates whether the read maps perfectly to the reference or not. We extract this information from both the CIGAR field and the

MD auxiliary field.[d] If the latter is not available, the corresponding mismatch information can be directly extracted by loading the reference and comparing the read to it.

- **List $\mathcal{S}$**: In case of a non-perfect match, if no *indels* (insertions and/or deletions) occur, we store the number of SNPs. We store a 0 otherwise.
- **List $\mathcal{I}$**: If at least one *indel* occurs we store the number of insertions, deletions, and substitutions (in the following we denote them as variations) occurring within the read.
- **List $\mathcal{V}$**: For each of the variations (note that each read may have multiple variations), we store an integer that indicates the position where they occur within the read.
- **List $\mathcal{C}$**: Finally, for each insertion and substitution, we store the corresponding new base pair, together with the one of the reference in case of a substitution.

The information contained in the above lists suffices to reconstruct the reads, assuming the reference is available for decompression.

Note that the amount of information we store per read depends on the quality of the mapping. For example,

- If a read maps perfectly to the reference we store just a 1 in $\mathcal{M}$.
- If it only has SNPs, we store a 0 in $\mathcal{M}$, the number of SNPs in $\mathcal{S}$ and their positions and base pairs in $\mathcal{V}$ and $\mathcal{C}$, respectively.
- If it does have insertions and/or deletions, we store a 0 in $\mathcal{M}$, a 0 in $\mathcal{S}$, the number of insertions, deletions, and SNPs in $\mathcal{I}$, and their respective positions and base pairs in $\mathcal{V}$ and $\mathcal{C}$, respectively.

It can be verified that this transformation of the data is information lossless and thus no compression capabilities are lost.

Next we describe the model used for each of these lists that will be fed to the arithmetic encoder. Recall that every list is compressed in a sequential manner, that is, at every step we compress the next symbol using the model computed from the previous symbols. The computation of the model for a symbol that has not appeared yet is the well known *zero frequency problem.* We use different solutions to this problem for each of the lists.

(1) **Modeling of $\mathcal{F}$**:

We use a binary model with no context, as empirical calculations suggest that the directions of the strands are almost independent and identically distributed.

(2) **Modeling of $\mathcal{P}$**:

The main challenge in compression of this list is that the alphabet of the data is unknown, very large and almost uniformly distributed. To address this challenge, since the positions are ordered, we first transform the original positions into gaps

[d] The CIGAR is a string that contains information regarding the mismatches between the read and the region of the reference where it maps to. The MD field is a string used to indicate the different SNPs that occur in the read.

between consecutive positions (delta encoding). This transformation considerably reduces the size of the alphabet and reshapes the distribution, strongly biasing it toward low numbers. This technique is also used in the previously proposed algorithms.

Regarding the unknown alphabet problem, the only information that is available beforehand is that the size of the alphabet is upper bounded by the length of the largest chromosome, which for example for humans is $3 \cdot 10.$[6] Therefore, a uniform initialization of the model to avoid the *zero probability problem* is prohibitive. Several solutions have been proposed to address this problem, as the use of Golomb Codes — which models the data with a Geometric distribution — or the use of byte-oriented arithmetic encoders which uses an independent model for each of the 4 bytes that forms the integer. However, since the distribution is not truly geometrical and a byte-oriented arithmetic encoder has a relevant overhead, we propose a different approach.

We use a modified order-0 Prediction by Partial Match (PPM)[11] scheme to model the data. At the beginning, the alphabet of the model is uniquely composed by an escape symbol e. When a new symbol appears, the encoder looks into the model and if it cannot find the symbol, it emits an escape symbol. Then the new symbol is stored in a separate list, and the model is updated to contain the new symbol. In this way, the encoder works nicely in both low coverage data sets, where lots of new symbols are expected; and in high coverage data sets, where the alphabet is reduced dramatically.

We also use this list to indicate changes of chromosomes and end of files.

(3) **Modeling of $\mathcal{M}$**:

To create the model we use the mapping position of the previous read (stored in $\mathcal{P}$) and its match value (stored in $\mathcal{M}$) as context. The rationale behind this approach is that we expect each region of the reference to be covered by several reads (specially for high coverage data), and thus if the previous read mapped perfectly, it is likely that the next read will also map perfectly if they start at close-by positions. The same intuition holds for the case of reads that do not map perfectly. Finally, the matches are compressed using an arithmetic encoder over a binary alphabet.

(4) **Compression of $\mathcal{S}$**:

To model this list, we use the previously processed number of SNPs seen per read to continuously update the model. However, due to the non-stationarity of the data, we update the statistics often enough such that the local behavior of the data is reflected in the model.

(5) **Modeling of $\mathcal{I}$**:

The modeling and compression of this list is done analogously to the previous list.

(6) **Modeling of $\mathcal{V}$**:

The positions of the variations is the less compressible list together with the mapping position list ($\mathcal{P}$), mainly due to the large number of elements it contains. In order to model it we use several contexts. First, we generate a global vector that indicates, for each position of the genome, if a variation has previously occurred.

Since we know in which region of the reference the read maps, we can use the information stored in the aforementioned vector to know which variations have previously occurred in that region, and use it as context to estimate the position of the next variation. We also use the position of the previous variation within the read (in case of no previous variations a 0 is used), and the direction of the strand of the read (stored in $\mathcal{F}$), as contexts. Finally, note that if the read is of length L and the previous variation has occurred at position $L-10$, the current variation must occur between $L-9$ and L. Therefore, the model updates its probabilities accordingly to assign a nonzero value only to these numbers.

The purpose of this model is to use the information of the previous variations to estimate where the current variations are going to occur. This works especially well for high coverage data, as one can expect several reads mapping to the same region of the reference, and thus having similar — if not the same — variations.

(7) **Compression of** $\mathcal{C}$:

Finally, for the compression of the base-pairs, we distinguish between insertions and SNPs. Moreover, within the SNPs, we use a different model depending on the base-pair of the reference that is to be modified. That is, given a SNP, we use the model associated with the base-pair in the reference. This choice of the model comes from the observation that the probability of having a SNP between non-conjugated base-pairs is higher than between conjugated base-pairs.

The models described above for the different lists use previously seen information (not restricted to the same list) to estimate the next values. These models rely on the assumption that the reads contain redundant information, a fact particularly apparent in high coverage data sets. We show in the simulation results that the compression ratio improvements with respect to the previously proposed algorithms increase with the coverage of the data, as one may have expected. This is an advantage of the proposed algorithm in light of the continued drop in the sequencing cost and the increased throughput of the NGS technologies which will boost the generation of high coverage data sets.

2.1. *Data*

In order to assess the performance of the proposed algorithm and compare it with the previously proposed ones, we consider the raw sequencing data shown in Table 1. To generate the aligned data (SAM files), we used the alignment program Bowtie2.[e,12] Detailed information regarding the parameters and the references used for Bowtie2 can be found in the *Supplementary Data*. For each of the data sets we specify the reference used to perform the alignment (which belongs to the same species as the data set under consideration), the number of reads that mapped to the reference after the alignment, the read length, the coverage, and the size of the SAM file. Recall that the SAM files contain only the reads that mapped to the reference. The references

[e] Note that any other alignment program could have been used for this purpose. We chose Bowtie2 because it was the one employed by Ref. [5] to perform the alignments.

hg19, *mm GRCm38*, and *ce ws235* belong to the *H. Sapiens*, the *M. Musculus*, and the *C. Elegans* species, respectively.

As shown in the table, we have divided the data sets into two ensembles, the low coverage data ensemble and the high coverage data ensemble.

The low coverage data ensemble is formed by human (*H. Sapiens*) and mouse (*M. Musculus*) sequencing data of coverage less than 1×. We chose these human data sets because they were used in the previously proposed algorithms to assess their performances. Although these data sets are easy to handle and fast to compress due to their small size (of the order of a few GB), we believe that they do not represent the majority of the sequencing data used by researchers and institutions. Furthermore, in order to analyze the performance of the different compression algorithms, it is important to consider data sets of different characteristics. With this in mind, in this paper, we also consider data sets of coverage up to 32×, which are presented next.

The high coverage data ensemble is composed by the sequencing data of two *H. Sapiens* and two *C. Elegans* (ERR and SRR, respectively). The data sets of the *C. Elegans* were also used in previous publications. However, note that the total size of these files is still in the order of few GB, as the size of the *C. Elegans* genome is significantly smaller than that of the *H. Sapiens*.

To the best of our knowledge, this is the first time that *H. Sapiens* sequencing data of coverage higher than 10× is used to assess the performance of SAM file compression algorithms. These files are of considerably larger sizes than the low coverage ones. For example, the size of the ERR262997_2 SAM file is 122 GB, whereas the one of the low coverage SRR027520_1 is 5.1 GB.

All the data sets have been retrieved from the European Nucleotide Archive (http://www.ebi.ac.uk/ena/).

2.2. *Machine specifications*

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

3. Results

Next we show the performance of the proposed compression method when applied to the data sets shown in Table 1, and compare it with the previously proposed algorithms. As mentioned above, we divide the results into two categories, namely, the low coverage data sets and the high coverage ones. In the following, we express the gain in compression ratio of an algorithm A with respect to another algorithm B as gain $= 1 - \text{size}(A)/\text{size}(B)$. For example, a reduction from 100 MB to 80 MB represents a 20% gain (improvement). Note that with this metric a 0% means the file size remains the same, a 100% improvement is not possible, as this will mean the new file is of size 0, and a negative value means that the new file is of bigger size.

Table 1. Data sets used for the assessment of the proposed algorithm.

Name	Reference	Mapped reads[a]	Read length	Coverage	Size [GB]
		Low coverage data sets			
SRR062634_1	hg19	23.3 M	100	0.26×	6.9
SRR027520_1	hg19	20.4 M	76	0.17×	5.1
SRR027520_2	hg19	20.4 M	76	0.17×	5.1
SRR032209	mm GRCm38	13.5 M	36	0.17×	2.7
SRR043366_2	hg19	13.5 M	76	0.11×	3.4
SRR013951_2	hg19	9.5 M	76	0.08×	2.4
SRR005729_1	hg19	9.4 M	76	0.08×	2.5
		High coverage data sets			
SRR065390_1	ce ws235	31.6 M	100	32×	9.5
SRR065390_2	ce ws235	31.2 M	100	32×	9.4
ERR262997_2	hg19	410.5 M	101	14×	122
ERR262996_1	hg19	272.8 M	101	10×	81

Note: The alignment program used to generate the SAM files is Bowtie2.[12] The data sets are divided in two ensembles, low coverage data sets, and high coverage data sets. The size corresponds to the SAM file.
[a]M stands for millions.

3.1. *Low coverage data sets*

We start by considering the low coverage data sets introduced in Table 1 to assess the performance of the proposed method. Figure 1 shows the compression ratio, in bits per base pair, of the different algorithms proposed in the literature. For

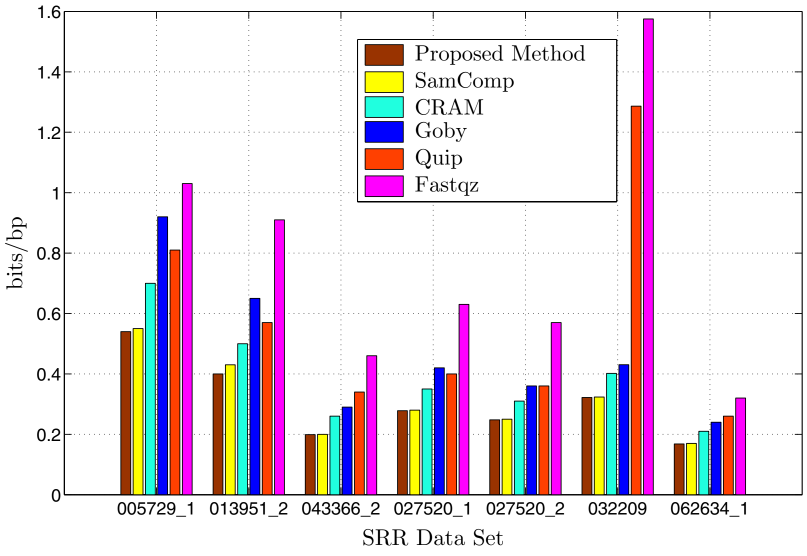

Fig. 1. Performance of the proposed method and the previously proposed algorithms when compressing aligned reads of low coverage data sets.

completeness, we also show the performance of Fastqz when it uses a reference.[5] However, note that the comparison is not strictly fair, as Fastqz performs its own fast alignment before compression, instead of using the alignment information provided in the SAM file. The data sets shown in the figure are ordered — from left to right — starting from the data set with the lowest coverage (SRR005729_1 with a coverage of 0.08×) and finishing with the largest (SRR0062634_1 with a coverage of 0.26×).

Note that the results shown in the figure refer to the compression of the reads. For Fastqz, Quip, and SamComp this value can be computed exactly, whereas for Goby and CRAM only an approximate value can be computed. We refer the reader to the *Supplementary Data* for further information, together with a description on the version and usage of each of the algorithms.

The results show that the compression ratio of Goby and Quip is similar, each of them outperforming the other in at least one data set, but worse than that of CRAM. SamComp, on the other hand, outperfoms the other algorithms in all the cases. Moreover, as shown in the figure, the proposed method outperforms all the previously proposed algorithms in all the data sets. The gain with respect to SamComp varies from 0.3% (SRR013951_2), to 8.6% (SRR013951_2). These gains become higher when the performance is compared with that of the other algorithms. We refer the reader to the *Supplementary Data* for exact numbers on the final size of the compressed files achieved by each of the algorithms. It is important to remark that these results show the compression ratio (shown in the figure as bits/bp) of the aligned reads, and it does not include the quality values and/or the identifiers.

Finally, we observe a somewhat expected relation between the compression ratio and the coverage of each data set (see Table 1): The higher the coverage, the more compressible the reads.

Regarding the running time, SamComp offers the best compression times, ranging from 45 to 160 s, followed by the proposed method, which takes between 124 and 163 s. Goby, CRAM, and Quip need more time for compression. However, note that they are compressing the whole SAM file. The slowest is Fastqz, mainly because it is performing its own alignment. Regarding the decompression time, all the algorithms except Fastqz employ similar times, ranging from 70 to 400 s. Exact values for the running times can be found in the *Supplementary Data.*

Note that since all the algorithms are sequential (that is, they compress and decompress read by read), the memory consumption of all of them is independent of the number of reads to compress and/or decompress. For example, for the largest human dataset of the low coverage ensemble, SRR062634_1, the algorithm with lowest memory consumption on the compression is the proposed method with 0.3 GB, followed by SamComp with 0.5 GB. Fastqz and Quip use around 1.3 GB, Goby 3.2 GB, and CRAM 4 GB. On the other hand, for decompression, SamComp is the one with the lowest memory consumption (300 MB), followed by

Table 2. Compression results for the high coverage ensemble.

		Raw size[a]	SamComp			Proposed method			Gain
Data set	Coverage	[MB]	Size [MB]	bits/bp	C.T. [sec]	Size [MB]	bits/bp	C.T. [sec]	(%)
ERR262996_1	10×	27,500	608.6	0.18	1210	548.7	0.16	1092	**10**
ERR262997_2	14×	41,460	935	0.18	1781	827.3	0.16	1585	**11.5**
SRR065390_1	32×	3160	47.7	0.12	148	40.5	0.10	104	**15**
SRR065390_2	32×	3110	55.2	0.14	140	45.9	0.12	108	**17**

Note: The results in bold show the compression gain obtained by the proposed method with respect to SamComp.
We use the International System of Units for the prefixes, that is, 1 MB stands for 10^6 Bytes.
C.T. stands for compression time.
[a]Raw size refers solely to the size of the mapped reads (1 Byte per base pair).

the proposed method with 500 MB. Fastqz uses 700 MB, Quip 1.5 GB, and Goby 4 GB.[f] Note that this comparison is not fully fair as the other algorithms compress and decompress the whole SAM file, while the proposed algorithm focuses in the reads.

3.2. *High coverage data sets*

For this ensemble, we set Fastqz, Quip, Goby, and CRAM aside, as they are outperformed by SamComp and the proposed method in terms of compression ratio, and focus exclusively on the latter two.

The performance of SamComp and the proposed method for the high coverage data sets introduced in Table 1 is summarized in Table 2. Similarly as for the low coverage data sets, the proposed method outperforms SamComp in all the cases. Moreover, we observe that the gain in compression ratio with respect to SamComp increases as the coverage of the data sets increase. For example, for the 10× coverage data (ERR262996_1) we obtain a compression gain of 10%, whereas for the 32× coverage data set (SRR065390_2), this gain is boosted up to 17%. Note that this gain translates into savings in the MB needed to store the files.

Regarding the running time, we observe that both algorithms employ similar time for compression, which is around 2 min for the *C. Elegans* data sets, and between 20 and 30 min for the *H. Sapiens* data sets. The decompression times are similar to the compression times. We refer the reader to the *Supplementary Data* for the exact values.

4. Discussion

Inspection of the empirical results of the previous section shows the superior performance of the proposed scheme across a wide range of data sets, from very low coverage to high coverage. Next we discuss these results in more detail.

[f]We do not specify the memory consumption of CRAM during the decompression because we were unable to run it.

4.1. ***Low coverage data sets***

As shown in Fig. 1, SamComp and the proposed method clearly outperform the previously proposed algorithms in all cases. However, we should emphasize that while the aim of SamComp and the proposed method is to achieve the maximum possible compression, the aim of CRAM, Goby, and Quip is to provide a more general and robust platform (or toolkit) for compression. Moreover, as mentioned before, the compression scheme of Goby and CRAM facilitates the manipulation of the compressed data by the downstream applications.

In this context, we believe that the compression method performed by SamComp and Quip does not allow downstream applications to rapidly extract important information, as they perform a base-by-base compression. Thus, in order to find variations in the data, one must first reconstruct the whole read to then find the variations. On the other hand, we proposed a compression method that can considerably facilitate the downstream applications when working in the compressed domain.

Regarding the compression ratio, Ref. 5 mentioned that they believed that a pareto-optimal region was achieved in terms of compression ratio after the SeqSqueeze (http://www.sequencesqueeze.org) competition of 2012, in which SamComp was the winner in the SAM file compression category. As shown in Fig. 1, although our algorithm performs strictly better than SamComp in all the data sets of this ensemble, the gain is very small. This fact could validate the pareto-optimal statement made in Ref. 5. However, as we discuss in the next section, for high coverage data sets the pareto-optimal curve is not achieved, as significant improvements in compression ratio are possible. We believe the reason is that, in low coverage data sets, little information can be inferred from previously seen reads when modeling the subsequent reads, as the overlap between reads is in general small or not existent. This, however, is far from the case in high coverage data sets.

4.2. ***High coverage data sets***

As outlined before, the proposed method demonstrates that significant improvements in compression ratio are possible in high coverage data sets. Specifically, we showed in Table 2 performance gains varying from 10% to 17% with respect to SamComp, which achieves the best compression ratio among the previously proposed algorithms.

These improvements in compression ratio are significant as, in the high coverage scenario, small gains in compression ratio can translate to huge savings in storage space. For example, a 10% improvement (e.g. from 0.18 bits/bp to 0.16 bits/bp) over human data with very large coverage (around $200\times$) corresponds to a saving of approximately 12 GB per file, with a corresponding reduction in the time required to transfer the data. Thus, in the compression of several large data sets of high coverage, such improvements would lead to several PetaBytes of storage savings.

Furthermore, as mentioned before, the proposed scheme generates compressed files from which important information can be easily extracted, potentially allowing downstream applications to work over the compressed data. This is an important feature, as with the increase in the size of the sequencing data, the burden of compressing and decompressing the files in order to manipulate and/or analyze specific parts in them becomes increasingly acute.

5. Conclusion

We are currently in the $1000 *genome era* and, as such, a significant increase in the sequencing data being generated is expected in the near future. These files are also expected to grow in size as the different NGS technologies improve. Therefore, there is a growing need for honing the capabilities of compressors for the aligned data. Further, compression that facilitates downstream applications working in the compressed domain is becoming of primal importance.

With this in mind, we developed a new compression method for aligned reads that outperforms, in compression ratio, the previously proposed algorithms. Specifically, we show that applying effective models to the aligned data can boost the compression, especially when high coverage data sets are considered. These gains in compression ratio would translate to huge savings in storage, thus also facilitating the transmission of genomic data across researchers and institutions. Furthermore, we compress the data in a manner that allows downstream applications to work on the compressed domain, as relevant information can easily be extracted from specific locations along the compressed files.

Finally, we envisage the methods shown in this paper to be useful in the construction of future compression programs that consider the compression not only of the aligned reads, but also the quality values and the identifiers.

Acknowledgements

The authors would like to thank Golan Yona for providing the initial motivation for this work and for helpful discussions. This work is partially funded by a Stanford Graduate Fellowships Program in Science and Engineering, a fellowship from the Basque Government, a grant from the Center for Science of Information (CSoI) and 1157849-1-QAZCC NSF grant.

Availability

The software is written in C and can be downloaded from http://www.stanford.edu/~iochoa/cbc.html.

References

1. Pennisi E, Will computers crash genomics? *Science* **331**:666–668, 2011.
2. Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, Marth G, Abecasis G, Durbin R, The sequence alignment/Map format and SAMtools, *Bioinformatics* **25**:16, 2009.

3. Zhu Z, Zhang Y, Ji Z, He S, Yang X, High-throughput DNA sequence data compression, *Briefings in Bioinformatics*, Advance Access, 2013.
4. Deorowicz S, Grabowski S, Data compression for sequencing data, *Algorithms Mole Biol* **8**, 2013.
5. Bonfield JK, Mahoney MV, Compression of FASTQ and SAM format sequencing data, *PLoS one*, **8**:e59190, 2013.
6. Fritz MH-Y, Leinonen R, Cochrane G, Birney E, Efficient storage of high throughput DNA sequencing data using reference-based compression, *Genome Res*, **21**:734–740, 2011.
7. Campagne F, Dorff KC, Chambwe N, Robinson JT, Mesirov JP, Compression of structured high-throughput sequencing data, *PLoS one*, **8**:e79871, 2013.
8. Jones DC, Ruzzo WL, Peng X, Katze MG, Compression of next-generation sequencing reads aided by highly efficient de novo assembly, *Nucleic Acids Res*, **40**:22, 2012.
9. Ochoa I, Asnani H, Bharadia D, Chowdhury M, Weissman T, Yona G, QualComp: A new lossy compressor for quality scores based on rate distortion theory, *BMC Bioinformatics*, **14**:187, 2013.
10. Canovas R, Moffat A, Turpin A, Lossy compression of quality scores in genomic data, *Bioinformatics*, Advance Access, 2014.
11. Sayood K, *Introduction to Data Compression*, 4th Edition Elsevier, 2012.
12. Langmead B, Salzberg S, Fast gapped-read alignment with Bowtie 2, *Nat Meth*, **9**:357–359, 2012.

Idoia Ochoa graduated with B.Sc. and MSc degrees in Telecommunication Engineering from the University of Navarra, Spain, in 2009. She obtained M.Sc. degree in Electrical Engineering from Stanford University in 2012. Currently she is a Ph.D. student at Stanford University, working with Professor Tsachy Weissman.

Her main interests include compression, information theory, and machine learning, with special emphasis on designed algorithms targeted to genomic data. Honors include being awarded with several scholarships, such as the Stanford Graduate Fellowship.

Mikel Hernaez graduated with a BSc and MSc in Telecommunication Engineering from the University of Navarra, Spain, in 2009. He earned Ph.D. from the same institution in 2012, where he became the lecturer of the Information Theory and Coding course. Currently he is a postdoctoral researcher at Stanford University, working with Professor Tsachy Weissman.

His research during his Ph.D. focused on communication theory and coding, and on information theory. Currently his main interests include information theory and compression, with emphasis on applicable schemes for genomic data, and the assessment of lossy compression on downstream applications.

Tsachy-Weissman graduated summa cum laude with a B.Sc. in Electrical Engineering from the Technion in 1997. He earned PhD from the same institution in 2001. He worked at Hewlett-Packard Laboratories with the Information Theory Group until 2003. He is a Professor of Electrical Engineering at Stanford University since 2003.

His research focuses on Information Theory and Communications, Statistical Signal Processing, the interplay between them, and their applications, with recent focus on applications in genomic data compression. He is an inventor of several patents and involved in a number of hi-tech and software companies as co-founder or member of the technical board. He is a technical consultant to HBO show "Silicon Valley". Honors include an NSF CAREER award, several best paper awards, fellowships for Leaders in Science and Technology, and for excellence in research. Incumbent of the STMicroelectronics Chair in the School of Engineering. IEEE fellow. He is on the editorial boards of the IEEE Transactions on Information Theory, and of Foundations and Trends in Communications and Information Theory.