HUGO: Hierarchical mUlti-reference Genome cOmpression for aligned reads

Pinghao Li,[1] Xiaoqian Jiang,[2] Shuang Wang,[2] Jihoon Kim,[2] Hongkai Xiong,[1] Lucila Ohno-Machado[2]

▶ Additional material is published online only. To view please visit the journal online (http://dx.doi.org/10.1136/amiajnl-2013-002147).

[1]EE Department, Shanghai Jiaotong University, Shanghai, China
[2]Division of Biomedical Informatics, University of California–San Diego, La Jolla, California, USA

Correspondence to
Pinghao Li, EE Department, Shanghai Jiaotong University, SEIEE Building 5-516, 800 Dongchuan RD, Minhang District, Shanghai 200240, China; pinghaoli1989@gmail.com

Received 1 July 2013
Revised 27 November 2013
Accepted 3 December 2013
Published Online First 24 December 2013

ABSTRACT

Background and objective Short-read sequencing is becoming the standard of practice for the study of structural variants associated with disease. However, with the growth of sequence data largely surpassing reasonable storage capability, the biomedical community is challenged with the management, transfer, archiving, and storage of sequence data.

Methods We developed Hierarchical mUlti-reference Genome cOmpression (HUGO), a novel compression algorithm for aligned reads in the sorted Sequence Alignment/Map (SAM) format. We first aligned short reads against a reference genome and stored exactly mapped reads for compression. For the inexact mapped or unmapped reads, we realigned them against different reference genomes using an adaptive scheme by gradually shortening the read length. Regarding the base quality value, we offer lossy and lossless compression mechanisms. The lossy compression mechanism for the base quality values uses k-means clustering, where a user can adjust the balance between decompression quality and compression rate. The lossless compression can be produced by setting k (the number of clusters) to the number of different quality values.

Results The proposed method produced a compression ratio in the range 0.5–0.65, which corresponds to 35–50% storage savings based on experimental datasets. The proposed approach achieved 15% more storage savings over CRAM and comparable compression ratio with Samcomp (CRAM and Samcomp are two of the state-of-the-art genome compression algorithms). The software is freely available at https://sourceforge.net/projects/hierachicaldnac/with a General Public License (GPL) license.

Limitation Our method requires having different reference genomes and prolongs the execution time for additional alignments.

Conclusions The proposed multi-reference-based compression algorithm for aligned reads outperforms existing single-reference based algorithms.

To cite: Li P, Jiang X, Wang S, *et al*. *J Am Med Inform Assoc* 2014;**21**: 363–373.

BACKGROUND AND SIGNIFICANCE

Massively parallel sequencing is leading revolutionary advances in biology and helping unravel the genetic basis of disease.[1] With the advent of next generation sequencing (NGS) technologies supported by companies such as Helicos Biosciences, Pacific Biosciences, and Illumina, the cost of sequencing the whole genome has decreased dramatically in the past few years and it is expected to drop below $1000 per individual in the near future.

However, the massive amount of data produced by whole genome sequencing is becoming a large burden for data storage and transmission. For example, the DNA of an individual is comprised of two complementary copies of more than 3.2 GB of bases. Today, the 1000 Genomes Project[2] has produced more than 50 TB of data for 1092 individuals from 14 populations, and is quickly moving toward the goal of sequencing 2500 people.[3] Many other genome data repositories are also expanding quickly. For example, the size of the Sequence Read Archive (SRA), a public repository of sequencing data, will exceed 1000 TB by the end of 2013.[4] Although the storage capacity keeps increasing, it cannot catch up with the speed of sequencing data generation. Thus, advanced genome compression mechanisms need to be developed to reduce the storage and reduce the bandwidth that is necessary for transferring genome data.

Sequence alignment is the first and essential step in genome analysis. Alignment tools like BLAST,[5] CUDASW++,[6] or Bowtie,[7 8 9] take advantage of the fact that the nucleotide diversity within the same species is relatively small—the difference in humans is around 0.1%, that is, 1 base difference per 1000.[10] Two formats, FASTQ and Sequence Alignment/Map (SAM) are becoming the industry standards for NGS data. FASTQ, which is a common input to alignment tools, is a text-based format for storing a biological sequence and its corresponding quality scores. The SAM file, which is an output from the alignment tool, follows a TAB-delimited text format consisting of an optional header section and an alignment section. Each alignment line has a variable number of optional fields or aligner-specific information, and 11 mandatory fields with essential alignment information, such as the mapping position. A SAM file is commonly compressed in the Blocked GNU Zip Format (BGZF) format and converted to a smaller binary BAM file,[11] which is supported by Illumina GA/HiSeq[12] and Roche 454.[13] In this study, our core compression algorithm focuses on the SAM format.

Related work

There are some analysis toolkits that deal with NGS data, such as Goby[14] and GATK[15 16] for alignment data. Sakib *et al*[17] presented a specific encoding scheme SAMZIP for SAM files by exploiting knowledge of the SAM format and its specifications. It processes each field independently using encoding techniques such as run-length encoding (RLE), delta encoding, Huffman coding, and dictionary coding. The SLIMGENE algorithm implemented by Kozanitis *et al* in collaboration with iDASH (a national center for biomedical computing)[18] can achieve 40× compression ratio (CR)

of genomic fragments without quality values and 5× CR when quality values are included. Hach *et al*[19] introduced another compression approach called SCALCE, for HTS (high throughput sequencing) genome (or transcriptome, exome, etc) sequences. HTS is based on the reorganization of reads to boost the locality of reference. The aforementioned algorithms compress genome data by exploring the redundant structure of the target, however they do not take full advantage of publicly available genome information.

Recently, Fritz *et al*[20] presented a reference-based compression method called CRAM. CRAM demonstrated a high CR by enabling lossy compression on the quality information and unaligned sequences. CRAM remaps unmapped reads (UMRs) using a secondary compression framework that remaps the unaligned reads again to other reference sequences.

Regarding lossless compression algorithms for FASTQ and SAM/BAM formats, Quip, which was proposed by Jones *et al*,[21] is one of the first assembly-based compressors using a de novo assembly algorithm. Later on, Popitsch *et al*[22] presented another set of lossless and lossy compressors called NGC. The NGC algorithm improves the compression efficiency by exploiting the redundancy within the common features of reads that are mapped to the same genomic positions, followed by a highly configurable strategy for quantizing per-base quality values. Samcomp, introduced by Bonfield *et al*,[23] uses the SAM flags, position, and cigar strings to anchor each called base to a reference coordinate and encodes the base according to a per-coordinate model. The accuracy and compression performances rely on the availability of aligned data at each specific reference coordinate.

The algorithms listed above either rely on a single reference genome or generate a de novo assembly that is entirely self-contained to align the target sequence for compression. However, such strategies usually result in a low rate of exact mapped reads (EMRs), because the compression of inexact mapped reads (IMRs) (ie, reads with four or fewer mismatches) and UMRs (ie, reads with more than five mismatches) usually requires more bits than the compression of EMRs. In contrast, we aim at improving compression efficiency by increasing the rate of EMRs by considering alternative references. Our strategy relies on sequentially mapping reads against multiple reference genomes as well as on an adaptive read length shortening mechanism.

Our proposed algorithm, Hierarchical mUlti-reference Genome cOmpression (HUGO), compresses a SAM format file sorted by position within the reference by extracting the 11 mandatory fields and a variable number of optional fields into 12 separate files. Because all these fields have low inter-correlations,[17] we can process each field in parallel. The workflow of the proposed algorithm is shown in figure 1. Since the 'Sequence' and the 'Quality value' fields represent more than 50% of the total data and are usually hard to compress (due to lack of regularity), we concentrate on the compression of these two difficult fields. For the 'Sequence' field, we align the reads against the reference sequence using SOAP3,[24] a Binary Alignment/Map, and Graphic Processing Unit (GPU)-based aligning software, where the alignment information for the EMRs is recorded for compression. SOAP3[24] can be 20× faster than CPU-based tools like BWA or Bowtie. Since HUGO is designed to conduct several rounds of alignments, speed is of critical concern. We realign the IMRs and UMRs to different reference sequences, which, combined to an adaptive read length shortening mechanism, turns these reads into EMRs. For the 'Quality' field, we further propose a lossy quantization approach using the k-means clustering algorithm and investigate its impact on downstream applications. For the remaining fields in the SAM file, we explore their self-regularity and interrelationship, and then adopt appropriate compression algorithms for each of them.

MATERIALS AND METHODS

Datasets

The datasets are taken from several types of NGS datasets. Sequences whose names started with 'NA' or 'HG' were taken from the 1000 Genomes Project. The remaining sequences were from the SRA under study numbers/run accession numbers: ChIP-Seq (mouse): SRX014899/SRR032209; RNA-Seq (*Escherichia coli*): ERX007969/ERR019653. Additionally, we used the HCC1954 genome from the University of California, Santa Cruz mixed spike-in low coverage sample for the Cancer Genome Atlas Benchmark 4, found at https://cghub.ucsc.edu/datasets/benchmark_download.html.

Methods

The SAM format has one header section with approximately 100 lines in total, and one alignment section. The header section can be identified by the prefix character '@'. Each alignment record in the alignment section consists of 11 mandatory fields and a variable number of optional fields. As mentioned earlier, since there are very low inter-field correlations,[17] we can independently process each field in parallel. We can thus reduce the time used for compression. Since this paper mainly focuses on the compression of 'Sequence' and 'Quality value' fields, a review of classic encoding schemes for the remaining fields in the SAM format is shown in online supplementary appendix 1.

Sequence field encoding design

Our method focuses on the efficient compression of genome sequences by seeking the best match among multiple reference genomes. The rationale for using a reference genome to improve compression performance is mainly based on the fact that the nucleotide diversity within human species is relatively small, so most reads in a re-sequencing run can find EMRs or IMRs within the reference sequences. Although a part of differences between the reads and the reference genome stems from sequencing errors instead of genetic variations, the ratios among EMRs, IMRs, and UMRs for a given set of input sequence usually vary considerably for different references that are used in the alignment, since different references usually reflect different sample characteristics, such as ethnicity of the sample donor. Figure 2 shows an example of the alignment results for NA12878chrom20[25] against four different reference sequences: hg18, hg19, HuRef chromosome 20, and HuRef (see online supplementary appendix 4 for details of these references).

Given the short reads from the 1000 Genomes sample NA12878,[25] the reference genome hg19 shows the highest rate of EMRs when compared to the other three references due to its completeness and integrity. However, there are still more than 25% reads partially mapped or unmapped. Existing genome compression methods handle IMRs and UMRs either (a) in their original format (eg, NGC) by storing UMRs in BAM format while quantizing their quality values in lossy compression modes, or (b) in a lossless manner (eg, SLIMGENE and CRAM) by introducing extra bits to record the mismatches. Original storage of IMRs and UMRs in BAM format usually achieves a low compression rate, and lossy quantization of quality values is not always favored by researchers who require lossless recovery of quality values for their downstream

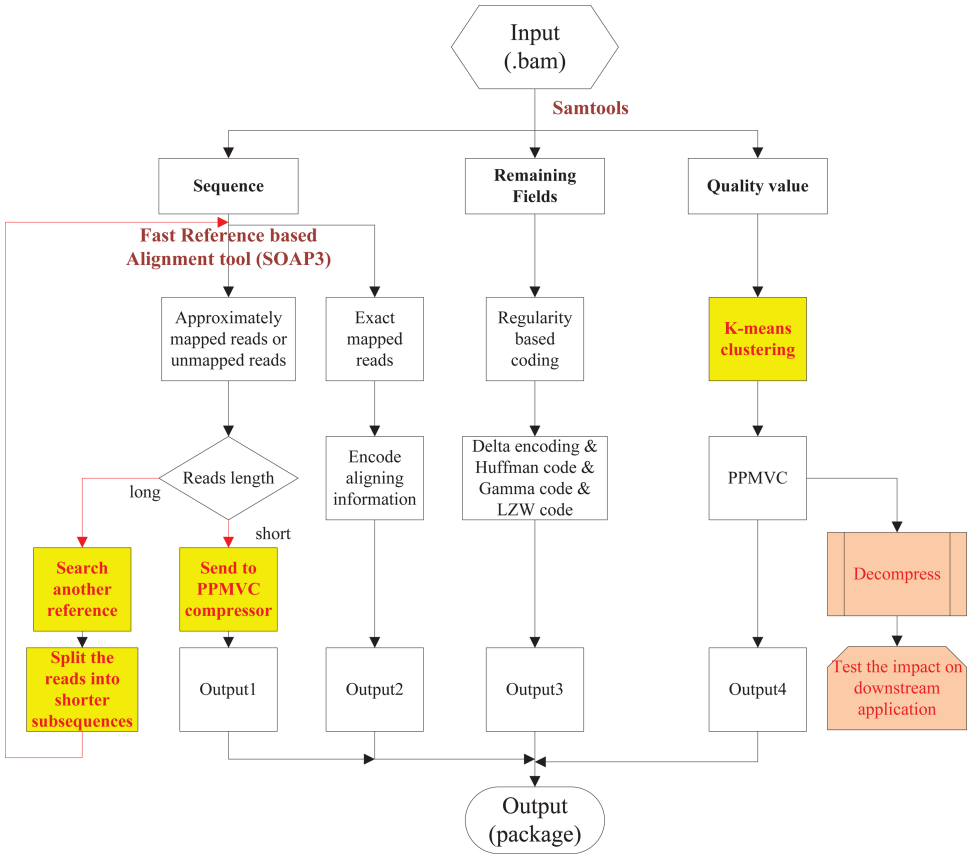

Figure 1 The framework for the proposed genome compression algorithm. The input file first goes through the exact matching scheme (leftmost branch) to iteratively identify matched sequences with reduced length. Unmatched sequences below a certain length threshold are compressed by PPMVC (command line parameter is 'e'), a variant of Prediction by a partial matching (PPM) algorithm[27] based on context modeling and prediction. The remaining fields, except for the quality value, are handled by regularity based coding (middle branch). We support two versions of quality value compression: (1) lossy version using k-means clustering and multi-thread PPMVC (command line parameter is 'e –o4 –r1'), (2) lossless version with multi-thread PPMVC (command line parameter is 'e –o4 –r1') only (rightmost branch).

applications. Although the existing lossless compression algorithms for IMRs and UMRs preserve the best data utility, they usually require a large amount of extra bits, and therefore they deteriorate the compression performance. For example, in order to encode IMRs/UMRs, the information about IMRs/UMRs' positions, strands (forward mapping or inverse mapping), substitutions, insertions, or deletions need to be recorded. In fact, such approach may not be the most efficient way to handle IMRs/UMRs, since a long IMR/UMR could be represented by multiple EMRs with shorter length in the same reference, which require fewer bits and usually result in better compression efficiency. Figure 2 illustrates that different references can provide different alignment coverage for a given set of input sequences. In other words, an IMR/UMR that cannot find an exact match in one reference might be perfectly aligned with another reference, or broken down into a smaller sequence that is an EMR.

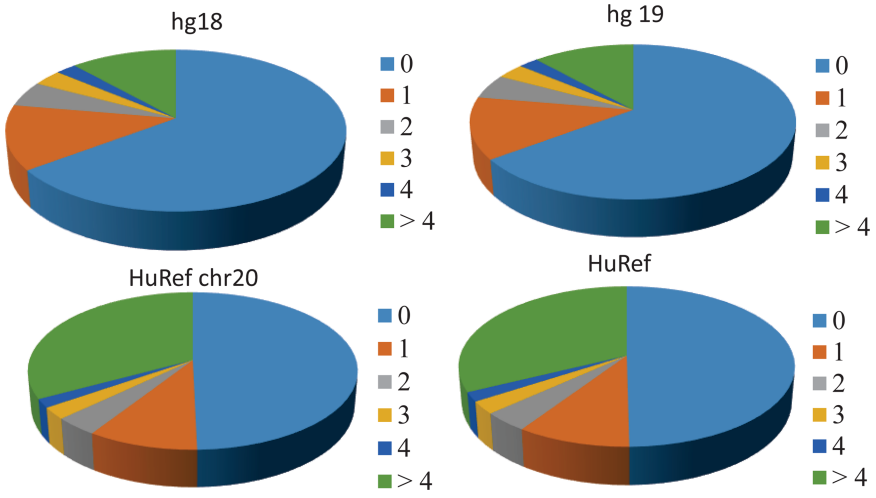

Figure 2 Alignment results of NA12878chrom20 against the four references. hg18 (upper-left), hg19 (upper-right), HuRef chromosome 20 (lower-left) and HuRef (lower-right). Each pie chart depicts the percentage of exact mapped reads (EMRs) with 0 mismatch, IMRs with 1 to 4 mismatches, and unmapped reads (UMRs) with more than 4 mismatches.

Based on the above observations, we propose a HUGO framework that incorporates mapping to alternative references as well as an adaptive read length shortening scheme to improve the compression performance for IMRs/UMRs.

Figure 3 shows an example of aligning the input file NA12878chrom20 against two references, hg19 and HuRef, using the read length shortening scheme, where IRMs or UMRs generated from hg19 are first shortened by half and realigned against hg19 or HuRef. Then, the HUGO encoder keeps reducing the lengths for all new generated IMRs/UMRs and realigning shorter reads until it reaches a predetermined threshold. We can see that the proportions of IMRs and UMRs decrease gradually as the read length gets shorter. In particular, the percentage of UMRs reduces to 0 when the read length is shortened to 19, as shown in figure 3.

Although figure 3 illustrates the advantages of multi-reference alignment and of a length shortening scheme for reducing the rate of IMRs and UMRs, it still shows each reference being considered independently. In figure 4, we explain how to build a hierarchical framework that can interactively select the best reference. In this case, we use HG00096chrom11 sequence[25] against two references, hg19 and HuRef. For example, in the second iteration, since the percentage of EMRs aligned by HuRef.fa (ie, 14.57%) is higher than that of hg19.fa (ie, 12.5%), the HUGO codec will pick HuRef.fa as the best reference for read length L = 50. Following the same procedure, figure 4 shows that a 99.99% EMRs rate, which is the sum over all the percentages in green, is achieved within four iterations. If two candidate references lead to the same EMRs rates, as shown in iteration 4 in figure 4, the reference that requires less information to represent EMRs will be selected. In practice, the EMRs rate in HUGO framework usually increases when sequences that are biologically more similar to that of the sample are utilized. In SOAP3, each EMR is represented by a very efficient data structure, a tetrad of {Read ID, Chrome Id, Offset, Strand}, where the encoding method for each field in the tetrad is presented as follows:

Read ID: This field refers to the read identifier of an EMR in original sequence. Since the read ID of an EMR is usually adjacent to another (refer to online supplementary figure S3 in appendix 3), we only need to calculate the differences of read IDs between adjacent EMRs through Delta coding. Next, Huffman coding is used to encode these differences, where the empirical statistic indicates that most differences are equal to or less than 2.

Chromosome ID: This field represents the chromosome identifier by which an EMR has been aligned. The vast majority of EMRs within the same sequence region are aligned to the same chromosome. Thus, run length encoding (RLE) is used to compress such identifiers. For example, an input data 'wwwwdddddd' can be encoded as 'w4d5' in RLE.

Offset: This field indicates the leftmost position that the exact alignment occurs in the designated chromosome of the

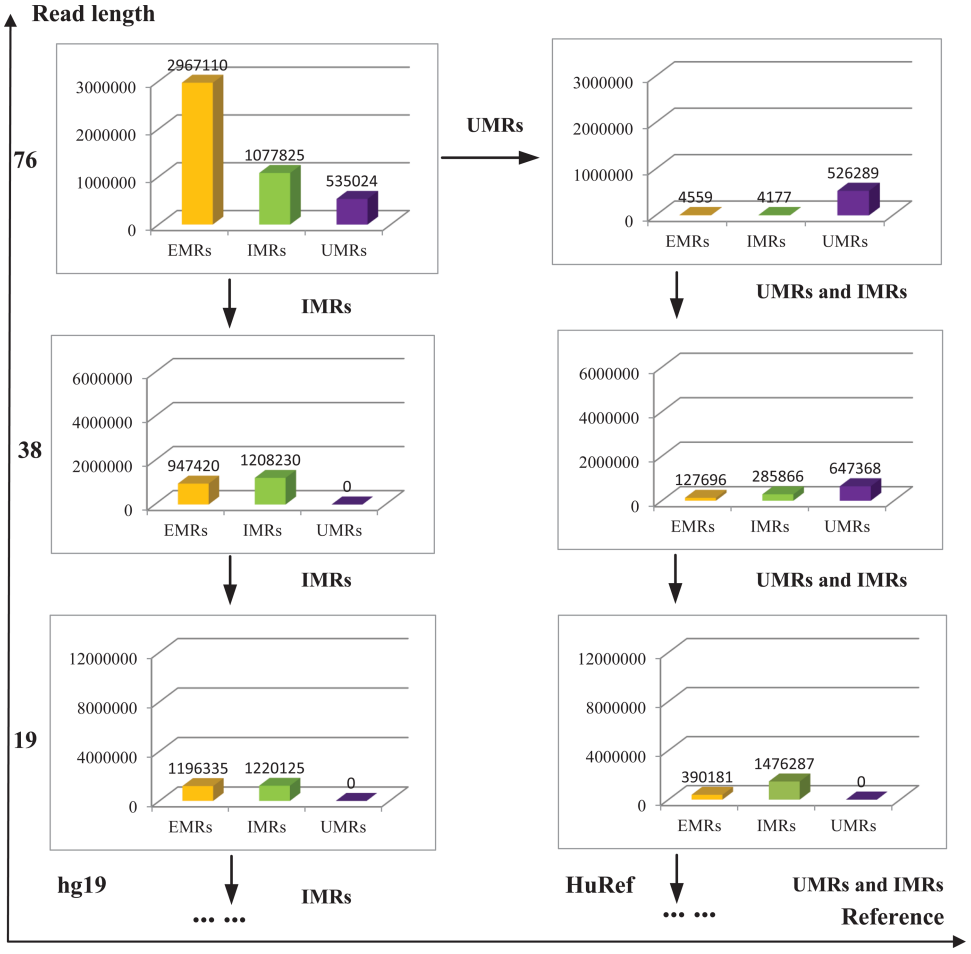

Figure 3 An example of aligning the input file NA12878chrom20 against two references, hg19 and HuRef. Exact mapped reads (EMRs)/IRMs and unmapped reads (UMRs) generated from hg19 are shortened and passed through the realignment module against hg19 and HuRef. Since each read length shortening step will double the number of IMRs/UMRs, we also double the y-axis limits to maintain an effective visual scale.

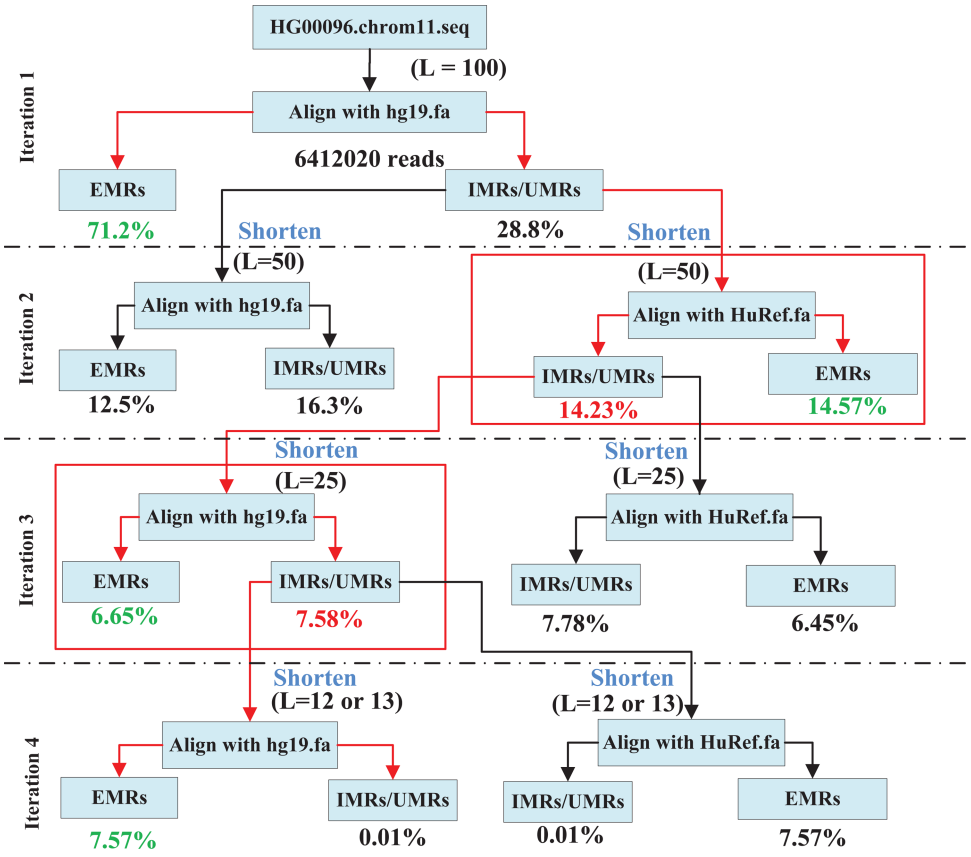

Figure 4 The procedure of aligning the input sequence HG00096chrom11 with references hg19 and HuRef. Here, a 99.99% exact mapped reads (EMRs) rate, which is the sum over all the percentages in green, is achieved in 4 iterations. The sequence length starts with L = 100 and the lengths of IMRs and unmapped reads (UMRs) are shortened by half after each iteration until L = 12 or 13. At each iteration, the reference, which leads to a lower IMRs/UMRs rate (as shown in red), will be selected as the current reference.

reference sequence. Our experiments indicate that the offset positions of adjacent EMRs are close to each other (see online supplementary figure S3 in appendix 3). Similarly to the Read ID case, we also apply Delta coding and Huffman coding to calculate and encode these differences.

Strand: This field consists of two identifiers, '+' and '–'. They represent the forward mapped read and reverse complementary mapped read, respectively. RLE is selected for its compression.

Since the current release of 1000 Genome data is generated from Illumina sequencers, where the read lengths usually range from 70 to 120 bp, the final read lengths after three read-length-shortening iterations will range from 17 to 30 bp. As observed in our experiments, further length splits with more than three iterations can only lead to a very little compression gain, but result in a high increase in time to compress. Thus, a maximum of three iterations will be used in our codec. For IMRs and UMRs generated in the last iterations, we encode them through the PPM (prediction by partial matching) algorithm.[26 27] The details of PPM can be found in online supplementary appendix 1.

It is worthwhile noting that Fritz *et al* have proposed to remap UMRs in CRAM using a secondary compression framework that may be built from multiple datasets, including an alternative human sequence. However, our HUGO approach differs from CRAM in three main aspects:

1. Unlike our hierarchical structure, CRAM only remaps the unaligned reads once to either contiguous sequences that are present across the first mapping or to another human sequence (eg, HuRef), and only later tries to map these reads to all bacterial and viral sequences to account for potential laboratory or sample contamination.
2. Given that it does not employ the length-shortening procedure from our approach, CRAM's remapping operation can not map as many unaligned reads as our proposed method.
3. CRAM has not implemented its multi-reference slices even in the latest V.2.0. The efficiency of this proposal special framework has not been tested. Hence, the multi-reference idea does not seem to have been put into practice yet.

Quality value field encoding design

The proposed strategy for the quantization of quality values seems related to the NGC's quantization 'binning' scheme. However, NGC determines the quantization borders by mapping all quality values that lie within an interval to a single value within this interval, and by allowing the application of different quantization schemas to quality values of different categories. In contrast, the proposed method automatically determines the quantization borders through a k-means clustering method to minimize the quantization error of all quality values by minimizing the within-cluster sum of squares (WCSS) of the errors.

For sequencing data from a Illumina sequencer, the original phred quality values (Q_{phred}) range from 0 to 93, where most of them are invisible ASCII characters. A shifted ASCII quality value Q_{ASCII} from 33 to 126 is introduced in equation (1), where P_e represents the error probability of the corresponding base-call. In practice, Q_{ASCII} has a wide value range and follows a quasi-random distribution, which results in high entropy, making

it hard to efficiently compress the Quality value.[17 21 28 29] Therefore, the best practice in most existing genome compression schemes[20 22 28] is to compress Q_{ASCII} in a lossy manner, which provides a trade-off between compression performance and data utility.

$$Q_{ASCII} = Q_{phred} + 33, P_e = 10^{-\frac{Q_{ASCII}-33}{10}},$$
$$Q_{phred} = -10\log_{10}P_e \quad (1)$$

However, the most important question for such a method is whether the information loss due to lossy compression on quality values will have a large impact on the results of downstream applications, such as software for variant calling,[16] genotype calling, and for removal of potential PCR duplicates,[30] which consider the quality values as an authoritative inference. To allow a flexible adjustment between the compression efficiency and the data utility for downstream applications, we present a user-configurable quantization scheme for the lossy compression of Q-value through k-means clustering. Assume an individual has the total $L_r = \sum_{t=1}^{W} m_t$ quality values, where a read t has length m_t and the total number of reads is W. The whole L_r quality values are reduced to n unique ones Q_{ASCII} by $\{q_1, q_2, q_j, \ldots q_n\}$, where each q_j appears num_j times in W reads. The k-means clustering aims at partitioning L_r quality values into k clusters, so that the quality values within the same cluster can be replaced by the quality values of the cluster center.[31] We can minimize the WCSS in terms of the error probability:

$$\arg\min_{S} \sum_{i=1}^{k} \sum_{q_j \in S_i} \left\| 10^{-\frac{q_j-33}{10}} - 10^{-\frac{u_i-33}{10}} \right\|^2 \cdot num_j \quad (2)$$

Where u_i is the cluster mean of points in S_i and the exponential terms come from the definition in equation (1). Algorithm 1 in box 1 shows the proposed k-means clustering scheme for Q-value quantization and compression.

Based on algorithm 1, the quantization result after k-means clustering is shown in figure 5, where k = 10 and the input file is NA12878chrom20. In figure 5, the value of each cluster center is shown in red and q_j's within the same cluster are depicted in the same color. Online supplementary table S1 in appendix 3 illustrates the trade-offs between the compression performance using bzip2 and the quantization error in terms of mean absolute percentage error (MAPE) after k-means clustering for quality values. Although it decreases the storage by more than 50% with k = 5, the MAPE increases to 27.49%. This result indicates that the proposed quantization method might generate too much noise for small k's. A more important efficacy measurement is the impact of the lossy scheme to the downstream applications, especially variant calling. We tested such impact in the following section. Finally, it is worth mentioning that the proposed algorithm is equivalent to a lossless compression scheme without quantization when the k value is set to the number of unique quality values, which is 51 in this case.

The influence of quantizing per-base quality values on downstream analysis

We used a popular variant calling tool, SAMTOOLS, for testing the variant and genotype prediction. The resulting variant sets from the file that was decompressed from the lossy compressed BAM file were then compared with the ones obtained from uncompressed datasets that served as our 'gold standard'. If the variant sets obtained from these two inputs show very similar results, we can declare that the lossy quantization of quality values has little influence in the downstream analysis. We counted the number of recovered (ie, tp: true positive), lost (ie, fn: false negative), and additional (ie, fp: false positive) variants, and recovered invariants (ie, tn: true negative). Then, we defined the variant recovery precision as $\frac{tp}{tp+fp}$, the sensitivity (a.k.a. recall rate) as $\frac{tp}{tp+fn}$, the genotype preservation percentage as $1-\frac{cgt}{tp}$, and the specificity as $\frac{tn}{tn+fp}$ following the specifications,[22] where cgt is the number of variants from the set of true positives that changed their genotype classification from homozygous to heterozygous or vice versa.

The variant recovery precision and sensitivity analysis reveal how much k-means clustering quantization has to sacrifice in terms of false positive and false negative calls. The genotype preservation percentage indicates how well each quantization mode preserves the predicted genotype of called variants. The variant recovery specificity measures the proportion of negatives (ie, invariants) which are correctly identified. In figure 6, we plotted the percentage of these four statistical measures for NA12878chrom20 and HG00096chrom11, respectively. The specific statistics of these four statistical measures corresponding to figure 6 are included in supplementary table S2 in appendix 3. Figure 6 shows that, as expected, with lower k's (ie, higher CRs) one can observe more fp + fn as well as less tp + tn with incorrectly predicted genotypes. However, it does not mean that tp (or tn) with lower k is smaller or fp (or fn) with lower k is larger than that with higher k, which can be also inferred from online supplementary table S2. That is why HUGO_L with six clusters (ie, HUGO_L(6)) performs worse than the situation with one and two clusters in terms of variant

Box 1 The algorithm description of the proposed k-means clustering scheme for Q-values' quantization and compression.

Algorithm 1: Proposed k-means clustering scheme for Q-values' quantization and compression

1. Initiate the set of k means $U^{(1)} = \{u_1^{(1)}, u_2^{(1)}, \ldots, u_k^{(1)}\}$.
2. *while* the set of k cluster center $U^{(t)} \neq U^{(t-1)}$ *do*
3. Assignment step: Assign each observation to the cluster whose mean is closest.
4. $$S_i^{(t)} = \{q_p : \left\| 10^{-\frac{q_p-33}{10}} - 10^{-\frac{u_i^{(t)}-33}{10}} \right\| \leq \left\| 10^{-\frac{q_p-33}{10}} - 10^{-\frac{u_j^{(t)}-33}{10}} \right\|, \forall 1 \leq j \leq k\}$$
5. Update center step: Calculate the new means to be the centroids of the observations in the new clusters.
6. $$u_i^{(t+1)} = -10 \cdot \log\left(\frac{1}{|S_i^{(t)}|} \sum_{q_j \in S_i^{(t)}} (10^{-\frac{q_j-33}{10}} \cdot num_j)\right) + 33$$
7. *end while*
8. Calculate the mean absolute percentage error (ie, MAPE):
9. $$MAPE = \frac{100}{\sum_{q_i} num_i} \cdot \sum_{i=1}^{k} \sum_{q_j \in S_i} \left| \frac{10^{-\frac{q_j-33}{10}} - 10^{-\frac{u_i-33}{10}}}{10^{-\frac{q_j-33}{10}}} \right|$$

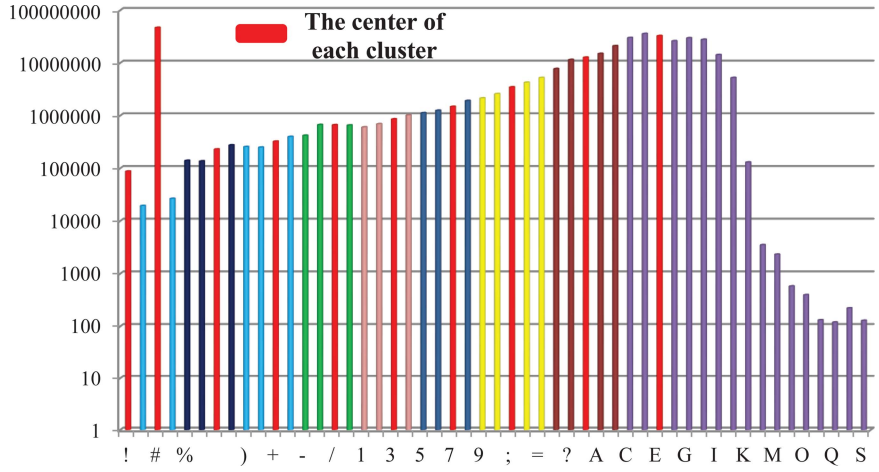

Figure 5 The distribution of k-means clustering results with k=10. The vertical axis shows the occurring times for each quality value in the log scale. Since n is 51 in this sample, there is a corresponding number of unique quality values in the original 'Quality value' field, and we quantize these integers into 10 clusters (red bars) using the k-means clustering scheme. Quality values in the same cluster are depicted by the same color.

recovery precision and variant recovery specificity. In any case, in figure 6, when $k \geq 10$ the proposed algorithm results in reasonable accuracy (ie, the accuracy rates of four statistical measures are all above 95%). HG00096chrom11 generates more accurate results than NA12878chrom20 using the same lossy mode k. Moreover, the proposed lossy quality values quantization schemes outperform the lossy mode of CRAM (ie, CRAM-l) with higher variant recovery, higher variant recovery specificity, and higher genotype preservation rates. However, HUGO_L(1) and HUGO_L(2) show lower variant recovery sensitivity (ie, more false negatives) than CRAM-l. Clearly, there is not a single winner but our experiment empirically suggests that we can reduce storage without sacrificing much data utility by selecting the appropriate cluster number k.

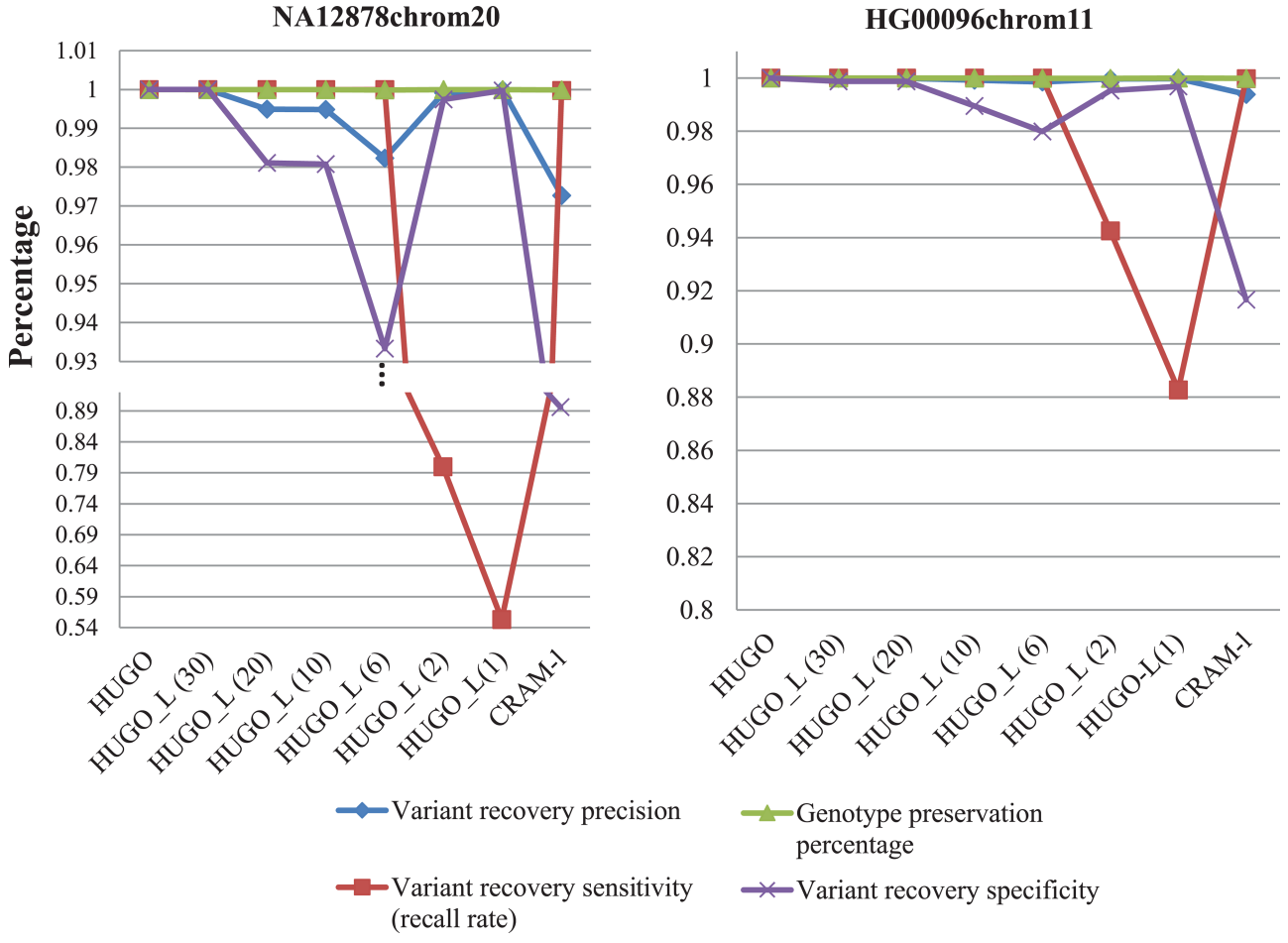

Figure 6 The impact of lossy compression on variant callings based on (left) NA12878chrom20 and (right) HG00096chrom11 sequences using SAMTOOLS. Here 'Hierarchical mUlti-reference Genome cOmpression (HUGO)' refers to the lossless compression of quality values. 'HUGO_L' refers to the lossy compression of quality values using the k-means algorithm, and the number in the bracket indicates the value of k. 'CRAM-l' is the lossy mode of CRAM with the lossy compression command line '--capture-insertion-quality-scores --capture-piled-quality-scores --capture-substitution-quality-scores --capture-unmapped-quality-scores --capture-all-tags'.

Encoding techniques for fields other than 'Sequence' and 'Quality value' fields

The distribution regularity of each field and corresponding compression methods are described in online supplementary appendix 2. We summarize the specific compression scheme for each field in table 1. We tested the compression performance for the corresponding 10 fields in NA12878chrom20 and HG00096chrom11 sequences. The original size of each individual SAM field is determined by the file size of the uncompressed field extracted from the corresponding SAM file. As shown in table 1, the compressed files are much smaller when compared to the original ones.

At decompression, we first decompress and transform the 12 fields that consist in principle of the 11 mandatory and all optional SAM fields independently. Then, we reconstruct the SAM file from the 12 separate decompressed files, each of which contains information on one field, except that the twelfth file contains the information of all optional fields. For the 'Sequence' field, because the IMRs/UMRs in each alignment are sent to the next alignment during the compression, we have to recover reads from the last alignment to the first alignment during the decompression and therefore decompress the IMRs/UMRs at each alignment. Consequently, the decompressed reads remain the same as the original ones. Since each field's content has the same order, we concatenate the 12 fragments from the 12 files in a record-by-record way in SAM's standard format (ie, 'QNAME', 'FLAG', 'RNAME', 'POS', 'MAPQ', 'CIGAR', 'MRNM', 'MPOS', 'TLEN', 'SEQ', 'QUAL', 'OPT'). The names of candidate references are recorded directly from the users' command line settings. If needed, we convert the generated SAM file to the BAM format using SAMTOOLS.

RESULTS

We implemented both our encoder and decoder in C++ and ran experiments on a Linux workstation with a 2.4 GHz Intel Xeon CPU, 96 GB of memory, and an NVIDIA Tesla M2090 GPU. In addition, we tested the performances of the proposed HUGO codec with different setups over the datasets taken from several types of NGS data; the results were also compared with several existing compression algorithms, such as bzip2, CRAM/CRAM-l, and Samcomp. The latest CRAM V.2.0 and Samcomp V.0.7 were used in the comparison. The command line parameters used for bzip2, CRAM, CRAM-l, and Samcomp were, respectively, '-z', '--capture-all-quality-scores –capture-all-tags', '–capture-insertion-quality-scores –capture-piled-quality-scores –capture-substitution-quality-scores –capture-unmapped-quality-scores –capture-all-tags' and '<input.sam>'.

The results of our evaluation are summarized in table 2, which includes the compressed size, CR, and compression/decompression time (CT/DT) for different settings. For all input sequences, bzip2 showed the worst compression performance, as it was designed for fast compression of general-purpose data. The failure of general-purpose compression algorithms is one of the crucial reasons why biomedical researchers pursue alternatives for efficient storage of large genome data. Due to the use of per-position models, Samcomp seems to provide the best compression efficiency over all input sequences. However, it is important to note some limitations of Samcomp, which is not a full-field SAM/BAM compressor because it primarily focuses on identifiers, sequence, and quality values. The SAM header, auxiliary fields, and the template fields in columns 7–9 are not preserved. For a fair comparison with Samcomp, we present the results of 'HUGO*' in table 2 when auxiliary fields and columns 7–9 are removed. Since the proposed method HUGO and the CRAM tool only store the number of each query name instead of the full name, the decompressed sequences of HUGO/CRAM act as the input sequences for Samcomp. Under these equivalent conditions, we can see that HUGO* achieves similar CRs as Samcomp, although its CPU costs (time and memory) are higher than those of Samcomp. In this study, we focus on HUGO's adaptive length and highly tuned encoding and its compression performance. Table 2 illustrates that our tool achieved lossless CRs between 0.5 and 0.65, which corresponds to space savings of 35–50%. For the lossy compression modes, the proportion of space savings increases up to 50–90% by quantizing the quality values with k-means clustering. HUGO also saves 6–20% space compared to the CRAM tool for the same lossless or lossy compression mode.

Although the proposed HUGO scheme achieves better compression efficiency, the compression time is increased by 20–40% when compared with CRAM. Additionally, the maximum memory usage of compression doubles over the experimental sets, as shown in table 3. It is worth mentioning that the memory consumption in 'HUGO' during compression is mainly due to the fast alignment tool SOAP3. In contrast, the memory usage for decompression in HUGO is much less than CRAM. In addition, the execution time can be considerably reduced when the SAM format is used as an input, since the conversion between SAM and BAM formats using SAMTOOLS introduces a large compression overhead.

LIMITATIONS AND DISCUSSION

Even though EMRs can be efficiently stored, IMRs and UMRs, which account for 20–50% of total reads in a sequence, are not as well compressed by traditional methods, and may dominate the final storage size. The key idea of our proposed method is to use a tunable multi-reference based scheme for different resolution of reads. However, in several instances there was no

Table 1 Compression results for the corresponding 10 fields in NA12878chrom20 and HG00096chrom11 sequences

Field	QNAME	FLAG	RNAME	POS	MAPQ	CIGAR	MRNM	MPOS	TLEN	OPT	Total
Compression scheme	HC	HC	RLE	DC/HC	HC	LZW	RLE	DC/HC	HC	bzip2	–
NA12878 chrom20											
Original size (MB)	87.1	16.6	13.7	40.4	13.6	22.8	9.2	40.3	19.7	696.0	659.4
Compressed size	2.2	1.9	0.1	3.2	0.9	2.7	0.1	5.9	1.7	11.9	30.6
HG00096 chrom11											
Original size (MB)	113.0	21.5	18.3	56.2	18.2	33.0	12.2	56.0	27.3	1127.0	1482.7
Compressed size	2.2	2.0	0.1	5.5	1.2	2.8	0.2	7.7	2.0	21.9	45.6

DC, Delta coding; HC, Huffman coding; LZW, Lempel-Ziv-Welch coding; RLE, run-length encoding.

Table 2 Comparisons of compression efficiency among the proposed HUGO/HUGO_L (lossy compression using k clusters), bzip, CRAM, and Samcomp for a broad types of sequences

Input sequence	Program name	Reference	BAM size	CS	CR	CT	DT
NA12878 chrom20	bzip2	hg19	356 MB	358 MB	–	2 min 5 s	1 min 32 s
	Samcomp	hg19		163 MB	0.46	50 s	49 s
	HUGO*	hg19		169 MB	0.47	2 min 38 s	2 min 00 s
	CRAM	hg19		217 MB	0.61	2 min 30 s	2 min 25 s
	HUGO	hg19		189 MB	0.53	3 min 50 s	2 min 55 s
	HUGO	hg19,HuRef		189 MB	0.53	3 min 49 s	2 min 57 s
	HUGO_L(30)	hg19		187 MB	0.53	4 min 00 s	2 min 58 s
	HUGO_L(20)	hg19		143 MB	0.40	3 min 55 s	2 min 55 s
	HUGO_L(10)	hg19		98 MB	0.28	3 min 50 s	2 min 39 s
	HUGO_L(02)	hg19		56 MB	0.16	3 min 41 s	1 min 40 s
	HUGO_L(01)	hg19		53 MB	0.15	3 min 40 s	1 min 40 s
	CRAM-l	hg19		54 MB	0.15	2 min 20 s	2 min 20 s
HG00096 chrom11	bzip2	hg19	661 MB	663 MB	–	3 min 15 s	1 min 30 s
	Samcomp	hg19		304 MB	0.46	1 min 30 s	1 min 44 s
	HUGO*	hg19, HuRef		315 MB	0.48	3 min 20 s	2 min 38 s
	CRAM	hg19		411 MB	0.62	4 min 00 s	4 min 30 s
	HUGO	hg19		348 MB	0.53	5 min 01 s	4 min 20 s
	HUGO	hg19, HuRef		346 MB	0.52	5 min 00 s	4 min 15 s
	HUGO_L (30)	hg19		343 MB	0.52	5 min 10 s	4 min 14 s
	HUGO_L (20)	hg19		271 MB	0.41	5 min 05 s	4 min 09 s
	HUGO_L (10)	hg19		151 MB	0.23	4 min 58 s	4 min 02 s
	HUGO_L(02)	hg19		75 MB	0.11	4 min 55 s	2 min 38 s
	HUGO_L (01)	hg19, HuRef		72 MB	0.11	4 min 56 s	2 min 30 s
	CRAM-l	hg19		71 MB	0.11	3 min 55 s	4 min 31 s
HG00103 chrom11	bzip2	hg19	717 MB	720 MB	–	3 min 34 s	1 min 30 s
	Samcomp	hg19		333 MB	0.46	1 min 36 s	1 min 55 s
	HUGO*	hg19, HuRef		344 MB	0.48	3 min 50 s	2 min 49 s
	CRAM	hg19		454 MB	0.63	5 min 30 s	5 min 15 s
	HUGO	hg19		382 MB	0.53	5 min 33 s	4 min 40 s
	HUGO	hg19, HuRef		380 MB	0.53	5 min 32 s	4 min 38 s
HG01028 chrom11	bzip2	hg19	964 MB	967 MB	–	4 min 45 s	2 min 10 s
	Samcomp	hg19		458 MB	0.48	2 min 06 s	2 min 39 s
	HUGO*	hg19, HuRef		476 MB	0.49	4 min 15 s	2 min 58 s
	CRAM	hg19		585 MB	0.61	5 min 16 s	7 min 00 s
	HUGO	hg19		532 MB	0.55	7 min 38 s	5 min 20 s
	HUGO	hg19, HuRef		529 MB	0.55	7 min 38 s	5 min 22 s
NA06984 chrom11	bzip2	hg19	1.19 GB	1.192 GB	–	6 min 08 s	2 min 32 s
	Samcomp	hg19		504 MB	0.41	2 min 32 s	3 min 04 s
	HUGO*	hg19, HuRef		542 MB	0.44	4 min 0 s	3 min 42 s
	CRAM	hg19		737 MB	0.60	6 min 43 s	9 min 00 s
	HUGO	hg19		634 MB	0.52	9 min 25 s	8 min 55 s
	HUGO	hg19, HuRef		630 MB	0.52	9 min 20 s	8 min 50 s
NA06985 chrom11	bzip2	hg19	2.33 GB	2.34 GB	–	11 min 35 s	5 min 10 s
	Samcomp	hg19		1188 MB	0.50	5 min 42 s	6 min 21 s
	HUGO*	hg19, HuRef		1295 MB	0.54	8 min 34 s	7 min 14 s
	CRAM	hg19		1570 MB	0.66	12 min 40 s	15 min 20 s
	HUGO	hg19		1456 MB	0.61	17 min 18 s	12 min 43 s
	HUGO	hg19, HuRef		1451 MB	0.61	17 min 15 s	12 min 40 s
HG00096 mapped	Samcomp	hg19	14.53 GB	6936 MB	0.47	33 min 40 s	38 min 20 s
	HUGO*	hg19, HuRef		7165 MB	0.48	1 h 20 min	1 h 20 min
	CRAM	hg19		n/a	n/a	n/a	n/a
	HUGO	hg19		7965 MB	0.54	2 h 30 min	2 h 20 min
	HUGO	hg19, HuRef		7919 MB	0.53	2 h 30 min	2 h 20 min
RNA-Seq ERR019653	Samcomp	NC_000913	169 MB	79 MB	0.47	30 s	52 s
	HUGO*	NC_000913		96 MB	0.57	1 min 8 s	1 min 45 s
	CRAM	NC_000913		102 MB	0.60	1 min 30 s	1 min 32 s
	HUGO	NC_000913		101 MB	0.60	2 min 35 s	2 min 30 s
ChIP-Seq SRR032209	Samcomp	mouse	608 MB	270 MB	0.44	1 min 25 s	1 min 30 s
	HUGO*	mouse		275 MB	0.45	3 min 01 s	3 min 15 s
	CRAM	mouse		289 MB	0.48	4 min 40 s	4 min 17 s
	HUGO	mouse		299 MB	0.49	4 min 42 s	6 min 10 s
Cancer HCC1954	Samcomp	hg19	29.7 GB	8041 MB	0.27	50 min 27 s	53 min 38 s
	HUGO*	hg19		8313 MB	0.27	2 h 5 min	2 h 2 min
	CRAM	hg19		19 586 MB	0.64	2 h 1 min	3 h 8 min
	HUGO	hg19		16 759 MB	0.55	4 h 10 min	4 h 8 min

Here, 'n/a' indicates that an Exception error occurred because some reference sequences were not found in the fasta file when compressing the input file 'HG00096mapped' using CRAM.
CR, compression ratio; CS, compressed size; CT, compression time; DT, decompression time; HUGO, Hierarchical mUlti-reference Genome cOmpression.

Table 3 Comparison of maximum memory usage among the proposed HUGO method, CRAM, and Samcomp, for different types of sequences

Input sequence	Program name	BAM size	SAM size	Memory usage	
				Compression	Decompression
NA12878chrom20	CRAM	356 MB	1.58 GB	6.7 GB	6.5 GB
	Samcomp			292 MB	320 MB
	HUGO			14 GB	1.9 GB
HG00096chrom11	CRAM	661 MB	2.65 GB	8.1 GB	4.3 GB
	Samcomp			300 MB	320 MB
	HUGO			15 GB	2.4 GB
HG00103chrom11	CRAM	717 MB	2.91 GB	8.0 GB	4.6 GB
	Samcomp			301 MB	320 MB
	HUGO			16 GB	2.6 GB
HG01028chrom11	CRAM	964 MB	3.95 GB	8 GB	3.7 GB
	Samcomp			301 MB	320 MB
	HUGO			16 GB	3.2 GB
NA06984chrom11	CRAM	1.19 GB	5.16 GB	8.2 GB	4.1 GB
	Samcomp			300 MB	320 MB
	HUGO			19 GB	3.5 GB
NA06985chrom11	CRAM	2.33 GB	9.41 GB	8.1 GB	4.3 GB
	Samcomp			300 MB	320 MB
	HUGO			19 GB	4.0 GB
HG00096mapped	CRAM	14.53 GB	60.1 GB	n/a	n/a
	Samcomp			314 MB	320 MB
	HUGO			45 GB	34 GB
RNA-Seq ERR019653	CRAM	169 MB	1.32 GB	6.2 GB	3.3 GB
	Samcomp			285 MB	290 MB
	HUGO			3.2 GB	1.7 GB
ChIP-Seq SRR032209	CRAM	608 MB	2.36 GB	8.1 GB	4.3 GB
	Samcomp			300 MB	318 MB
	HUGO			14.8 GB	2.3 GB
Cancer HCC1954	CRAM	29.7 GB	92.8 GB	9.6 GB	9.1 GB
	Samcomp			320 MB	330 MB
	HUGO			46 GB	37 GB

HUGO, Hierarchical mUlti-reference Genome cOmpression; SAM, Sequence Alignment/Map.

obvious (ie, CR decreases by at least 0.05) improvement in the rate of EMRs when compared to direct alignment of the shortened reads against hg19 reference alone. The reason is that almost all target sequences are close to the reference sequence hg19, and we do not have other reference sequences that represent these sequences any better. As shown in figure 3, the adoption of a multi-reference based structure even generates slightly less exact mapped reads than the single reference based framework for the input sequence NA12878chrom20. Our method is limited by the availability of 'useful' reference genomes, which might have identical or near-identical structure characteristics to those of the sequence being compressed. For this reason, the order of the reference genomes used in hierarchical alignment is also important. The best yield can be achieved when the most similar reference to the source reads is chosen, even though part of these differences stems from sequencing errors and not from genetic variation. This could be done, for example, by matching phenotype characteristics such as ethnicity or disease, or by leveraging additional information such as family data. For example, Illumina recently started to provide whole genome sequence and variant call data for 17 members of the Coriell CEPH/UTAH 1463 family (http://www.illumina.com/platinumgenomes/) in order to create a 'platinum' standard comprehensive set of variant calls. We expect that our multi-reference method can be useful for compressing 'pedigree' genomes that are becoming increasingly available. For genomes with a high number of somatic mutations it is still difficult to find an appropriate reference, but as more data become available the similarity data could be recorded for purposes other than compression.

On the other hand, the compression time, required memory, and the success of our proposed method strongly depend on the SOAP3 alignment tool, which sets a high bar for the running environment. Users need a Linux workstation equipped with a multi-core CPU (default quad-core) with at least 20 GB main memory and a CUDA-enabled GPU with compute capability 2.0, if the GPU-based alignment tool is used, and at least 3 GB memory (default 6 GB). Fortunately, new alignment tools are being developed with increasing speed and sensitivity every year, which naturally boosts the efficiency of our proposed method.

CONCLUSION

Storage and transmission are important challenges in the use of large sequencing '*Big Data*'. We developed a novel compression technique, the HUGO framework, for compressing aligned reads. Our method also presents an innovative way of hierarchically matching gradually shortened reads in order to make full use of available reference genomes. Our experiments compared the performance of our algorithm with other state-of-the-art compression algorithms, such as CRAM, to which it was superior, and Samcomp, which had similar compression performance.

Contributors PL and XJ designed the study and were responsible for the experiments, SW assisted with GPU computing and manuscript preparation, JK assisted with the analyses, HX and LO-M wrote parts and edited all versions of the manuscript.

Funding This work was supported by NIH grant numbers K99LM011392 and U54HL108460, NLM grant number R00LM011392, and NSFC grant numbers U1201255, 61271218 and 61228101.

Competing interests None.

Provenance and peer review Not commissioned; externally peer reviewed.

REFERENCES

1 Ng SB, Turner EH, Robertson PD, *et al*. Targeted capture and massively parallel sequencing of 12 human exomes. *Nature* 2009;461:272–6.
2 Abecasis GR, Altshuler D, Auton A, *et al*. A map of human genome variation from population-scale sequencing. *Nature* 2010;467:1061–73.
3 The 1000 Genomes Project Consortium. An integrated map of genetic variation from 1,092 human genomes. *Nature* 2012;491:56–65.
4 Sequence Read Archive. http://www.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?view=announcement/
5 Altschul SF, Gish W, Miller W, *et al*. Basic local alignment search tool. *J Mol Biol* 1990;215:403–10.
6 Liu Y, Schmidt B, Maskell DL. CUDASW++ 2.0: enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions. *BMC Res Notes* 2010;3:93.
7 Langmead B, Trapnell C, Pop M, *et al*. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol* 2009;10:R25.
8 Pearson WR, Lipman DJ. Improved tools for biological sequence comparison. *Proc Natl Acad Sci* 1988;85:2444–8.
9 Langmead B. Aligning short sequencing reads with Bowtie. *Curr Protoco Bioinformatics* 2010;32:11.7.1–11.7.14.
10 Jorde LB, Wooding SP. Genetic variation, classification and 'race'. *Nat Genet* 2004;36:S28–33.
11 Li H, Handsaker B, Wysoker A, *et al*. The sequence alignment/map format and SAMtools. *Bioinformatics* 2009;25:2078–9.
12 Bentley DR, Balasubramanian S, Swerdlow HP, *et al*. Accurate whole human genome sequencing using reversible terminator chemistry. *Nature* 2008;456:53–9.
13 Margulies M, Egholm M, Altman WE, *et al*. Genome sequencing in microfabricated high-density picolitre reactors. *Nature* 2005;437:376–80.
14 Dorff KC, Chambwe N, Zeno Z, *et al*. GobyWeb: simplified management and analysis of gene expression and DNA methylation sequencing data. *Quantitative Methods arXiv* 2012;1211:6666.
15 DePristo MA, Banks E, Poplin R, *et al*. A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nat Genet* 2011;43:491–8.
16 McKenna A, Hanna M, Banks E, *et al*. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Res* 2010;20:1297–303.
17 Sakib MN, Tang J, Zheng WJ, *et al*. Improving transmission efficiency of large sequence alignment/map (SAM) files. *PloS ONE* 2011;6:e28251.
18 Ohno-Machado L, Bafna V, Boxwala AA, *et al*. iDASH: integrating data for analysis, anonymization, and sharing. *J Am Med Inform Assoc* 2012;19:196–201.
19 Hach F, Numanagić I, Alkan C, *et al*. SCALCE: boosting sequence compression algorithms using locally consistent encoding. *Bioinformatics* 2012;28:3051–7.
20 Fritz MH-Y, Leinonen R, Cochrane G, *et al*. Efficient storage of high throughput DNA sequencing data using reference-based compression. *Genome Res* 2011;21:734–40.
21 Jones DC, Ruzzo WL, Peng X, *et al*. Compression of next-generation sequencing reads aided by highly efficient de novo assembly. *Nucleic Acids Res* 2012;40:e171.
22 Popitsch N, von Haeseler A. NGC: lossless and lossy compression of aligned high-throughput sequencing data. *Nucleic Acids Res* 2013;41:e27.
23 Bonfield JK, Mahoney MV. Compression of FASTQ and SAM format sequencing data. *PloS ONE* 2013;8:e59190.
24 Liu C-M, Wong T, Wu E, *et al*. SOAP3: ultra-fast GPU-based parallel alignment tool for short reads. *Bioinformatics* 2012;28:878–9.
25 Genome. ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/pilot2_high_cov_GRCh37_bams/data/NA12878/alignment/NA12878.chrom20.ILLUMINA.bwa.CEU.low_coverage.20111114.bam
26 Cleary J, Witten I. Data compression using adaptive coding and partial string matching. *IEEE Trans Commun* 1984;32:396–402.
27 Moffat A. Implementing the PPM data compression scheme. *IEEE Trans Commun* 1990;38:1917–21.
28 Kozanitis C, Saunders C, Kruglyak S, *et al*. Compressing genomic sequence fragments using SlimGene. *J Comput Biol* 2011;18:401–13.
29 Deorowicz S, Grabowski S. Compression of DNA sequence reads in FASTQ format. *Bioinformatics* 2011;27:860–2.
30 Xu H, Luo X, Qian J, *et al*. FastUniq: a fast de novo duplicates removal tool for paired short reads. *PloS ONE* 2012;7:e52249.
31 MacQueen J. Some methods for classification and analysis of multivariate observations. In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*; California, USA: 1967.