Rapid, Separable Compression Enables Fast Analyses of Sequence Alignments

Darya Filippova
Computational Biology Department
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA, USA
dfilippo@cs.cmu.edu

Carl Kingsford
Computational Biology Department
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA, USA
carlk@cs.cmu.edu

ABSTRACT

Continued growth of generated sequencing data demands novel scalable approaches to its storage and transmission. It is also crucial that analyses can be run on data in its compressed form without having to fully reconstruct it. We propose a novel approach to compression of sequence alignment data, a well established data format that is used for a variety of tasks ranging from genome assembly to variant calling. Such alignment files may exceed the size of the original sequence by an order of magnitude, however, REFERENCE, our tool implementing the approach, is able to compress alignment files to 1/10 of the original SAM file size and is twice as efficient as SAM's binary BAM variant. REFERENCE is fast, highly parallelizable, and outperforms state of the art tools by an average of 8.1% while enabling a variety of sequence-related tasks that require only a partial decompression. Computations like depth of sequencing that involve seeking through all alignments take from 8 to 44 seconds for REFERENCE as opposed to tens of minutes with `samtools`. REFERENCE uses a lightweight streaming clustering algorithm to improve quality values compression and encodes sequence information very efficiently, with compression rates as low as 0.06 bits per base. Its modular structure allows one to omit extraneous alignment information from the download reducing sequencing data from many gigabytes to under a hundred megabytes.

Categories and Subject Descriptors

E.4 [Coding and Information Theory]: Data compaction and compression

General Terms

Algorithms, compression, theory

Keywords

Compression, sequence alignment

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
BCB'15, September 9–12, 2015, Atlanta, GA, USA.

Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3853-0/15/09 ...\$15.00.
<http://dx.doi.org/10.1145/2808719.2808739>.

1. INTRODUCTION

With increasing amounts of sequencing data being generated, the problem of storing, transmitting, and analyzing such data at scale becomes a bottleneck. Various compression methods designed for sequencing data can significantly reduce the size of data, alleviating the storage burden and making transmission times faster. *De novo* compression methods accept raw sequencing reads, FASTA, or FASTQ files as input and focus on ways to compress sequence [16, 11, 18], quality values [1, 22, 9], or both [20, 19]. Tools like SCALCE [19] and Mince [16] take the approach of grouping the reads based on their similarity allowing the downstream general-purpose compression tools to build better dictionary encodings. Rozov et al. [18] and recent work by Kingsford and Patro [11] use an available reference sequence to inform their compression algorithm without explicitly aligning reads to it. Quip [10] takes a step closer to reference-based approach by creating contigs based on the subset of data and encoding the rest of the reads relative to their lightweight assembly. *De novo* compression tools compress sequence down to 3-10% of its original size at reasonable speeds [3], however, most tools require that the dataset is completely decompressed before it could be used for analysis.

A process of mapping reads to a known reference sequence is often a prerequisite for many downstream tasks like assembly [7], expression estimation [21, 13], or variant calling [4], however, alignment files stored in a popular SAM format [14] occupy orders of magnitude more space than the original sequencing reads themselves. A commonly used approach is to convert SAM files to its block-compressed equivalent, BAM, offers a significant improvement in the used disk space while offering the same set of analyses at a low computational cost. Later approaches improve the compression ratio by sacrificing the ability to run analyses without full decompression [2, 10], by turning to lossy compression, or by doing without certain sequence data [12, 6]. More recently, Hach et al. introduced a more efficient block-wise BAM/SAM compressor that allows for block-wise decompression while offering savings of up to 44% over BAM [8].

We introduce REFERENCE that makes global sequence analyses possible without ever fully reconstructing the alignments and further improves SAM compression rates by leveraging separability of data within alignments. REFERENCE performs common tasks at speeds orders of magnitude faster than those for similar tasks within `samtools` package [14] and compresses data as fast as the existing alternatives. REFERENCE operates by grouping related data and converting it

into its more compact representation, then passing it down to a general-purpose dictionary coder like `plzip` for further compression. REFERENCE can compress sequence data down to 0.06 bits per base (a 33x improvement over 2-bit encoding for the same data), its total compressed sizes can be as small as 1/10 of the original SAM file or up to 50% smaller than an equivalent BAM file. On average, REFERENCE achieves an 8.1% improvement over the next best approach [8]. REFERENCE reduces the size occupied by quality values by identifying highly similar quality vectors in a streaming fashion and encoding them separately from the rest of the qualities. REFERENCE can compress a 13.5Gb SAM file to 1.5Gb in about 10 minutes (with 10 threads) and while using less than 2Gb of memory. It is highly parallelizable and can be made significantly faster by allowing it to use more threads and memory. When operating on sequencing data exclusively, REFERENCE can process the same 13.5Gb file in under three minutes requiring <100Mb of disk space to losslessly represent aligned sequences, a two- to fourfold improvement over competing approaches. Global statistics like depth of coverage can be computed using these compressed sequence data alone and take 44 seconds for a 72Gb SAM file as opposed to 48 minutes when performed by `samtools`.

2. METHODS

2.1 Improving data homogeneity

REFERENCE uses an idea of separable streams to both improve compression rates and to allow for efficient stream-specific computations on compressed data. Data pertaining to sequence, specifics of the alignments, read identifiers, and sequence qualities are all encoded separately. These data streams are structured in a flat, sequential manner (unlike hierarchical organization in Goby [2]). Within each stream, we re-organize and transform the data for the purposes of minimizing the size of data and representing the data in a more homogenous fashion by remapping it into a smaller, more uniform domain. The data in each stream is organized into blocks that are then passed down to a general-purpose dictionary coder. The dictionary coder is then able to capture the exposed redundancy and to compress the preprocessed streams more efficiently. This strategy was previously shown to work well for *de novo* sequence compression [19, 16]. Additionally, we record the genomic coordinates corresponding to each block of data in a stand-alone lightweight index that provides random access to original data without reconstructing alignments fully. We use `plzip` for downstream compression since it is fast and highly parallelizable in addition to offering a slight compression improvement on most of our inputs. Below, we discuss specific operations applied to sequencing, numerical, and quality values data.

2.2 Estimating reference-based compression rates

Suppose a sequencing read r can be mapped to the reference sequence G (i.e. a genome) without errors. Then the only information needed to reconstruct the read is its mapping position within the reference and its length. Assuming uniform read length and given the reference length in bases, $|G|$, we would only need $\lceil \log_2 |G| \rceil$ bits to encode the mapping position. Given a collection of a N reads of length $|r|$, we can then compress the input $N|r|$ bytes to $N\lceil \log_2 |G| \rceil / 8$ bytes, resulting in a compression ratio of $\lceil \log_2 |G| \rceil / (8|r|)$. For example, ten million human RNA-seq

100-bp long reads would occupy $10^7 \times 100 = 10^9$ bytes, or 953Mb on disk. We can compress these data down to around $10^7 \times \lceil \log_2 3 \times 10^8 \rceil / 8$ bytes, or 33Mb, given a 300Mb transcriptome length and assuming that every read mapped without errors. This would be equivalent to spending just 0.29 bits on average to encode a single base, a sevenfold improvement over the classic 2-bit sequence encoding.

This theoretical argument could be extended to account for sequencing errors and variation. A single error can be recorded using a short binary code and its position relative to the beginning of the read. To encode a position, one would spend $\lceil \log_2 |r| \rceil$ bits of information. Similarly, one would spend $\lceil \log_2 |E| \rceil = 3$ bits to encode an operation given the set of six edit operations $E = \{A, C, G, T, I, D\}$ — four mismatches for individual nucleotides, an insertion, and a deletion. Altogether, a single error can be recorded using $3 + \lceil \log_2 |r| \rceil$ bits of information. Assuming an error rate of at most 3 edits per read, we would spend at most $10^7 \times 3 \times (\lceil \log_2 |E| \rceil + \lceil \log_2 |r| \rceil) = 10^7 \times 3 \times 7$ bits of information, or 35Mb, to encode errors in the above ten million reads — a 3.6% of the size of the original sequence.

In reality, our estimate of $\log_2 |G|$ bits per read is an upper bound on the amount of information needed to record the mapping positions, and REFERENCE achieves better compression rates (especially for deep coverage RNA-seq experiments — see Table 3). When mapping positions are sorted, recording the intervals between pairs of consecutive positions (*delta encoding*) allows the reconstruction of all the positions exactly while representing the sequence in a smaller domain — therefore, requiring fewer bits per single position. In biological data, the sequencing depth may further aid in compression: for multiple reads mapping to the same position it is enough to save the position once and separately maintain the count of reads mapping there.

2.3 Practical sequence compression implementation

REFERENCE separates sequence and mapping position information from the rest of the SAM data and represents it in a concise manner making it more amenable to efficient downstream compression. SAM format uses `RNAME` and `POS` columns to encode the mapping address and `CIGAR` and `MD` columns to encode the differences, if any, between the read and the reference sequence. Storing the original read sequence in `SEQ` column is, in fact, unnecessary when the mapping and all the differences between the reference and the read are known. REFERENCE parses `CIGAR` and `MD` strings and merges the differences encoded within each string into a single list of edits. All `POS` belonging to a single reference sequence `RNAME` are organized into a sorted list of locations with reference sequences appearing in the same order as in the SAM file. The order in which REFERENCE records merged edit lists matches the order in which the reads appear in the sorted SAM file which, in turn, is equivalent to the order in which reference sequences and the list of mapping locations appear in REFERENCE's format.

To efficiently encode information from `RNAME` and `POS` fields, all mapping locations for a single `RNAME` are organized into a single sorted list. To encode reads represented by the list of mapping locations, REFERENCE spells out the reference sequence identifier, followed by the sorted list of locations. The list of mapping locations is a sequence of non-decreasing highly repetitive integers that naturally lend themselves to

an efficient delta and run-length encoding (RLE) scheme. Taken together, a delta encoding and RLE transform a list of mapping locations $d_{11}, d_{12}, \dots, d_{1q_1}, d_{21}, \dots$ into a list of pairs $(d_1, q_1), (d_2, q_2), \dots$ where d_i is the offset in bases from the previous mapping location and q_i represents the number of reads mapping to that position in the reference. The list minimizes the volume of data sent to the downstream compressor thus decreasing REFERENCE's memory usage and improving runtime. Additionally, such an approach to encoding offsets makes REFERENCE robust to increasing sequencing depths and allows one to losslessly encode data from **RNAME** and **POS** columns in space on the order of 0.01% (0.1%) of the original SAM file size for RNA-seq data (whole-genome data respectively).

To efficiently encode all differences between the reference sequence and the read, REFERENCE parses **CIGAR** and **MD** strings and merges edits represented by each string (alternatively, REFERENCE can obtain mismatch locations by comparing read to the reference genome if **MD** string is not available). **CIGAR** strings are capable of encoding split junctions, clipped sequences, indels, and mismatches, although it is a common practice to omit mismatch information or encode it elsewhere. **MD** strings represent mismatch positions within the aligned part of the read. Taken together, the information in **CIGAR** and **MD** strings allows the read to be reconstructed exactly. REFERENCE accounts for duplicate edits between the two strings, orders the list of merged edits by their position within the read, and delta-encodes the positions — in contrast to saving edited **CIGAR** strings as in [8]. A list of edits for a single alignment is then appended to the end of an edit stream and REFERENCE writes a 1 to the end of a separate bit vector B . If the read aligned to the reference with no errors, no edit list is generated, REFERENCE appends a 0 to the bit vector B , and proceeds to the next alignment.

Merging the edits and maintaining a bit vector B to differentiate exact alignments reduces amount of data passed down to the common-purpose compressor. A large percent of **CIGAR** strings encode edit data inefficiently: for example, a single insertion in the middle of a 32bp read would be represented as an 8 byte string **20M1I11M** where 8 bits would suffice as discussed above. Likewise, a large proportion of **MD** strings spend additional space to encode mismatches, even in the case when the read and the reference are identical, and **MD** string is redundant. Lastly, a significant fraction of reads may match the reference exactly, in which case both edit strings are redundant and can be replaced by a single bit in B . Our experiments on test data show that maintaining a bit vector B to record exact alignments separately from the edit lists allows to compress edit data more efficiently than when maintaining separate edit strings for every read. B itself is highly amenable to compression: runs inexact alignments translate into series of identical bytes that are efficiently compressed by the run-length coders.

The order of 1's in B corresponds to the order in which edit lists appear in the edit stream making it easy to match an alignment to its edits when restoring the original sequences. Some biologically meaningful edits, e.g. single nucleotide polymorphisms, would have the same relative encoding from one alignment to another. Storing such edits in a single stream allows the downstream dictionary compressor to pick up on the similarities and to capitalize on such consistent edits.

However, the compression benefits due to such co-located edits would be minor. To estimate the savings from explicitly identifying consistent edits and encoding them via a modified contig as described in [8], we first identified all bases that had at least one read with an edit in that location. We assumed that a base had to have over half of all reads aligning to it agree on a single edit for that edit to be beneficial for compression. For example, if a given position in the reference sequence has a 'C' and there are 100 reads aligned to it with 40 reads agreeing with the reference, 30 of the reads having an 'A' instead of a 'C', and the remaining 30 reads having a 'G', modifying the reference to contain an 'A' or a 'C' would result in 70 of the reads differing from the new reference sequence and only 30 reads matching it exactly. However, if the distribution of the read counts was such that only 10 reads were matching the original reference, 60 reads contained an 'A' and 30 reads contained a 'G', then modifying the reference to contain 'A' would result in 60 reads matching the reference and only 40 reads having an edit. We observe that across all tested datasets less than 10% of all loci that have mapping errors display a significant agreement, i.e. had a single edit repeated among > 50% of all reads mapped to the same locus.

2.4 Reads mapping to multiple loci

A single sequencing read might map to several locations in the reference sequence with reasonable accuracy, and every such mapping will appear as a separate alignment in the SAM file. By default, REFERENCE will treat secondary alignments as independent of the primary alignment and encode them accordingly. Users can specify **-discardSecondary** flag to enable REFERENCE to only retain primary alignments and unaligned reads.

2.5 Unaligned sequence

REFERENCE compresses the unaligned sequences separately from the rest of the alignment data and stores unaligned reads along with their quality values and read identifiers in FASTQ format. Since read order does not matter for these reads, the reads are reordered lexicographically in batches as REFERENCE accumulates them while compressing alignments. Before reordering the reads, each sequence is compared to its reverse complement. REFERENCE chooses to keep the original sequence or its reverse complement based on which minimizer was lexicographically smaller. This operation reduces the space of substrings observed in the unaligned reads and aids in compression. A more sophisticated compression tool like Mince [16] can further reduce compressed size of the unaligned reads.

2.6 Quality values

We use three observations to compress quality vectors efficiently: the fact that quality values often get worse towards the end of the read [1]; that a large percent of all quality vectors are mostly consistent (i.e. a single quality value dominates the vector); and that many vectors share a sufficiently long subsequence. We also observe that — at least for Illumina quality vectors — the prefix may also contain a sequence of low values that tend to be similar across many of the vectors. Additionally, we observe that for many datasets, several quality values are significantly more frequent than the others. For example, for a *P. aeruginosa* dataset, the top 6 quality values account for almost 80% of

all characters observed in quality vectors (top 10 values for SRR445718). The distribution of these values within vectors follows a certain trend as well: for 64% of the quality vectors in *P. aeruginosa*, occurrences of a single value add up to more than 50% of the vector's length. It is not surprising, therefore, that many vectors have similar profiles and can be encoded relative to each other in an efficient way.

To identify groups of quality vectors that share a significant amount of information, we devised a streaming clustering algorithm that separates the incoming vectors into distinct classes that are then compressed separately. Every incoming quality vector must first pass a filter to determine whether it is likely that the vector will compress well. If the vector does not meet the criteria (see details below), it is placed into a general pile — a special cluster that contains vectors that were not assigned a cluster otherwise. Vectors that passed the QC stage are then trimmed to separate the core substring from the lower quality prefix and suffix. REFERENCE then compares the core substring to the centroids of the existing clusters. If neither of the centroids is within distance 0.05 of the core substring, or if there are no clusters, REFERENCE forms a new cluster with the core substring as its centroid. Otherwise, the core substring and its corresponding prefix and suffix are placed into the first cluster passing the similarity threshold. REFERENCE maintains separate substreams within the cluster to store prefixes, suffixes, and core substrings of the vectors assigned to it. REFERENCE applies a run-length encoding to each substream before passing these data to the downstream compressor.

We use the quality vector's weighted mode q^* and the mode's frequency f_{q^*} to distinguish between the vectors that are likely to compress well and should proceed to the clustering stage and the vectors that will be placed into the general pile. To compute the weighted mode, we treat the quality vector v_i as a collection of numerical values and define weighted mode as $\arg \max_q n(q)w_q$ where $n(q)$ is the number of q 's occurrences in vector v_i and weight w_q is the ordinal value for q transposed so that all quality values fall within the $[0; q_{max}]$ range. The weighted mode resolves ambiguities in favor of "better" quality values when several values occur with near equal frequency in v_i . If the mode value is not frequent enough ($\leq 25\%$ of the vector's values), the variation in the vector makes it less likely to compress well, and we assign v_i to the general pile.

For a vector that has a sufficiently frequent mode, we trim its prefix and the suffix to account for flanking low quality values. We retain both substrings to be able to fully reconstruct the vector, but only use the remaining core string when comparing v_i to the clusters' centroids. The characters in v_i are considered low quality until we observe the mode q^* twice in a row. When such low quality prefixes and suffixes are separated from the rest of the quality vector and are stored separately, each substream — the prefixes, the core substrings, and the suffixes — compresses better.

We use $D2$ distance [15] to compute the similarity between v_i 's core substring and the candidate cluster's centroid. $D2$ captures substring similarity between two variable length strings and is lower if the two strings share most of their substrings. More specifically, $D2 = \sum_{s_i \in S} (s_{i1}/|v_1| - s_{i2}/|v_2|)^2$ where S is a union of all substrings of length k that occur in vectors v_1 and v_2 , s_{i1} (s_{i2}) represents the number of s_i 's occurrences in v_1 (v_2). Using $D2$ metric to compare two quality vectors allows REFERENCE to capture subsequence sim-

ilarity without enforcing a specific order between matching parts of the vector. We find that using substrings of length 4 allows $D2$ to accurately capture similarity between quality vectors at a reasonable speed.

REFERENCE runs a bootstrap stage on the first $b = 200000$ alignments to avoid forming too many clusters with a long tailed distribution of sizes. The size b of the bootstrap set is an input parameter and can be adjusted for different size datasets. REFERENCE reviews the clusters formed at the bootstrap stage and only retains the clusters that contain at least 5% of the vectors seen so far. Quality vectors in the clusters smaller than $0.05b$ are merged with the vectors in the general pile maintaining their relative order. The remaining clusters formed during the bootstrap are used to separate all of the following vectors. Finally, we record the cluster membership for every incoming quality vector into a separate data stream ensuring that we can match quality vectors to their corresponding alignments during decompression.

2.7 Encoding numerical fields

SAM format contains a number of numerical fields that encode details such as whether or not read was reverse complements or the quality of the alignment assigned by the mapping tool. Even though the domain for these fields can be rather large (for example, PNEXT field can assume values in $[0, 2^{31} - 1]$), in practice, there often are just a few frequent values that these fields assume. To exploit this fact, we make a mapping from the original domain to a new integer domain $[0, f - 1]$ where f is the number of unique values a field assumes in a given input file. On average, this increases homogeneity between the numerical fields within a single alignment and across many consecutive alignments.

2.8 Random access for sequencing reads

REFERENCE enables random access to data without having to decompress all of the streams: REFERENCE compresses data block by block with each block being compressed independently of its predecessor. The blocks can each be reconstructed separately without any knowledge shared between the blocks. REFERENCE creates a *.fai-like index for the blocks that keeps track of the genomic coordinates that span each block. lzlib format for the compressed data encodes information about the size of the blocks separately and makes seeking through the blocks easy given the format. The index only has to store a pair of coordinates for every block and an identifier for the type of stream the block corresponds to making such indices space-efficient and further amenable to delta encoding. The addition of such lightweight index makes decompression at random genomic coordinates possible: a functionality similar to viewing read pileups at random genomic locations (much like samtools view or IGV's visualization [17]). To extract original reads within a given genomic range, REFERENCE locates all data blocks overlapping the input genomic interval: using interval trees allows to do that in $O(\log m)$ times where m is the number of data blocks within a single stream. Overlapping blocks are then queued and are decompressed sequentially as REFERENCE starts stitching data coming from separate streams. Users can decompress subsets of the original data without ever having to reconstruct the original alignments fully.

3. RESULTS

Table 1: Comparison between BAM/SAM compression methods and Referee on human RNA-seq, bacterial and human WGS datasets. The second number for Referee indicates % improvement over the next best approach. The sizes are reported in megabytes.

File	Referee (%)	Quip	Deez	BAM	SAM
SRR445718	1242.67 (8.30%)	1355.5	1607.11	2282.45	9544.98
SRR1294122	1545.15 (10.7%)	1730.15	2022.63	3107.67	13650.17
<i>P. aeruginosa</i>	1866.87 (2.80%)	2180.95	1921.06	3339.54	19008.37
K562	7147.04 (10.6%)	8274.43	7997.7	13119.44	72398.19
NA12878_S1	62176.53 (-0.4%)	61904.65	62774.92	108289.6	437589.43

Table 2: Test set of SAM files. Experiments were performed on human and bacterial RNA-seq as well as human whole genome sequencing data (WGS) for reads of different lengths. All datasets, except K562, had a significant number of unaligned reads, varying in depth of coverage and error rates. Datasets mostly comprised of Illumina (I) sequences with one exception (Solexa, S).

File	Type	r	Alignments	Unalign.
SRR445718, I	RNA-seq	100	35677496	5677563
SRR1294122, I	RNA-seq	101	47933574	1859223
<i>P. aeruginosa</i> , S	RNA-seq	51	83412201	5560731
K562, I	RNA-seq	76	246476391	0
NA12878_S1, I	WGS	101	1015694132	202874775

REFeree operates in a lossless manner and achieves better compression rates than state of the art tools in the area while remaining competitive in compression speeds and enabling random access to data without full decompression. We test REFeree on a human and bacterial RNA-seq and human whole genome sequencing (WGS) data with varying number of alignments, depth of coverage, error rates, and read lengths (Table 2). We generated alignments for SRR445718 and SRR1294122 using STAR [5], a fast multi-threaded mapper, with different error rate cutoffs (see later sections). The K562_cytosol_LID8465_TopHat_v2 and NA12878_S1 datasets are available from EMBL-EBI’s ArrayExpress data repository, accession numbers ERX283488 and ERX237515 correspondingly; *P. aeruginosa* was available for download with DeeZ [8]. All alignment files were sorted and converted into SAM format using samtools [14]. Experiments were performed on a Dell PowerEdge T620 machine with 256Mb of RAM.

3.1 Referee achieves up to 10x compression on alignment data

REFeree outperformed its competitors on all but one test cases, saving hundreds of megabytes in each case (Table 1). Compared to the original plain text SAM format, REFeree offers seven- to tenfold improvement in storage; or up to twofold improvement over its binary variant BAM. Our results show that REFeree can effectively compress alignments from a wide range of depths of coverage: REFeree achieves equally good performance on bacterial *P. aeruginosa* sequenced at an average depth of 687 and human RNA-seq SRR445718 dataset with average depth of coverage of 32.

REFeree performed significantly better than DeeZ on human RNA-seq data where alignments contained a significant

number of splicing events and achieved a modest improvement for the bacterial RNA-seq alignments. Quip’s performance paralleled that of REFeree on two of the smaller human RNA-seq datasets, but Quip spent an additional gigabyte encoding alignments for the large K562 RNA-seq dataset. REFeree had an additional 272Mb over Quip for the human whole genome sequencing dataset (NA12878_S1), however, it required 600Mb less than DeeZ to encode the same information.

Quality values contributed significantly to the size of the compressed data adding up to as much as 80% of the total compressed size. At the same time, data encoding sequence took no more than 5.6% across all datasets with read identifiers and unaligned reads contributing most significantly to the overall size after qualities.

3.2 Referee compresses sequence up to 33 times better than 2-bit encoding

REFeree achieves significantly better compression rates for sequencing data within alignments over other competing methods (see Table 3), especially on datasets at high depth of coverage and low error rates. To compare REFeree performance to other tools on only the aligned sequence, we replaced read identifiers, quality values, and non-essential fields with default values. We further modified DeeZ to produce additional debugging information on the size of each stream and ran Quip in a verbose mode. Since both tools could retain some information from other SAM file columns or build indices to allow for fast data access, we only report the sizes for compressed streams that were directly relatable to sequence and not the total size of the file on disk (seq for Quip and the sum of seq and edits for DeeZ, see Table 3). For REFeree, we report a sum of stream sizes for alignment offsets, clipped prefixes and suffixes, and edits. REFeree surpasses both DeeZ and Quip for every dataset leaving a considerable margin.

The REFeree compression rates of bits spent to encode a single base vary from 0.06 to 0.24 — up to 33 times better than a straightforward 2-bit sequence encoding and up to 8 times better than the two competing methods. In section 2.2, we discussed how sequence compression quality can be related to read length and should be more effective on longer reads, however, our results show that other factors, like error rate and depth of coverage, can significantly affect the predicted performance (Table 3). We observe that REFeree and DeeZ scale up well with increasing depth of coverage (Figure 1(a)) while Quip’s rates drastically go up for the deep sequencing dataset *P. aeruginosa*. REFeree achieves best performance for the two datasets with the highest coverage, K562 and *P. aeruginosa*, which can be explained by the

Table 3: Referee can compress sequence very efficiently down to 0.03 bits per base (bpb). Sequence sizes are shown in Megabytes. Error rate was calculated as an average number of edits (clipping, indels, splices, and mismatches) over all aligned bases. Sequencing depth was calculated as the average number of aligned reads over the covered bases.

File	Total bases	Referee, Mb(bpb)	Quip	Deez	Error rate	Depth
SRR445718	3567749600	86.63 (0.20)	145.34 (0.34)	298.40 (0.70)	2.85	32.67
SRR1294122	4841290974	84.33 (0.15)	150.54 (0.26)	268.47 (0.47)	1.95	21.51
P. aeruginosa	4254022251	41.61 (0.08)	359.75 (0.71)	59.20 (0.12)	1.04	687.19
K562	18732205716	135.99 (0.06)	417.98 (0.19)	258.78 (0.12)	1.20	73.53
NA12878_S1	101569413200	2940.07 (0.24)	5845.28 (0.48)	4229.62 (0.35)	1.50	36.11

Figure 1: Referee outperformed both Quip and Deez in sequence compression: Referee handles increasing error rates and sequencing depths better than its competitors. Deez's rates deteriorate with increasing error rate and Quip's rates tend to be significantly affected by the sequencing depth. Each line in the plots represents a single file; files correspond to those in Table 3.

fact that these also happen to be the two datasets with the lowest error rates. Indeed, compression rates for REFERENCE's and Quip start to decrease slowly as error rates increase — these two tools are more robust to changes in error rates than Deez for which compression rates quickly deteriorates at higher error rates (Figure 1(b)).

REFERENCE also compresses sequence more efficiently than CRAM [6], the first work to consider reference-based sequence compression, for the same depths, error rates, and read lengths. CRAM estimates the bpb ratio to be between 0.175 and 0.20 for unpaired 100bp reads at 1% error rate

and sequencing depth of 25. However, REFERENCE achieves a ratio of 0.14bpb at a higher error rate and similar coverage. As error rates decrease, REFERENCE compression rates improve and are comparable to the rates for 200 and 400 base long reads reported by CRAM (see Figure 2 in [6]).

3.3 Clustering improves quality values compression

Although high variability of quality values makes them notoriously hard to compress, REFERENCE's clustering scheme results in 13 to 17% reduction in compressed size for quality values over `plzip` without a significant increase in runtime or memory used. Across the tested datasets, REFERENCE performs similarly or better than arithmetic coding (AC) schemes employed by Deez and Quip (see Table 4) while making partial decompression feasible. For quality vectors that get placed into clusters, the average compression rate lies between 6.2 and 10.9. Combined with a considerably lower compression ratio for the quality vectors placed in the generic pile, the overall compression rate for qualities varies between 3.33 and 4.22 on the tested datasets.

The number of quality clusters did not exceed four (including the generic pile) with the number of vectors assigned to non-generic clusters ranging from 31 to 67% over all datasets. For most datasets, two clusters formed consistently: a cluster of vectors consisting of mostly the highest quality value and a smaller cluster of vectors consisting almost entirely of the lowest quality value. For vectors in clusters, collections of core substrings compressed down to $1/3 - 1/4$ of the original size. In general, quality vectors compressed better when separated into prefixes, cores, and suffixes as opposed to vectors that were not split, e.g. clusters for SRR1294122 before separation would compress at 5.6, 5.7, and 4.5 ratios, but with prefixes and suffixes compressed separately the compression rate over the three clusters increases to 6.3, 6.6, and 4.8 correspondingly.

Table 4: Compressed quality values, sizes shown in Mb. Referee's clustering approach improves quality values compression up to 15%.

File	REFERENCE	Deez	Quip
SRR445718	976.76	1055.70	984.54
SRR1294122	1092.92	1328.17	1226.99
P. aeruginosa	1160.75	1084.87	994.1
K562	5358.94	5830.98	5715.71
NA12878_1	32560.26	38413.76	38097.35

3.4 Operations on separate streams and random access

Separability of data streams in REFEREE allows for field-specific analyses that are order of magnitudes faster than comparable tasks done with, e.g. `samtools` [14]. A common task such as computing a dataset’s sequencing depth involves going through all alignments recording number of reads aligned to any covered base. `samtools depth` takes about 19 minutes to compute depth for a modestly sized SRR445718 dataset and over 48 minutes for K562 while REFEREE accomplishes that same task in 8 and 44 seconds correspondingly. The difference is explained by the fact that `samtools` has to scan through all of alignment data, including data unrelated to depth calculation, while REFEREE accomplishes the same task by only having to go through compressed offsets (about 5% of the compressed data).

Likewise, REFEREE format allows one to quickly reconstruct aligned reads originally embedded in SAM file (an equivalent of `samtools faidx` command) without ever reconstructing the alignments fully. Users can specify a genomic region of interest, and REFEREE can reconstruct alignments from that genomic region without decompressing the whole dataset. The index that REFEREE constructs to enable partial decompression is lightweight: for example, to index all data blocks for NA12878, REFEREE uses 1.2Mb which can be further compressed down to 233Kb – a size negligible when compared to the size of the data itself.

3.5 Speed and memory demands

REFEREE is fast and only requires on the order of 1.5Gb of RAM to run making it usable on even smaller machines (see Table 5). REFEREE streams through the alignments in a sequential fashion only keeping intermediate information in memory before it is passed on to `lzlib`. REFEREE launches several threads that each invoke `lzlib` operations on a single block of data at a time making use of modern multithreading architectures. REFEREE will run faster if given more processing power, however, REFEREE can reach a saturation point where data blocks prepared by REFEREE are compressed at a faster rate than the new blocks are being produced.

Table 5: Referee is comparable or faster to the other competing tools. We report wall running times and % of CPU used by each tool (in parentheses). Both DeeZ and Referee were run with 10 threads, Quip was run in its default mode since it does not offer a multithreaded functionality.

File	Referee	Deez	Quip -r
SRR445718	7:05 (1017)	10:34 (190)	6:59 (139)
SRR1294122	10:26 (1016)	11:07 (204)	7:48 (149)
P. aeruginosa	18:41 (608)	7:34 (163)	9:18 (153)
K562	54:09 (620)	50:07 (180)	36:06 (157)
NA12878_S1	5:18:12 (828)	3:00:08 (190)	3:13:31 (164)

REFEREE could be run in a lossy manner in which only the essential information about sequence and alignment is recorded and information like read identifiers, flags, quality values, and optional SAM fields are not stored. Users can enable this mode by providing an optional “`-seqOnly`” parameter: REFEREE would encode offsets and edits for every alignment along with the unaligned reads. Alternatively,

users can further restrict REFEREE to only encode primary alignments by using a “`-discardSecondary`” option. Since REFEREE has to look over less data, sequence only mode is much faster than the lossless operation: for example, the sequence in either of the three smallest RNA-seq datasets can be compressed in under 3 minutes.

Given the fact that modern mapping tools can generate alignments in the matter of minutes even for large files, the “`-seqOnly`” options make REFEREE a reasonable choice when the goal is to compress sequencing data. Indeed, an aligner like STAR [5] can process 11Gb human RNA-seq dataset (e.g. SRR1294122) in under five minutes generating alignments for 94.31% of all reads. REFEREE takes under three minutes to compress the resulting SAM file with “`-seqOnly -discardSecondary`” options encoding the aligned reads in 2.1% of their original size and the unaligned reads in 25% of their original size. Overall, this combined scheme encodes all of the 39.7 million reads in 3.3% of their original size requiring only eight minutes to do so. Considering REFEREE’s consistently good performance at different error rates (Figure 1), we suggest that for the purposes of sequence compression, it is reasonable to trade off the quality of alignments for the number of aligned reads.

4. DISCUSSION & CONCLUSIONS

We present a novel approach for efficient compression and storage of the alignment information that enables certain computational analyses to be performed without ever having to decompress the data fully. REFEREE operates by separating related streams of data and encoding them individually achieving better compression rates for each stream. REFEREE is competitive in speed and performance to other state of the art tools that employ more complex compression schemes [8, 10]. We show that REFEREE can compress alignment data down to $1/10^{th}$ of its original SAM file size. When focusing on alignments that retain only sequence data (i.e. stripped of extraneous information like read identifiers), REFEREE can compress it in a lossless manner 2 to 9 times better than the competing methods (see Table 3). REFEREE’s efficient encoding of mapping locations and sequence variation makes it robust to increasing error rates (Figure 1). Coupled with the fact that modern alignment tools can successfully map millions of sequencing reads in minutes, REFEREE becomes an appealing choice in sequence compression, especially for the rapidly growing body of RNA-seq and WGS datasets for the model organisms for which well-annotated reference genomes are available.

Many analyses relying on the alignment information do not require original read identifiers or unaligned reads focusing instead on the sequence that aligned successfully. In such cases, being able to download only the data one needs may significantly reduce the download size. REFEREE separates data streams into distinct compressed files and only requires a subset of them to reconstruct the original SAM/BAM data. Read identifiers, quality values, optional SAM fields and all of the unaligned reads can be skipped; that means a reduction of download size by 84-93% (see Figure 2).

REFEREE is written in C++ and is available on GitHub (<https://github.com/Kingsford-Group/referee>).

5. ACKNOWLEDGEMENTS

Figure 2: Quality values and read identifiers contribute significantly to the size of the compressed file, however, Referee does not rely on these data to reconstruct sequence alignments. Omitting the non-essential alignment data can reduce the size of the download by up to 93%.

The authors would like to thank Rob Patro and Geet Duggal for many valuable discussions. This research is funded in part by the Gordon and Betty Moore Foundation's Data-Driven Discovery Initiative through Grant GBMF4554 to C.K. It is partially funded by the US National Science Foundation (CCF-1256087, CCF-1319998) and the US National Institutes of Health (R21HG006913, R01HG007104). C.K. received support as an Alfred P. Sloan Research Fellow.

6. REFERENCES

- [1] J. K. Bonfield and M. V. Mahoney. Compression of FASTQ and SAM format sequencing data. *PLoS ONE*, 8(3):e59190, 03 2013.
- [2] F. Campagne, K. C. Dorff, N. Chambwe, J. T. Robinson, and J. P. Mesirov. Compression of structured high-throughput sequencing data. *PLoS ONE*, 8(11):e79871, 11 2013.
- [3] S. Deorowicz, A. Danek, and S. Grabowski. Genome compression: A novel approach for large collections. *Bioinformatics*, 29(20):2572–2578, 2013.
- [4] M. DePristo, E. Banks, R. Poplin, K. Garimella, J. Maguire, C. Hartl, A. Philippakis, G. del Angel, M. Rivas, M. Hanna, A. McKenna, T. Fennell, A. Kernysky, A. Sivachenko, K. Cibulskis, S. Gabriel, D. Altshuler, and M. Daly. A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nature Genetics*, 43:491–498, 2011.
- [5] A. Dobin, C. A. Davis, F. Schlesinger, J. Drenkow, C. Zaleski, S. Jha, P. Batut, M. Chaisson, and T. R. Gingeras. STAR: ultrafast universal RNA-seq aligner. *Bioinformatics*, 29(1):15–21, 2013.
- [6] M. H.-Y. Fritz, R. Leinonen, G. Cochrane, and E. Birney. Efficient storage of high throughput DNA sequencing data using reference-based compression. *Genome Research*, 21:734–740, 2011.
- [7] S. Gnerre, E. S. Lander, K. Lindblad-Toh, and D. B. Jaffe. Assisted assembly: how to improve a de novo genome assembly by using related species. *Genome Research*, 10:R88, 2009.
- [8] F. Hach, I. Numanagic, and S. C. Sahinalp. DeeZ: reference-based compression by local assembly. *Nature Methods*, 11:1082–1084, 2014.
- [9] L. Janin, G. Rosone, and A. J. Cox. Adaptive reference-free compression of sequence quality scores. *Bioinformatics*, 30(1):24–30, 2014.
- [10] D. C. Jones, W. L. Ruzzo, X. Peng, and M. G. Katze. Compression of next-generation sequencing reads aided by highly efficient de novo assembly. *Bioinformatics*, 40(22), 2012.
- [11] C. Kingsford and R. Patro. Reference-based compression of short-read sequences using path encoding. *Bioinformatics*, 31(12):1920–1928, 2015.
- [12] C. Kozanitis, C. Saunders, S. Kruglyak, V. Bafna, and G. Varghese. Compressing genomic sequence fragments using SlimGene. *Journal of Computational Biology*, 18(3):401–413, 2011.
- [13] B. Li and C. N. Dewey. RSEM: accurate transcript quantification from RNA-Seq data with or without a reference genome. *BMC Bioinformatics*, 12:323, 2011.
- [14] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. Durbin. The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, 2009.
- [15] X. Liu, L. Wan, J. Li, G. Reinert, M. S. Waterman, and F. Sun. New powerful statistics for alignment-free sequence comparison under a pattern transfer model. *Journal of Theoretical Biology*, 284(1):106–116, 2011.
- [16] R. Patro and C. Kingsford. Data-dependent bucketing improves reference-free compression of sequencing reads. *Bioinformatics*, In press, 2015.
- [17] J. T. Robinson, H. Thorvaldsdóttir, W. Winckler, M. Guttman, E. S. Lander, G. Getz, and J. P. Mesirov. Integrative genomics viewer. *Nature Biotechnology*, 29:24–26, 2011.
- [18] R. Rozov, R. Shamir, and E. Halperin. Fast lossless compression via cascading Bloom filters. *BMC Bioinformatics*, 15(Suppl 9):S7, 2014.
- [19] S. C. Sahinalp, F. Hach, I. Namanagik, and C. Alkan. SCALCE: boosting sequence compression algorithms using locally consistent encoding. *Bioinformatics*, 28(23):3051–3057, 2012.
- [20] W. Tembe, J. Lowey, and E. Suh. G-SQZ: compact encoding of genomic sequence and quality data. *Bioinformatics*, 26(17):2192–2194, 2010.
- [21] C. Trapnell, B. A. Williams, G. Pertea, A. Mortazavi, G. Kwan, M. J. van Baren, S. L. Salzberg, B. J. Wold, and L. Pachter. Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation. *Nature Biotechnology*, 28:511–515, 2010.
- [22] Y. Yu, D. Yorukoglu, and B. Berger. Traversing the k-mer landscape of NGS read datasets for quality score sparsification. In *RECOMB'14*, pages 385–399, 2014.