Compressing Genomic Sequence Fragments Using SLIMGENE

Christos Kozanitis[1], Chris Saunders[2], Semyon Kruglyak[2], Vineet Bafna[1], and George Varghese[1]

[1] University of California San Diego, La Jolla CA 92093, USA
[2] Illumina Inc, San Diego CA 92121, USA

Abstract. With the advent of next generation sequencing technologies, the cost of sequencing whole genomes is poised to go below $1000 per human individual in a few years. As more and more genomes are sequenced, analysis methods are undergoing rapid development, making it tempting to store sequencing data for long periods of time so that the data can be re-analyzed with the latest techniques. The challenging open research problems, huge influx of data, and rapidly improving analysis techniques have created the need to store and transfer very large volumes of data.

Compression can be achieved at many levels, including trace level (image data), sequence level (compressing a non-redundant genomic sequence), and fragment-level, which involves compressing a set of short, and redundant collection of fragment reads, along with quality-values on the base-calls. We focus on fragment-level compression, which is the pressing need today.

Our paper makes two contributions, implemented in a tool, SLIMGENE. First, we introduce a set of domain specific loss-less compression schemes that achieve over $40\times$ compression of fragments, outperforming bzip2 by over $6\times$. Including quality values, we show a $5\times$ compression using less running time than bzip2. Second, given the discrepancy between the compression factor obtained with and without quality values, we initiate the study of using 'lossy' quality values. Specifically, we show that a lossy quality value quantization results in $14\times$ compression but has minimal impact on downstream applications like SNP calling that use the quality values. Discrepancies between SNP calls made between the lossy and lossless versions of the data are limited to low coverage areas where even the SNP calls made by the lossless version are marginal.

1 Introduction

With the advent of next generation sequencing technologies[?,?,?,?], the cost of sequencing whole genomes has decreased dramatically in the past several years, and is poised to go below $1000 per human individual in a few years. As more and more genomes are sequenced, researchers are faced with the daunting challenge of interpreting all of the data. At the same time, analysis methods are undergoing rapid development making it tempting to store sequencing data for long periods of time so that the data can be re-analyzed with the latest techniques. The challenging open research problems, huge influx of data, and rapidly improving analysis techniques have created the need to store and transfer very large volumes of data.

The study of human variation, and genome wide association (GWAS) was traditionally accomplished using micro-arrays, for which the data is smaller by over 3 orders of magnitude. However, these GWA studies have been able to explain only a very small fraction of heritable variation present in complex diseases. Many researchers believe that whole genome sequencing may overcome some of the limitation of micro-arrays. The incomplete picture formed by micro-arrays, the many applications of sequencing (Ex: structural variations), and the expected improvement in cost and throughput of sequencing technology ensure that sequencing studies will continue to expand rapidly. The question of data handling must therefore be addressed.

Even with the limited amount of genetic information available today, sites such as the Broad Institute and the European Bioinformatics institute are among the biggest storage consumers in the world, spending millions of dollars on storage [?]. Beyond research laboratories, the fastest growing market for sequencing studies is big pharmaceutical companies [?]. Further, population studies on hundreds of thousands of individuals in the future will be extremely slow if individual disks have to be shipped to an analysis center. The *single* genome data set we use for our experiments takes 285Gb in uncompressed form. At network download rate of 10Mb/s this data set would take 63.3 hours to transfer over the Internet. In summary, reducing storage costs and improving interactivity for genomic analysis makes it imperative to look for ways to compress genomic data.

While agnostic compression schemes like Lempel-Ziv[?] can certainly be used, we ask if we can exploit the specific domain to achieve better compression. As an example, domain-specific compression schemes like MPEG-2[?] exploit the use of a dictionary or reference specific to the domain. Here, we exploit the fact that the existing human assembly can be used as a reference for encoding. We mostly consider loss-less compression algorithms. Specifically, given a set of genomic data S, we define a compression algorithm by a pair of functions $(\mathcal{C}, \mathcal{D})$ such that $\mathcal{D}(\mathcal{C}(S)) = S$, The compression factor *c.f.*, defined by $|S|/|\mathcal{C}(\mathcal{S})|$ describes the amount of compression achieved.

The genomic data S itself can have multiple forms and depends upon the technology used. Threfore, the notion of 'loss-less' must be clarified in context. In the Illumina Genome Analyzer, each cycle produces 4 images, one for each of the nucleotides; consequently, the set S consists of the set of all images in all cycles. By contrast, the ABI technology maps adjacent pairs of nucleotides to a 'color-space' in the unprocessed stage. We refer to compression at this raw level as *a. Trace Compression*. The raw, trace data is processed into base-calls creating a set of fragments (or, reads). This processing may have errors, and a quality value (typically a Phred-like score given by $-\lfloor 10 \log(P_{\text{error}}) \rfloor$) is used to encode the confidence in the base-call. In *b. Fragment Compression*, we define the genomic data S by the set of reads, along with quality values of each base-call. Note that the set of reads all come from the genomic sequence of an individual. In *c. Sequence Level Compression*, we define the set S simply as the diploid genome of the individual.

There has been some recent work on compressing at the sequence level[?,?,?,?]. Brandon and colleagues introduce the important notion of maintaining differences against a genomic reference, and integer codes for storing offsets[?]. However, such sequence compression relies on having the fragments reconciled into a single (or diploid) se-

quence. While populations of entire genomes are available for mitochondria, and other microbial strains sequenced using Sanger reads, current technologies provide the data as small genomic fragments. The analysis of this data is evolving, and researchers demand access to the fragments and use proprietary methods to identify variation, not only small mutations, but also large structural variations[?,?,?,?]. Further, there are several applications (e.g., identifying SNPs, structural variation) of fragment data that do not require the intermediate step of constructing a complete sequence.

Clearly, trace data are the lowest level data, and the most difficult to compress. However, it is typically accessed only by a few expert researchers (if at all), focusing on a smaller subset of fragments. Once the base-calls are made (along with quality values), the trace data is usually discarded.

For these reasons, we focus here on fragment level compression. Note that we share the common idea of compressing with respect to a reference sequence. However, our input data are a collection of potentially overlapping fragments (each, say 100 bps long) annotated with quality values. These lead to different compression needs and algorithms from [?,?] because fragment compression must address the additional redundancy caused by high coverage and quality values. Further, the compression must efficiently encode differences due to normal variation *and* sequencing errors, for the downstream researcher.

Contribution: Our paper makes two contributions, implemented in a tool, SLIMGENE. First, we introduce a set of domain specific loss-less compression schemes that achieve over $40\times$ compression of fragments, outperforming bzip2 by over $6\times$. Including quality values, we show a $5\times$ compression. Unoptimized versions of SLIMGENE run at comparable speeds to bzip2. Second, given the discrepancy between the compression factor obtained with and without quality values, we initiate the study of using 'lossy' quality values and investigate its effect on downstream applications. Specifically, we show that using a lossy quality value quantization results in $14\times$ compression but has minimal impact on SNP calls using the CASAVA software. Less than 1% of the calls are discrepant, and we show that the discrepant SNPs are so close to the threshold of detection, that no miscalls cannot be attributed to lossy compression. While there are dozens of downstream applications and much work needs to be done to ensure that coarsely quantized quality values will be acceptable for users, our paper suggests this is a promising direction for investigation.

2 Data-sets and generic compression techniques

Generic compression techniques: Consider the data as a string over an alphabet Σ. We consider some generic techniques. First, we use a reference string so that we only need to encode the differences from the reference. As each fragment is very small, it is critical to encode the differences carefully. Second, suppose that the letters of $\sigma \in \Sigma$ are distributed according to probability $P(\sigma)$. Then, known compression schemes (Ex: Huffman codes, Arithmetic codes) encode each symbol σ using $\log_2 \frac{1}{p(\sigma)}$ bits, giving an average of $\mathcal{H}(P)$ (entropy of the distribution) bits per symbol, which is optimal for the distribution, and degrades to $\log(|\Sigma|)$ bits in the worst case.

Our goal is to devise an encoding (based on domain specific knowledge) that minimizes the entropy. In the following, we will often use this scheme, describing the suitability of the encoding by providing the entropy values. Also, while it is asymptotically perfect, the exact reduction is achievable only if the probabilities are powers of 2. Therefore, we often resort to techniques that mimic the effect of Huffman codes. Finally, if there are inherent redundancies in data, we can compress by maintaining pointers to the first occurrence of a repeated string. This is efficiently done by tools such as bzip2, and we reuse the tools.

Data formats: Many formats have been proposed for packaging and exporting genomic fragments, including the SAM/BAM format[?,?], and the Illumina Export format[?]. Here, we work with the Illumina Export format, which provides a standard representation of Illumina data. It is a tab delimited format, in which every row corresponds to a read, and different columns provide qualifiers for the read. These include ReadID, CloneID, fragment sequence, and a collection of quality values. In addition, the format also encodes information obtained from aligning the read to a reference, including the chromosome strand, and position of the match. More details on the Export format are provided in the Appendix. The key thing to note is that the fragment sequences, the quality values, and the match to the chromosomes represent about 80% of the data, and we will focus on compressing these. In SLIMGENE , each column is compressed independently, and the resulting data is concatenated.

Data Sets: experimental, and simulated

We consider a data-set of human fragments, obtained using the Illumina Genome Analyzer, and mapped to the reference (NCBI 36.1, Mar. 2006). A total of $1.1B$ reads of length 100 were mapped, representing $35\times$ base-coverage of the haploid genome. We refer to this data-set as GAHUM. The fragments differ from the reference either due to sequencing errors or genetic variation, but we refer to all changes as errors. The number of errors per fragment is distributed roughly exponentially, with a heavy tail, and a mean of 2.3 errors per fragment, as shown below. Because of the heavy tail, we did not attempt to fit the experimental data to a standard distribution.

#Errors(k)	0	1	2	3	4	5	6	7	8	9	≥ 10
Pr(k errors)	0.43	0.2	0.09	0.06	0.04	0.03	0.02	0.02	0.01	0.01	0.09

Simulating coverage: While we show all compression results on GAHUM, the results could vary on other data-sets depending upon the quality of the reads, and the coverage. To examine this dependence, we simulated data-sets with different error-rates, and coverage values. We choose fragments of length 100 at random locations from Chr 20, with a read coverage given by parameter c. To simulate errors, we use a single parameter P_0 as the probability of 0 errors in the fragment. For all $k > 0$, the probability of a fragment having exactly k errors is given by $P_k = \lambda \Pr(k \text{ errors})$ from the distribution above. The parameter λ is adjusted to balance the distribution ($\sum_i P_i = 1$). The simulated data-set with parameters c, P_0 is denoted as GASIM(c, P_0).

3 Compressing fragment sequences

Consider an experiment with sequence coverage $c(\sim 30\times)$, which describes the expected number of times each nucleotide is sequenced. Each fragment is a string of characters of length $L(\simeq 100)$ over the nucleotide alphabet Σ ($\Sigma = \{A, C, G, T, N\}$). The naive requirement is $8c$ bits per nucleotide, which could be reduced to $c \log(|\Sigma|) \simeq 2.32c$ bits with a more efficient encoding. We describe an encoding based on comparison to a reference that all fragments have been mapped to.

The position vector: Assume a *Position* bit vector POS with one position for every possible location of the human genome. We set $\text{POS}[i] = 1$ if at least one fragment maps to position i with up to k errors ($\text{POS}[i] = 0$ otherwise). For illustration, imagine an 8-character reference sequence, ACGTACGC, as depicted in Figure 1. We consider two 4bp fragments, CGTA and TACG, aligned to positions 2 and 4, respectively, with no error. Then, $\text{POS} = [0, 1, 0, 1, 0, 0, 0, 0]$. The bit vector POS would suffice if (a) each fragment matched perfectly (no errors), (b) matches to the forward strand and (c) at most one fragment aligns to a single position (possible if $L > c$). The space needed reduces to 1 bits per nucleotide, (possibly smaller with a compression of POS), independent of coverage c.

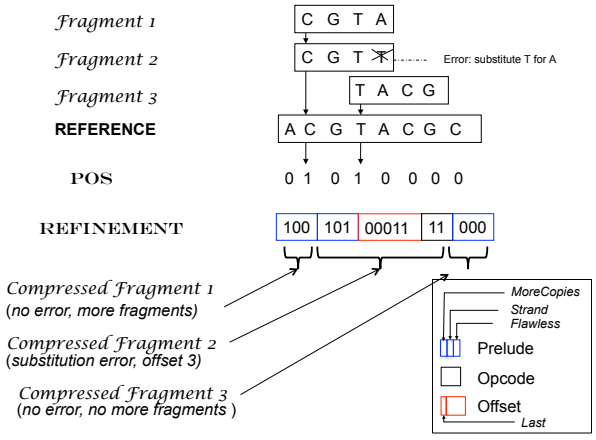

Fig. 1. A simple proposal for fragment proposal starts by mapping the fragments shown on top to a reference sequence. The fragments are encoded by a Position Vector and a Refinement Vector consisting of variable size records representing each compressed fragment. The compressed fragments are encoded on a "pay as needed" basis in which more bits are used to encode fragments that map with more errors

In reality, these assumptions are not true. For example, two Fragments 1 and 2 match at position 2, and Fragment 2 matches with a substitution (Figure 1). We use a *Refinement* vector that adds count and error information. However, the Refinement

vector is designed on a "pay as needed basis" – in other words, fragments that align with fewer errors and fewer repeats need fewer bits to encode.

The refinement vector: The Refinement Vector is a vector of records, one for each fragment, each entry of which consists of a static *Prelude* (3 bits) and an *ErrorInstruction* record, with a variable number of bits for each error in the alignment.

The 3-bit *Prelude* consists of a *MoreCopies*, a *Strand* and a *Flawless* bit. All fragments that align with the same location of the reference genome are placed consecutively in the Refinement Vector and their *MoreCopies* bits share the same value, while the respective bits of consecutive fragments that align to different locations differ. Thus, in a set of fragments, the *MoreCopies* bit changes value when the chromosome location varies. The *Strand* bit is set to 0 if the fragment aligns with the forward strand and 1 otherwise, while the *Flawless* bit indicates whether the fragment aligns perfectly with the reference, in which case there is no following ErrorInstruction.

When indicated by the *Flawless* bit, the *Prelude* is followed by an *ErrorInstruction*, one for every error in the alignment. The ErrorInstruction consists of an *Offset* code (# bp from the last erroneous location), followed by a variable length *Operation Code or OpCode* field describing the type of error.

Opcode: As sequencing errors are mostly nucleotide substitutions, the latter are encoded by using 2 bits, while the overhead of allocating more space to other types of error is negligible. Opcode 00 is reserved for other errors. To describe all substitutions using only 3 possibilities, we use the circular chain $A \to C \to G \to T \to A$. The opcode specifies the distance in chain between the original and substituted nucleotide. For example, an A to C substitution is encoded as 01. Insertions, deletions, and counts of N are encoded using a Huffman-like code, to get an average of $T = 3$ bits for Opcode.

Offset: Clearly, no more than $O = \log_2 L$ bits are needed to encode the offset. To improve upon the $\log_2(100) \simeq 7$ bits per error, note that the quality of base calling worsens in the later cycles of a run. Therefore, we use a *back-to-front* error ordering to exploit this fact, and a Huffman-like code to decrease O.

The record for Fragment 2 (CGTT, Figure 1) provides an example for the error encoding, with a prelude equal to 000 (last fragment that maps to this location and error instructions follow) followed by a single ErrorInstruction. Note that the first bit indicates whether more fragments have been mapped to this offset, the next 4 bits indicate the relative offset of the error from the end of the fragment, and the last 2 bits indicate the error type. Further improvement is possible.

Compact Offset encoding: Let $\mathcal{E}$ denote the expected number of errors per fragment, implying an offset of $\frac{L}{\mathcal{E}}$bp. Instead, we use a single bitmap, ERROR, to mark the positions of all errors from all fragments. Second, we specify the error location for a given fragment as the number of bits we need to skip in ERROR from the start offset of the fragment to reach the error. We expect to see a '1' after $\max\{1, \frac{L}{c\mathcal{E}}\}$ bits in ERROR. Thus, instead of encoding the error offset as $\frac{L}{\mathcal{E}}$ bp, we encode it as the count using

$$O = \log_2 \frac{L/\mathcal{E}}{\max\{1, \frac{L}{c\mathcal{E}}\}} = \min\{\log_2 \frac{L}{\mathcal{E}}, \log_2 c\}$$

bits. For smaller coverage $c < \frac{L}{\mathcal{E}}$, we can gain a few bits in computing O. Overall, the back-to-front ordering, and compact offset encoding leads to $O \simeq 4$ bits.

Compression analysis: Here, we do a back-of-the-envelope calculation of compression gains, mainly to understand bottlenecks. Compression results on real data, and simulations will be shown in Section 3.1. To encode *Refinement*, each fragment contributes a *Prelude* (3 bits), followed by a collection of *Opcodes* (T bits each), and *Offsets* (O bits each). Let $\mathcal{E}$ denote the expected number of errors per fragment, implying a refinement vector length of

$$\mathcal{E} \cdot (3 + T + O)$$

per fragment. Also, encoding POS, and ERROR requires 1 bit each, per nucleotide of the reference. The total number of bits needed per nucleotide of the reference is given by

$$2 + \frac{c}{L} \cdot \mathcal{E} \cdot (3 + O + T) \tag{1}$$

Substituting $T = 3, O = 4, L = 100$, we have

$$\text{c.f.} = \frac{8c}{2 + 0.1c\mathcal{E}} \tag{2}$$

Equation 2 provides a basic calculation of the impact of error-rate and coverage on compressibility using SLIMGENE. For GAHUM, $\mathcal{E} = 2.3$ (Section 2). For high coverages, the c.f. is $\simeq 8/0.23 \simeq 35$. For lower coverages, the fixed costs are more important, but the POS and ERROR bitmaps are very sparse and can be compressed well, by (say) bzip2.

Effectiveness: The reader may wonder how our seemingly ad hoc encoding technique compares to information theoretic bounds. We first did an experiment to evaluate the effectiveness of OpCode assignment. We tabulated the probability of each type of error (all possible substitutions, deletions, and insertions) on our data set and used these probabilities to compute the expected OpCode length using our encoding scheme. We found that the expected OpCode length using our encoding was 2.97 which compares favorably with the entropy which was 1.9.

We also did an experiment to determine the effectiveness of the offset encoding. The width of the error location O depends on the number of bits that we need to skip in ERROR to reach the error location for a given fragment. We computed the error distribution of chromosome 20 of GAHUM and found that the majority of cases involved the skipping of no more than 10 bits in ERROR. Indeed, the entropy of the distribution of error offsets was 3.69. Thus, an initial allocation of 3 or 4 bits (with additional allocation as necessary) is reasonable.

3.1 Experimental results on GASIM(c, P_0)

We tested SLIMGENE on GASIM(c, P_0) to investigate the effect of coverage and errors on compressibility. Recall that for GAHUM, $P_0 = 0.43$, $c = 30$, $\mathcal{E} = 2.3$. As P_0 is varied, $\mathcal{E}$ is approximately $\approx \frac{2.3}{1-0.43} \cdot (1 - P_0) \simeq 4(1 - P_0)$.

In Figure 2a, we fix $P_0 = 0.43$, and test compressibility of GASIM$(c, 0.43)$. As suggested by Eq. 2, the compressibility pf SLIMGENE stabilizes once the coverage is sufficient. Also, using SLIMGENE+bzip2, the compressibility for lower coverage is

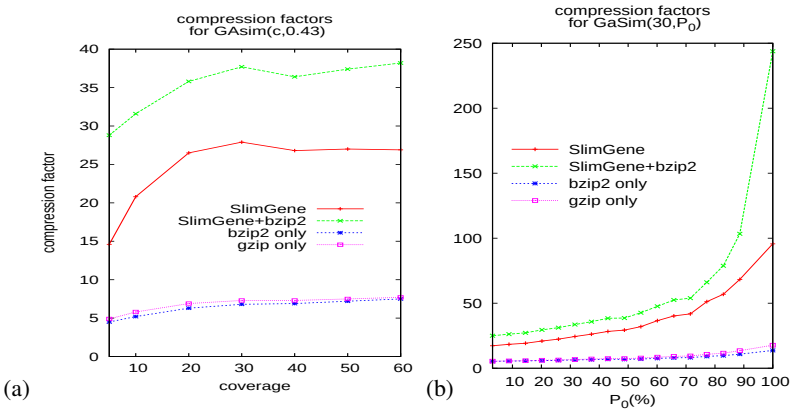

Fig. 2. Compressibility of GASIM(c, P_0). (a) The compression factors achieved with change in coverage. c.f. is lower for lower coverage due to the fixed costs of POS, and ERROR, and stabilizes subsequently. (b) Compressibility as a function of errors. With high values of P_0 (low error), up to 2 orders of magnitude compression is possible. Note that the values of P_0 are multiplied by 100.

enhanced due to better compression of POS and ERROR. Figure 2b explores the dependency on the error rates using GASIM$(30, P_0)$. Again, the experimental results follow the calculations in Eq. 2, which can be rewritten as

$$\frac{8 \cdot 30}{2 + 0.1 \cdot 30 \cdot 4(1 - P_0)} \simeq \frac{20}{1 - P_0}$$

At high values of P_0, SLIMGENE produces 2 orders of magnitude compression. However, it outperforms bzip2 and gzip even for lower values of P_0.

4 Compressing Quality Values

For the Genome analyzer, as well as other technologies, the Quality values are often described as $\approx -\log(P_{\text{err}})$. Specifically, the Phred score is given to be $\lfloor -10 \cdot \log(P_{\text{err}}) \rfloor$. The default encoding for GAHUM require 8 bits to encode each Q-value. We started by testing empirically if there was a non-uniform distribution on these values (see Section 2). The entropy of the Q-values is 4.01. A bzip2 compression of the data-set resulted in 3.8 bits per Q-value. For further compression, we need to use some characteristics of common sequencers.

Position dependent quality: Base calling is highly accurate in the early cycles, while it gradually loses its accuracy in subsequent cycles. Thus, earlier cycles are populated by higher quality values and later cycles by lower values. To exploit this, consider a matrix in which each row corresponds to the Q-values of a single read in order. Each column therefore corresponds (approximately) to the Q-values of all reads in a single cycle. In Figure 3a, we plot the entropy of Q-value distribution at each columns. Not

surprisingly, the entropy is low at the beginning (all values are high), and at the end (all values are low), but increases in the middle, with an average entropy of 3.85.

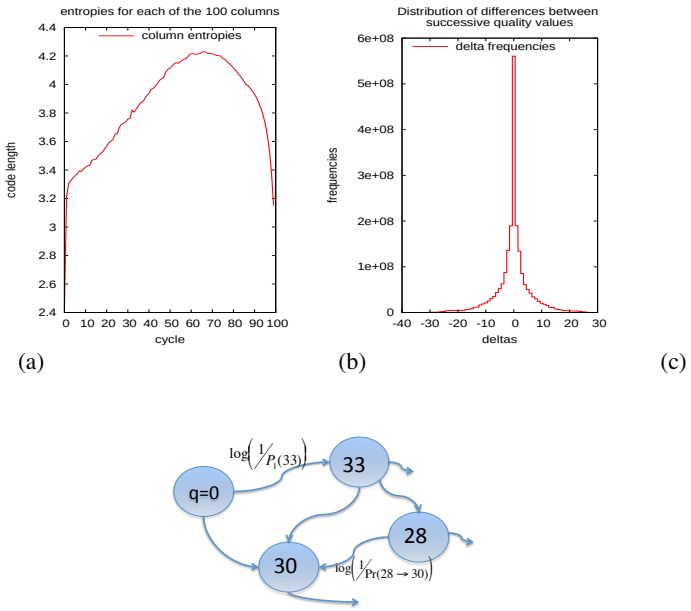

Fig. 3. Distribution of quality, and Δ values, and Markov encoding. (a) A distribution of Q-values at each position. The entropy is low at the beginning (all values are high), and at the end (all values are low), but increases in the middle. (b) A histogram of Δ-values. (c) Markov encoding: Each string of Q-values is described by a unique path in the automaton starting from $q = 0$, and is encoded by concatenating the code for each transition. Huffman encoding bounds the number of required bits by the entropy of the distribution. Edge labels describe the required number of bits for each transition.

Encoding Δ values: The gradual degradation of the Q-values leads to the observation that Q-values that belong to neighboring positions differ slightly. Thus, if instead of encoding the quality values, one encodes their differences between adjacent values (Δ), it is expected that such a representation would be populated by smaller differences. For instance, Figure 3b shows a histogram of the distribution of Δ-values. However, the entropy of the distribution is 4.26 bits per Δ-value.

Markov encoding: We can combine the two ideas above by noting that the Δ-values also have a Markovian property. As a simple example, assume that all Q-values from 2 to 41 are equally abundant in the empirical data. Then, a straightforward encoding would need $\lceil \log_2(41-2+1) \rceil = 6$ bits. However, suppose when we are at quality value

(say) 34 (Figure 3c), the next quality value is always one of $33, 32, 31, 30$. Therefore, instead of encoding Q' using 6 bits, we can encode it using 2 bits, conditioning on the previous Q-value of 34.

We formalize this using a Markov model. Consider an automaton M in which there is a distinct node q for each quality value, and an additional start state $q = 0$. To start with, there is a transition from 0 to q with probability $P_1(q)$. In each subsequent step, M transitions from q to q' with probability $\Pr(q \to q')$. Using an empirical data-set D of quality values, we can estimate the transition probabilities as

$$\Pr(q \to q') = \begin{cases} 0 & (* \text{ if } q' = 0 *) \\ \text{fraction of reads with initial quality } q' & (* \text{ if } q = 0 *) \\ \frac{\#\text{pairs } (q, q') \text{ in } D}{\#\text{occurrences of } q \text{ in } D} & (* \text{ otherwise } *) \end{cases} \tag{3}$$

Assuming a fixed length L for fragments, the Entropy of the Markov distribution is given by

$$\mathcal{H}(M) = \frac{1}{L}\mathcal{H}(P_1) + \frac{L-1}{L} \sum_{q,q' \neq 0} \Pr(q \to q') \log\left(\frac{1}{\Pr(q \to q')}\right) \tag{4}$$

Empirical calculations show the entropy to be 3.3 bits. To match this, we use a custom encoding scheme (denoted as *Markov-encoding*) in which every transition $q \to q'$ is encoded using a Huffman code of $-\log(\Pr(q \to q'))$ bits. Table 1 summarizes the results of Q-value compression. The Markov encoding scheme provides a $2.32\times$ compression, requiring 3.45 bits per character. Further compression using bzip2 does not improve on this.

	Raw File	bzip2	Δ (Huffman)	Markov (Huffman)
Bits per character	8	3.8	4.25	3.45
c.f.	1	2.11	1.88	2.32

Table 1. Quality value compression results.

5 Lossy compression of Quality Values

Certainly, further compression of Quality values remains an intriguing research question. However, even with increasing sophistication, it is likely that Q-value compression will be the bottleneck in fragment-level compression. So we ask the sacrilegious question: *can the quality values be discarded*? Possibly in the future, base-calling will improve to the point that Q-values become largely irrelevant. Unfortunately, the answer today seems to be 'no'. Many downstream applications including alignment, variant calling, and many others consider Q-values as a critical part of inference, and indeed, would not accept fragment data without Q-values. Here, we ask a different question: *is*

the downstream application robust to small changes in Q-values? If so, a 'lossy encoding' could be immaterial to the downstream application.

Denote the number of distinct quality values produced as $|Q| = Q_{\max} - Q_{\min}$, which is encoded using $\log_2(|Q|)$ bits. Note that a Q-score computation such as $\lfloor -10 \cdot \log_2(P_{\text{err}}) \rfloor$ already involves a loss of precision. The error here can be reduced by rounding, or even better, by a 'randomized' rounding, defined as

$$\text{rrand}(x) = \begin{cases} \lceil x \rceil & \text{with probability } x - \lfloor x \rfloor \\ \lfloor x \rfloor & \text{otherwise} \end{cases} \tag{5}$$

Randomized rounding helps to prevent errors in subsequent interpretation. For example, suppose $x = 1.4$ consistently over many experiments. Then, randomized rounding (which rounds x to 1 or 2) ensures that the expected value of $\text{rrand}(x)$ is 1.4. For parameter b, we define the lossy Q-score encoding by

$$\text{LQ-score}_b(Q) = \text{rrand}\left(\frac{\text{Q-score} \cdot 2^b}{|Q|}\right) \tag{6}$$

We encode the 2^b distinct values using Markov-encoding. A downstream application will therefore see $Q_{\min} + |Q| \cdot \text{LQ-score}_b(Q)$ instead of the original value Q.

We test the impact of the lossy scheme on Illumina's downstream application called CASAVA[7] that calls alleles based on fragments and associated Q-scores. CASAVA was run over a 50M wide portion of the Chr 2 of GAHUM using the original Q-values, and it returned a set S of $|S| = 17,021$ variants that matched an internal threshold (the BaCON score exceeded 5, and over 100 for good calls). For each choice of parameter $b \in \{1, \ldots, 5\}$, we reran CASAVA after replacing the original score Q with $Q_{\min} + |Q| \cdot \text{LQ-score}_b(Q)$. Denote each of the resulting variant sets as S_b. A variant $s \in S \cap S_b$ is concordant. It is considered *discrepant* if $s \in (S \setminus S_b) \cup (S_b \setminus S)$.

The results in Figure 4 are surprising. Even with $b = 1$ (using 1 bit to encode Q values), 98.6% of the variant calls in S are concordant. This improves to 99.4% using $b = 3$. Moreover, we observe (in Fig. 4b) that the discrepant SNPs are close to the threshold. Specifically, 85% of the discrepant SNPs have BaCON scores ≤ 10.

5.1 Is the loss-less scheme always better?

We consider the 38 positions in Chr 2 where the lossy (3-bits) compression is discrepant from the loss-less case. On the face of it, this implies a 0.2% error in SNP calling, clearly unacceptable when scaled to the size of the human genome. However, this assumes that the loss-less call is always correct. We show that this is clearly not true by comparing the SNP calls based on lossy and loss-less data in these 38 positions with the corresponding entries, if any, in dbSNP (version 29). We show that most discrepancies come from marginal decisions between homozygote and heterozygote calls.

For simplicity, we only describe the $26/38$ SNPs with single nucleotide substitution in dbSNP. In all discrepant cases, the coverage is no more than 5 reads (despite the fact that the mean coverage is $30\times$). Further, in all but 2 cases, both lossy, and lossless agree with dbSNP, and the main discrepancy is in calling heterozygote versus homozygotes.

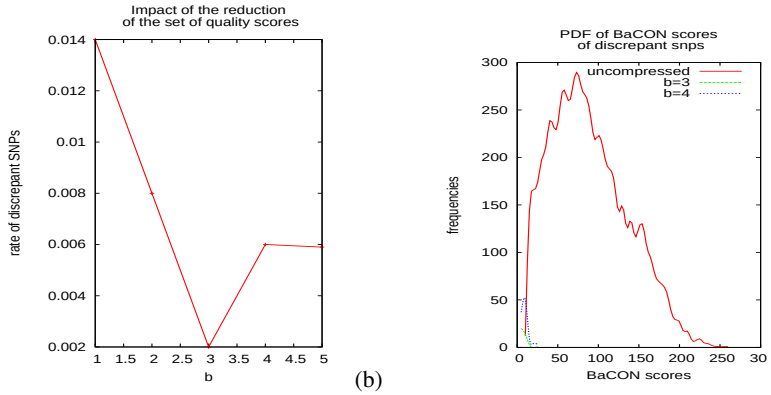

Fig. 4. Impact of Lossy Compression on CASAVA. CASAVA was run on the Chr 2 of GAHUM using original quality values, and after each lossy compression. A variant is discrepant if it was not predicted by CASAVA both before and after lossy compression. The y-axis plots the fraction of discrepant variants as a function of lossy compression. The x-axis describes lossy compression according to the number of bits, b. (b) The BaCON score distribution of the uncompressed, and discrepant variants, for 3, and 4-bit quantization. The plot indicates that that the discrepant variants are close to the threshold of detection (BaCON score =6).

Specifically, lossy calls 14/10 homozygotes and heterozygotes, against lossless (12/12). With coverage ≤ 5, the distinction between homozygote and heterozygotes is hard to make. Close to 50% of the differences were due to consideration of extra alleles due to lossy compression, while in the remaining, alleles are discarded. Given those numbers, it is totally unclear that our lossy compression scheme yields *worse* results than the lossless set, not to mention that in some cases it can lead to better results.

We next consider the two positions where the discrepant SNPs produced by the lossy scheme completely disagree with the dbSNP call. See Table 2. At position 43150343 (dbSNP:C/T), the loss-less allele calls (and Q-values) were $39G, 28G, 20G, 30G$, and CASAVA did not make a call. The lossy reconstruction led to values $41G, 27G, 22G, 32G$, which pushed the net allele quality marginally over the threshold, and led to the CASAVA call of 'G'. In this case, the lossy reconstruction is quite reasonable, and there is no way to conclude that an error was made in the first place. The second discrepant case tells an identical story.

Given the inherent errors in SNP calling (lossy *or* lossless), we suggest that the applications of these SNP calls are inherently robust to errors. The downstream applications are usually one of two types. In the first case, the genotype of the individual is important in determing correlations with a phenotype. In such cases, small coverage of an important SNP must always be validated by targeted sequencing. In the second case, the SNP calls are used to determine allelic frequencies and SNP discovery in a population. In such cases, marginally increasing the population size will correct errors in individual SNP calls (especially ones due to low coverage). Our results suggest that we can tremendously reduce storage while not impacting downstream applications by coarsely quantizing quality values.

position	dbSNP entry	scheme	Qvalues				allele quality	Decision
43150343	C/T	lossy-8	41G	27G	22G	32G	10.2	G
		lossless	39G	28G	20G	30G	9.9	-
46014280	A/G	lossy-8	27C	37C	37C		10.1	C
		lossless	27C	36C	36C		9.9	-

Table 2. Case of wrongly called alleles. In both cases the lossy quality values result in a score which marginally exceeds the threshold of 10 where the allele is called.

6 Putting it all together: compression results

We used SLIMGENE to compress the GAHUM data-set with $1.1B$ reads, a total size of 285Gb. We focus on the columns containing the reads, their chromosome locations and match descriptors (124.7Gb), and the column containing Q-values (103.4Gb), for a total size of 228.1Gb. The results are presented in Table **??** and show a $40\times$ compression of fragments. Using a lossy 1-bit encoding of Q-values results in a net compression of $14\times$ ($8\times$ with a 3-bit encoding). While space restriction preclude a detailed comparison with other data representation formats like SAM/BAM, we report that the BAM representation of GAHUM results in a $3\times$ compression of the dataset.

	fragments+ alignment(Gb)	Q-values (Gb)	total (Gb)	execution time(hr)
Uncompressed	124.7	103.4	228.1	N/A
gzip (in isolation)	15.83	49.92	65.75	N/A
bzip2 (in isolation)	17.9	46.49	64.39	10.79
SLIMGENE	3.2	42.23	45.43	7.38
SLIMGENE+bzip2	3.04	42.34	45.38	7.38
SLIMGENE+lossy Q-values($b = 3$)	3.2	26	29.8	7.38
SLIMGENE+lossy Q-values($b = 1$)	3.2	13.5	16.7	7.38

Table 3. Compression of GAHUM **using** SLIMGENE. Using a loss-less Q-value compression, we reduce the size by $5\times$. A lossy Q-value quantization results in a further $3\times$ compression, with minimal effect on downstream applications.

7 Discussion

The SLIMGENE toolkit described here is available on request from the authors. While we have obtained compression factors of 35 or more for fragment compression, we believe we could do somewhat better and get closer to information theoretic limits. Currently, error-encoding is the bottleneck, and we do not distinguish between sequencing errors and genetic variation. By storing multiple (even synthetic) references, common genetic variation can be captured by exact matches instead of error-encoding. To do this, we only have to increase the POS vector while greatly reducing the number of

ErrorInstructions. This trade-off between extra storage at the compressor/decompressor versus reduced transmission can be explored further.

While this paper has focused on fragment compression as opposed to sequence compression (Brandon et al.[?]), we believe both forms of compression are important, and in fact, complementary. In the *future*, if individuals have complete diploid genome sequences available as part of their personal health records, the focus will shift to sequence-level compression. It seems likely that fragment level compression will continue to be important to advance knowledge of human genetic variation, and is the pressing problem faced by researchers *today*. We note that Brandon et al.[?] also mention fragment compression briefly, but describe no techniques.

While we have shown 2-3$\times$ compression of quality values, we believe it is unlikely this can be improved further. It is barely conceivable that unsuspected relations exist, which allow us to predict Q-values at some positions using Q-values from other positions; this can then be exploited for additional compression. However, there is nothing in the physics of the sequencing process that suggests such complicated correlations exist. Further, it would be computationally hard to search for such relations.

If compressing quality values beyond $3\times$ is indeed infeasible, then lossy compression is the only alternative for order of magnitude reductions. Our results suggest that the loss is not significant for interpretation. However, we have only scratched the surface. Using *companding* (from Pulse Code Modulation[?]), we plan to deviate from uniform quantization, and focus on wider quantization spacings for the middle quality values and smaller spacing for very high and very low quantization values. Further, we need to investigate the effect of quantization on other analysis programs for say *de novo* assembly, structural variation, and CNV detection. The number of quantization values in SLIMGENE is parametrized, and so different application programs can choose the level of quantization for their needs. A more intriguing idea is to use multi-level encoding as has been suggested for video[?]; initially, coarsely quantized quality values are transmitted, and the analysis program only requests finely quantized values if needed.

As sequencing of individuals becomes commoditized, its production will shift from large sequencing centers to small, distributed laboratories. Further, analysis is also likely be distributed among specialists who focus on specific aspects of human biology. Our paper initiates a study of fragment compression, both loss-less and lossy, which should reduce the effort of distributing and synthesizing this vast genomic resource.

8 Acknowledgements

VB and CK were supported by grants from the NSF (-III #0810905), and NIH (HG004962).