Supplementary material for *The Human Genome Contracts Again*

Dmitri Pavlichin\(^1\), Tsachy Weissman\(^2\), and Golan Yona\(^2\)

\(^1\)Department of Physics, Stanford University, CA 94305.
\(^2\)Department of Electrical Engineering, Stanford University, CA 94305

In this document we explain in detail elements of our compression algorithm which are presented in the paper, such as entropy-based coding and the use of tag SNPs. We start with a brief summary of related studies (this section, and the references therein, were omitted from the main manuscript due to space limitation).

## 1 Related studies

Genome compression has been the subject of multiple studies in the past several years. In general, studies in this field can be grouped into two main categories: reference-based compression and non-reference based compression. Methods which do not use reference genomes exploit the repetitive nature of genomic sequences, and compress the input data by using variations over general-purpose compression schemes, such as LZ77 [Lempel & Ziv 1977], which are modified to account for features specific to genomic data, such as inversions, palindromes, and approximate repeats. There are many studies in this category, including [Grumbach & Tahi 1994, Powell et al 1998, Gusev et al. 1999, Apostolico & Lonardi 2000, Matsumoto et al. 2000, Adjeroh et al. 2002, Chen et al. 2002, Tabus et al. 2003, Manzini & Rastero 2004, Behzadi & Le Fessant 2005, Korodi & Tabus 2005, Srinivasa et al. 2006, Cao et al. 2007, Korodi & Tabus 2007, Pinho et al. 2011, Kreft & Navarro 2011]. For a review, see [Giancarlo et al. 2009]. The best compression obtained with such approaches results in about 600MB per genome (see [Pinho et al. 2011] for comparison of several methods).

Methods in the second category (reference-based) obtain much better compression by utilizing the fact that the genomes of different individuals are more than 99.8% identical and coding just the differences between the input genome and a reference genome. Studies in this category include [Christley et al. 2009, Brandon et al. 2009, Deorowics & Grabowski 2011b, Kuruppu et al. 2011, Wang & Zhang 2011, Hsi-Yang Fritz et al. 2011, Pinho et al. 2012, Chern et al. 2012]. The different methods differ in the way the reference sequence is selected or computed from a set of genomes, and the way the difference map is computed and compressed (usually using LZ-inspired schemes). Reference-based methods can reportedly reduce the size of the compressed genome to 3.1MB-19MB.

In general, better results may be obtained when multiple sequences are compressed simultaneously, or when multiple reference genomes are used (note that two tasks are closely related), as discussed in [Mantaci et al. 2005, Mäkinen et al. 2010, Kuruppu et al. 2011, Deorowics & Grabowski 2011b, Kuruppu et al. 2012]. The best results we are aware of among schemes that simultaneously compress multiple genomes are reported in [Deorowics & Grabowski 2011b], who compress 70 complete human genomes to an average size of 3.1 MB per genome. The best compression using a single reference genome thus far was reported in [Christley et al. 2009], who compressed James Watson’s genome to 4MB, by using dbSNP to represent more efficiently known SNPs in the difference map.

Another class of relevant studies focus on compression of short reads [Tembe et al. 2010, Deorowics & Grabowski 2011a, Hsi-Yang Fritz et al. 2011]. One of these schemes [Hsi-Yang Fritz et al. 2011] uses a reference genome against
which they align the reads. It is difficult to directly compare these studies to other genome compression algorithms, since they use a different initial file format (such reads are usually provided in FASTQ format). Moreover, they are usually not applied to full genomes, and in some cases they are applied to a collection of reads from multiple genomes.

2 Entropy coding of distance distributions

The entropy codes we used throughout the paper are either a Huffman code based on a power-law distribution fitted to an empirical distribution (frequency table), a Huffman code based on an empirical frequency table (without fitting this table to a power-law), or an arithmetic code based on an empirical frequency table. The first type, which results in major savings in SNPs representation, is described in detail next.

2.1 The distance distribution

Given a set of samples that are distributed according to a certain distribution, an efficient code utilizes the source distribution to assign binary code words for each sample. That is, shorter code words are assigned to more frequent samples. If the source distribution is unknown, then it can be approximated by the empirical distribution, and code words can be generated based on the frequency table.

The empirical SNP-to-SNP distance distribution for the 3.3M SNPs in James Watson’s genome is plotted in Figure 1. Interestingly, this distribution follows a double power-law distribution, of the functional form

\[ p(d) \sim \left( d^{\alpha} + \left( \frac{d}{d_0} \right)^{\beta/\gamma} \right)^{\gamma} \]

Where \( d \) is the distance in base pairs (bp) between consecutive SNPs, the parameters \( \alpha \) and \( \beta \) give the slope to the left and right of the kink, \( d_0 \) is the position in bp of the kink, and \( \gamma \) controls the curvature of the function near the kink (for \( \beta < \alpha < 0, \gamma < 0 \)).

Figure 1: The SNP-to-SNP distance distribution for James Watson’s 3.3M SNPs. Circles denote the empirical distribution and the solid line corresponds to the fit per equation 1. Dashed lines indicate limiting behavior of the fit to the left and right of the kink in the double power-law distribution. Parameter values in the notation of equation 1 are \( \alpha \approx -0.25, \beta \approx -3.2, d_0 \approx 780, \gamma \approx -2.7 \).
By using this functional form we can derive the code word for each distance value, based on its probability. Storing the coefficients of the functional form is much more efficient than storing the frequency table, and thus results in significant saving in the size of the compressed file.

2.2 Code construction

Instead of coding separately the locations of the three types of variations (novel SNPs, deletions and insertions) between the input genome and the reference genome, we merge them together into a single distance distribution and utilize entropy-based coding of the distances between adjacent variations. The combined distribution results in shorter distances between variations, and therefore smaller code overall. Figure 2 shows the empirical distribution of distances between 844K consecutive changes in Watson’s genome from the reference genome (624K novel SNPs, 157K deletions, 64K insertions merged into one vector, as described in section 2.2.1 of the paper).

This empirical distribution also follows a double power-law distribution as in equation 1, where $\alpha \approx -0.66$, $\beta \approx -4.0$, $d_0 \approx 2500$, and $\gamma \approx -1.6$. We do not provide confidence bounds or mean square error; since our goal is compression, the criterion of interest is the Kullback-Leibler divergence between the empirical and fit distributions (see next section).

To use the distribution $p(d)$ to encode our list of distances, we must form a code table that maps distances (positive integers) to codewords (binary strings). To keep our code table size manageable, we only encode distances smaller than some integer - only 276 of the 844K distances are larger than $10^5$, so we use at most $276 \times 2$ bytes to encode these exceptions separately as int32. We then construct a Shannon-Fano code [Cover & Thomas 2006] for the distribution $p(d)$ restricted to values of $d$ from 1 to $10^5$:

\[
\begin{align*}
1 & \rightarrow 000000 \\
2 & \rightarrow 0000010 \\
& \vdots \\
4 \cdot 10^4 & \rightarrow 111111110110001111001 \\
& \vdots \\
10^5 & \rightarrow 1111111111111111111111111111
\end{align*}
\]
2.3 Performance

Let $\hat{p}(d)$ be the empirical distance distribution (blue crosses in Figure 2). The expected number of bits used to encode a distance value for an entropy code that uses the fitted distribution $p(d)$ is

$$H(\hat{p}) + D(\hat{p}||p)$$

where $H(p)$ is the Shannon entropy of distribution $p$ and $D(\hat{p}||p) \geq 0$ is the Kullback-Leibler divergence between distributions $\hat{p}$ and $p$ (for proof see [Cover & Thomas 2006]). We can never incur an average cost below $H(p)$ if we assume our distances are i.i.d.

In our case $H(\hat{p}) \approx 12.65$ bits and $D(\hat{p}||p) \approx 0.06$ bits, so we are incurring a penalty of $0.06/12.65 \approx 0.5\%$ by using an entropy code corresponding to a double power law $p(d)$ as in equation (1), rather than the empirical distribution\(^1\). In practice, the average number of bits for the actual code may deviate by one bit from the theoretical result of equation 2 (see [Cover & Thomas 2006]). Indeed, the particular code we construct uses 12.73 bits per distance value (1.28MB total), which is 0.7\% above the minimum $H(\hat{p})$. This 0.7\% penalty for $p(d)$ amounts to 8.7KB total for all distances. On the other hand, it saves us a larger penalty of $\sim 0.15$MB if the list of unique distances is coded using the empirical distribution $\hat{p}(d)$, since we need not save the empirical frequencies for each distance, but only the coefficients of the double power law distribution.

It should be noted that our code table is larger than it needs to be: it has $10^5$ entries, but there are only $3 \cdot 10^4$ unique distances (for example, the value $4 \cdot 10^4$ shown above never appears in our distance list). This only affects performance in that a larger-than-necessary code table is temporarily constructed during encoding and decoding, but it does not affect the file size of the coded genome.

2.4 Entropy coding of entries in dbSNP

We find it profitable to repeat this procedure for the distances between indices in dbSNP of Watson’s 2.7M non-novel SNPs. Figure 3 mirrors Figure 2, but with the distribution of index-distances as opposed to the physical distances. Our entropy code costs 2.77 bits per distance value, 1.5\% more than the empirical entropy 2.74 bits, for a total cost of 0.89MB, but saves us the need to store the frequency table. If we use a Golomb code (geometric $p(d)$) for the distances, or an arithmetic code to encode the binary vector with one entry per row in dbSNP, the total cost would be 1.00MB.

3 Using tag SNPs

We can exploit the co-occurrence of SNPs in haplotype blocks by storing a binary vector with one entry per tag SNP, rather than one entry for each SNP in dbSNP. Ideally, the presence of a tag SNP means the presence of a particular allele, such that the tag SNP completely predicts the presence or absence of all other non-novel SNPs within the same block. In practice, this is not always the case, and we need to store a list of mispredictions whenever the tag SNP appears in the input genome but the other SNPs in the same block do not conform to those documented in the literature. So long as a tag SNP is informative at all about the other SNPs within the same block, the list of mispredictions will be compressible by an arithmetic code, since the fraction of 0s and 1s will be unequal.

\(^1\)In contrast, when using a geometric distribution to fit the empirical distribution of Figure 2, the entropy code (which corresponds to a Golomb code in that case) would result in a penalty of 4.2\%. Other entropy codes, such as Golomb-Rice and Elias gamma, are tailored to generic non-data-dependent distributions, and as such they are outperformed by the empirical code.
For the purpose of this work we used the Illumina Bead Array HumanMap300k data set [Gunderson et al. 2005], with the tag SNPs determined by [Pe’er et al. 2006]. The process is illustrated in Figure 4. For example, consider the first line of this dataset in marker 1, chromosome 1:

```
rs1000313:A:0.225  rs12022554:A:0.0333:0.119  rs1000313:A:0.225:1.000  rs4501834:G:0.2167:0.953
rs10803349:T:0.2167:0.953  rs10927672:A:0.2167:0.953
```

This specific block contains 5 SNPs, labeled rs12022554, rs1000313, rs4501834, rs10803349, and rs10927672. The tag SNP is always listed first, and is repeated inside the block according to its order within the block. The dbSNP labels provide the positions of the SNPs along the genome. Each non-tag SNP is listed with its frequency and $r^2$ correlation value. In this example, the tag SNP label is rs1000313, having nucleotide A in that position with frequency 0.225. This tag SNP predicts the remaining alleles A,A,G,T,A with correlation values 0.119, 1, 0.953, 0.953, 0.953 (the second one is the tag SNP and therefore the correlation is 1). And while the first non-tag SNP is not highly correlated with the tag SNP the other three are, and are therefore amenable for compression. Finally, we look up the positions of these SNPs in dbSNP to parse the line into:

```
Tag SNP is A at position 15278076 on chromosome 1 predicts (A, G, T, A) at positions (15211525, 15281743, 15284856, 15285508) on chromosome 1
```

It should be noted that the Illumina dataset is partial, and includes only 2.46M of the 9.6M SNPs in dbSNP (and uses an earlier dbSNP build, 122, than the one we use to compress SNP positions, 129). Of the 2.46M SNPs in the Illumina dataset, 260,109 are biallele tag SNPs that predict at least one other non-tag SNP (43,244 tag SNPs, or 14% of all tag SNPs, predict no other SNPs). The CEU dataset contains only biallele SNPs. These tag SNPs predict 2,161,498 non-tag SNPs (8.3 predicted SNPs per tag SNP on average).

Watson’s genome matched 183,089, or 70% of the tag SNPs. These tag SNPs predict a total of 1,561,717 non-tag SNPs. Watson’s genome was correctly predicted at 1,372,356, or 88% of these.

Without tag SNPs, we encoded the 9.61M SNPs in dbSNP using 920KB. Now we no longer need to encode

\[\text{Figure 3: Observed distribution of distances between indices of Watson’s SNPs in dbSNP. Blue crosses: histogram of distances, bin width 1. Red circles: smoothed histogram, log-spaced bins. Solid black line: double power law fit (minimum mean square error).}\]
the 1.56M predicted non-tag SNP, but we need to encode the list of mispredictions. Specifically, when using a tag SNP dataset, there are two objects to store: First, a list with one entry per tag SNP where a 1 (0) indicates a (mis)match with Watson’s genome. The compressed size of this list is

\[ \text{[260K tag SNPs]} \times \text{[H(70% match) bits / tag SNP]} = 28 \text{ KB} \]

Second, a list with one entry for every SNP predicted by some tag SNP, where a 1 (0) indicates a correct (incorrect) prediction. The compressed size of this list is

\[ \text{[1.56M predicted SNPs]} \times \text{[H(12% incorrect predictions) bits / predicted SNP]} = 101 \text{ KB} \]

We can exclude the Illumina SNPs (260K tag SNPs + 1.56M predicted SNPs) from our description of the 9.61M SNPs in dbSNP, for a savings of

\[ \frac{(260K + 1.56M)}{9.61M} \times \text{[cost of encoding dbSNP]} = 175 \text{ KB} \]

The total savings due to using the Illumina set is then

\[ 175KB - 28KB - 101KB = 46KB \]

and while the savings are small, they are expected to improve as more information on tag SNPs is accumulated to include a larger portion of the SNPs in dbSNP, and if the percentage of correctly predicted SNPs in the input genome is higher. Frequency and r2 values could also be used to improve the efficiency of the misprediction encoder.

4 Data structures

4.1 Uncompressed difference map

An uncompressed difference map is a list of variations between a reference genome and a genome to be compressed with respect to the reference, where each entry includes the variation’s type (SNP, deletion, insertion, or indel), position in the genome, and variation sequence (e.g., for an insertion, the inserted sequence). To allow direct comparison of our methods and the algorithm of [Christley et al. 2009] we follow their uncompressed format:

\[ \text{[variation type]},\text{[chromosome name]},\text{[starting position in chromosome]},\text{[reference sequence]} / \text{[alternate sequence]} \]

For example, the first SNP, deletion, and insertion in Watson’s genome with respect to hg18 are represented as follows, respectively:

- 0,chr1,108,C/T
- 1,chr1,713667,GA/–
- 2,chr1,715891,—–/ATGGA

and the list of variations is sorted by position within each chromosome.

For the 1000 genomes data, which additionally contains allele-specific information on variations, the difference map is augmented by including this information: 11 means homozygous variation, and 01 or 10 mean heterozygous variation. For example, the first SNP of HG00096 with respect to hg19 is represented as follows:
• 0,chr1,30923,G/T,11

where 11 means that HG00096 is homozygous for this SNP.

### 4.2 Compressed difference map

As described in the main text, our approach separates the variations found in an external database (dbSNP) from the novel variations. There are two uncompressed objects:

1. A binary list with as many entries as the external database (dbSNP), where 1 indicates that the variation appears in the genome to be compressed, and 0 otherwise.
2. A list of novel variations, stored in the same format as described above in Section 4.1.

#### 4.2.1 dbSNP representation

To compress the binary list we first convert it into a list of positive integers indicating the distance between consecutive 1’s, and then compress this integer list via Huffman coding. As described in the main text, we do not store the frequency or codeword list for this Huffman code, but only the four parameters in equation (1). The Huffman codeword list is generated each time we compress or uncompress, using these parameters.

It should be noted that there may be SNPs present in the genome to be compressed whose position is in dbSNP, but whose allele does not match either the alternate or reference allele in dbSNP. We refer to such SNPs as a partial match (as opposed to a full match, where both the position and allele match). There were only 4902 such partial matches for Watson’s genome. Since most of the space to store a SNP is used to store its position, we reduced file size by storing these partial matches in a binary string of length equal to the total number of full and partial dbSNP matches (1 means partial match, 0 full match), compressing this string using arithmetic coding, and storing each mismatched allele as if it were the allele of a novel (non-dbSNP) SNP, described in the following section.

#### 4.2.2 Novel variations representation

For the novel variations (those not documented in an external database), the uncompressed difference map format described in Section 4.1 is converted into the following objects:

1. A list of positive integers indicating the distance between consecutive variations.
2. A list of variation types: we use 0,1,2,3 to indicate SNP, deletion, insertion, indel, respectively.
3. A list of novel SNP nucleotides. To exploit the correlations between reference and alternate SNP values, this list is partitioned into four lists, one for each reference nucleotide (e.g. one of these four lists is the set of all novel SNP nucleotides that replace an ‘A’ in the reference genome).
4. A list of deletion lengths.
5. A list of insertion lengths.
6. A list of all insertion sequences concatenated into one sequence.
7. For 1000 genomes data: A list of the allele indicators (11, 01, or 10).
These objects are represented in compressed form by:

1. Huffman coding (binary file) with respect to the probability mass function given in (1). The four numerical parameters of this distribution are stored. To reduce memory usage during compression and decompression, we separately store a list of exceptionally large distances (greater than 100,000) between novel variations.

2. Arithmetic coding (binary file). The frequencies of the four variation types are stored.

3. Arithmetic coding (binary file). As described in the main text, a separate arithmetic code is used for each of the 4 reference nucleotides. The frequency table of alternate and reference nucleotides (4 by 4) is stored. A separate binary file is used to store the binary output of each of the 4 arithmetic codes.

4. Huffman coding (binary file). A codeword table for this code is stored.

5. Huffman coding (binary file). A codeword table for this code is stored.

6. Arithmetic coding (binary file). The frequencies of the four variation types are stored.

7. For 1000 genomes data: Huffman coding (binary file). We fit a third-order Markov chain to the sequence of allele indicators, store the transition probability table for this Markov chain, and use this table to construct a Huffman code.

These compressed objects and the list of variations in an external database are sufficient to reconstruct the uncompressed difference map.

Note that, for the allele indicator sequence for the 1000 genomes data, we found that the entries are not independent (e.g. being homozygous for one variation is predictive of being homozygous for nearby variations) and we exploit this dependence by fitting a Markov chain to this sequence. A third-order Markov chain was used to minimize the compressed file size plus the size of the stored transition probability table for the chain.

5 Running times and memory usage

Our software is available both as a Matlab code and as a C++ code. Matlab code was executed using version 7.7.0471 of Matlab. The nlinfit() function, used to fit the double power-law distribution to the empirical distributions, is included in the Statistics Toolbox. The C++ program runs faster than the Matlab code, but requires estimating the parameters of the double power-law distributions prior to running the program.

As far as running times and memory usage, the only direct comparison we can make is to the software by [Christley et al. 2009], since this is the only other algorithm which uses the same input data. This software takes about 35 seconds and 32MB of memory to compress Watson’s genome on a Sun Fire X2200 with 2.3Ghz processors. It takes 70 seconds and 260MB of memory to uncompress the genome. Our software uses about twice as much time and memory (105 seconds and 180 MB memory to compress, 130 seconds and 500 MB memory to uncompress). To reduce the memory usage, the compression is done chromosome by chromosome.

It is harder to compare our method to other reference-based studies [Deorowics & Grabowski 2011b, Kuruppu et al. 2011, Wang & Zhang 2011, Hsi-Yang Fritz et al. 2011, Pinho et al. 2012], since they perform a different task from

---

3 The number of large distances is small (1437 in Watson’s genome, with a maximum distance of 20,317,007), but if we did not store them separately from the other novel distances, we would need to construct a Huffman table that uses about 200 times as much memory during compression and decompression in order to include large enough distance values.
ours. Other studies usually use an LZ-inspired scheme that constructs a difference map, whereas we spend most of our time coding an existing difference map. Some of these studies report running times around 5min or more and about 10GB of memory. However, the numbers quoted were for different datasets (e.g. 70 human genomes, or a single Korean genome) and all were produced on different machines. For comparison of several LZ-inspired schemes see Table 5 in [Deorowics & Grabowski 2011b].

References


Figure 4: Using tag SNPs to predict variations.