Genome Compression Against a Reference

Aniruddha Laud Gaurav Menghani
Madhava Keralapura

Stony Brook University

December 7, 2011

.

Abstract

Being able to store and transmit human genome sequences is an important part in genomic research and industrial applications. The complete human genome has 3.1 billion base pairs (haploid), and storing the entire genome naively takes about 3 GB, which is infeasible for large scale usage.

However, human genomes are highly redundant. Any given individual's genome would differ from another individual's genome by less than 1%. There are tools like DNAZip, which express a given genome sequence by only noting down the differences between the given sequence and a reference genome sequence. This allows losslessly compressing the given genome to $\approx$ 4 MB in size.

In this work, we demonstrate additional improvements on top of the DNAZip library, where we show an additional $\approx$ 11% compression on top of DNAZip's already impressive results. This would allow further savings in disk space and network costs for transmitting human genome sequences.

Contents

1 Introduction

Genomes are usually distributed in two formats. Firstly, using the FASTA format. Secondly, and more commonly as two files, one containing the SNPs (Single Nucleotide Polymorphisms), and the other containing insertions and deletions, both with respect to a reference genome (usually hg18 [2, 3]).

While the size of a pure FASTA-based genome is about 3 GB, using the second format as mentioned above, James Watson's genome was expressed in about 1.8 GB [7], with reference to the hg18 genome.

The DNAZip project [4] makes use of the database of all SNPs reported (dbSNP) and the hg18 genome to represent this information cleverly in about 4.1 MB. We intend to represent this information in an even more efficient format, and reduce the size of the genome further, without using any extra information.

2 Relevant Work

The following techniques are used by DNAzip to achieve compression.

1. Variable size integer representation and delta positions:

   In the variation data, all the variation (indels and SNPs) are expressed as position on the reference genome, plus the variation actually present. The position is expressed as an offset from the beginning of the sequence. Thus, the size required to store the position increases while storing variations at the end of the genome. Instead of storing the entire offset, it is sufficient to store just the delta from the previous variation. One problem with this is that an integer value takes the same amount of space to store, regardless of the value within it.

   Thus, by implementing a mechanism to store exactly as many bits are required, storage utilized could be optimized. A variable integer as implemented by DNAzip uses 7 of 8 bits in a character to store the bits of the integer, and 1 bit to indicate whether an integer ends at that byte or not.

2. SNP mapping

   Most of the variation between genomes are generally substitutions usually referred to as single nucleotide polymorphisms (SNPs). Most of these substitutions exist in one of two possible alternatives (bi-allele). Most of these SNPs are collected and organized by NCBI in what is known as dbSNP. dbSNP contains all known SNPs organized by position against the human reference genome (hg18, for instance). Though dbSNP records variations other than just the SNPs (it also records indels and multi-allele SNPs), DNAzip just considers the bi-allele SNPs. DNAzip stores the SNPs as a bit vector of all possible SNPs, with 1 where the SNP exists and 0 where the SNP doesn't. For those SNPs that are not present in dbSNP, they are stored as position along with the actual substitution.

3. K-mer partitioning

   A common compression technique used is Huffman coding. By partitioning all the insertion data into K-mers and inserting them into a Huffman tree, the optimal representation of each k-mer can be obtain, and this representation is then stored, thus achieving some level of compression.

3 Work Done

3.1 Storing the Bit Vectors Efficiently

As explained above, DNAZip creates a bitmap from the dbSNP. If bit i is set in the bitmap, it implies that the ith SNP from the dbSNP occurs in the actual genome. The DNAZip project simply stores this sparse bitmap for each chromosome. Upon analysis, we found that these bitmaps compose of about 1.2 MB of the total 4.1 MB of the genome. This sparseness of the bitmap made a strong case for compressing it.

Compressing the bit-vectors using Huffman Encoding with k-mers of size 5 resulted in the genome being of size 3896467 bytes. While using k-mers of size 6 resulted in the genome being of size 3889100 bytes. This is against the original file of size 4198717 bytes. Thus, using k-mers of size 6, we decreased the size of the genome by 302.36 KB, which is a space-saving of 7.37%.

Chromosome	% Compression Achieved
1	26.4089
2	21.779
3	20.9808
4	19.7785
5	24.719
6	25.1927
7	24.6257
8	22.4158
9	32.0532
10	23.3288
11	22.3557
12	22.7921
13	17.8374
14	22.389
15	22.9731
16	28.062
17	26.9674
18	22.1033
19	27.468
20	31.6912
21	19.8773
22	33.8321
M	35.4922
X	49.1172
Y	80.8371

Table 1: Compression achieved for each chromosome

3.2 Improving the Variable Integer (VINT) storage

The DNAZip source code makes heavy use of Variable Integer (VINT) storage, so that they do not have to allocate a fixed number of bits for each integer value that needs to be written. The reason being, if we are writing very small values, and the data type of our choice is the standard integer, we would be using 32 bits for each value that we write but most of the higher bits in the number would not be set. However, if we choose a small data type like a byte, we would not be able to write values greater than $2^8 - 1$.

This is solved by writing 8 bits at a time, of which the MSB is set if there is another block of 8 bits for the number after the current block. Thus, out of the 8 bits, 1 bit is a flag, and the rest 7 come from the number to be written. We realized that this might not be the most efficient way of doing it. A large number of values written as variable integers are small delta values, and writing 8 bits at a time would mean that atleast 8 bits would be written even if the number is very small.

The values being written as VINTs during the compression process were written to an auxilliary file. We then tried to compare the space consumed when the size of the VINT word size is varied.

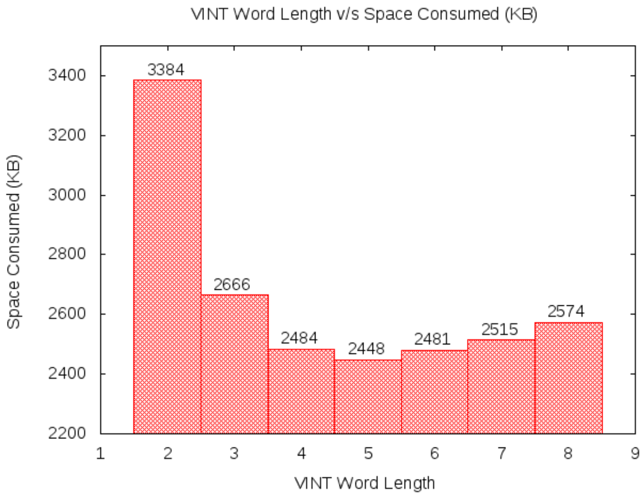

Figure 1: VINT Word Length v/s Space Consumed (KB)

Thus, with the VINT word size as 8, 2574 KB of space was being used up for writing VINTs. The optimal VINT word size was found to be 5, which as per experiments would have used up 2448 KB of space.

The actual compression received from this was 129 KB roughly, which was in line with our expectation before this experiment.

3.3 Improving the storage of Insertions/Deletions (INDELs)

After thorough analysis, we realized that the Insertion/Deletion format in iteself was pretty packed. One optimization possible was to create a Bit Vector for the Insertions and Deletions which were present in dbSNP. This meant that we would not need to store the information for the insertions and deletions which existed in the dbSNP. A bit vector for the insertions / deletions present in dbSNP could be used to encode that information.

The number of deletions in the genome which also existed in dbSNP, was significant but not large enough (Refer Table 2). The percentage of deletions in a chromosome which exist in dbSNP varies from 5.3% to 11.68% (apart from the Mitochondrial DNA, which has only one deletion, which does not exist in dbSNP).

We compressed the bit vector using Run-Length Encoding. This gave us a saving of about 10 KB. Using Huffman Encoding gave us a saving of roughly 20 KB. Another 1 KB of compression was squeezed in by noticing that the bit vector was sparser than the SNP bit vector, and a k-mer length of size 7 was the most suitable for this.

Thus, we only changed the storage for the deletions. The deletions which were present in dbSNP were stored using bitvectors, and compressed using Huffman Encoding with k-mer size as 7.

For insertions we noticed that the number of insertions in the Genome which existed in dbSNP were very few. Thus, the bit vector was extremely sparse. And hence, there was not much benefit of encoding them as bit-vectors.

Chromosome	% of Deletions found in dbSNP
1	8.54
2	8.76
3	8.80
4	11.67
5	10.46
6	8.87
7	6.90
8	9.45
9	7.84
10	9.40
11	10.09
12	6.39
13	6.70
14	9.63
15	6.99
16	6.74
17	6.50
18	10.61
19	5.30
20	13.06
21	10.36
22	7.08
Mitochondrial	0.0
X	7.57

Table 2: Percentage of Deletions in each chromosome found in dbSNP

3.4 Miscellaneous Experiments

1. For SNPs which are not present in dbSNP, DNAZip writes the substituted and the substituted characters as bits to the file. We tried compressing this string with different k-mer sizes but the bit-string was random enough not to yield any decrease in the compression.

2. Further in our efforts to improve the encoding of SNPs which were not present in dbSNP, we tried storing all the substitute characters in order of the character they replace. However, now this increased the delta values, this did not give a better compression. Efforts to tweak the VINT storage did not help either.

4 Results & Conclusion

The original DNAZip format was already pretty efficient. However, over the course of project, we have exploited avenues where we could foresee significant compression possible. Towards the end of the project, it became hard to find any further improvements.

In all we have compressed James Watson's genome to **3736918** bytes, from the original 4198717 bytes used by the DNAZip format. This is a compression of roughly **11%**.

With increase in the size of dbSNP, we expect the compression to become better with time, since majority of the file size, roughly 2 MB out of the current 3.56 MB is in deltas used for encoding SNPs absent from dbSNP. With time this number is expected to go down, and the bit-vector size to increase. Also, the bit-vector gets sparser and the compression is expected to work better.

5 Further Work

1. After experimentation we have made certain choices, such as the k-mer size for compressing the SNP bitvectors. While we expect other genomes to have similar data and our choices to be relevant, but we can optimize this at the cost of performance. Doing one pass to figure out the data to be written, such as the bit-vectors, VINT data, and then trying out all possible choices and making the best possible choice. However, as apparent, it would be roughly more than 2x slower than the current DNAzip build with our code.

2. Of the 2574 KB occupied by VINT storage, roughly 2 MB is used by Deltas stored while encoding SNPs. Using a different VINT storage as discussed earlier we saved about 129 KB. However, we think there should be scope of further optimizing the VINT storage.

It is possible to make the DNAZip utility more useful. A few of them are as follows:

1. Allowing compression and decompression on a chromosome-by-chromosome basis.

2. There is scope for constant-factor improvement in DNAzip's performance, this could be done by using C-style data structures instead of the C++ STL data structures being used.

References

[1] Jorde, Lynn B. and Wooding, Stephen P., *Genetic variation, classification and race*. Nature Genetics 36 (11 Suppl): S28–S33. doi:10.1038/ng1435. PMID 15508000, 2004

[2] UCSC Genome Bioinformatics, Human (Homo sapiens) Genome Browser Gateway http://genome.ucsc.edu/cgi-bin/hgGateway

[3] UCSC Genome Bioinformatics, Human Chromosome (hg18) http://hgdownload.cse.ucsc.edu/goldenPath/hg18/chromosomes/

[4] The DNA Zip Project http://www.ics.uci.edu/ dnazip/

[5] Christley, Scott and Lu, Yiming and Li, Chen and Xie, Xiaohui, *Human genomes as email attachments*.

[6] Chen, Xi and Kwong, Sam and Li, Ming, *A Compression Algorithm for DNA Sequences and Its Applications in Genome Comparison*

[7] James Watson's Sequence http://jimwatsonsequence.cshl.edu/cgi-perl/gbrowse/jwsequence/

[8] Complete Genomics http://www.completegenomics.com/