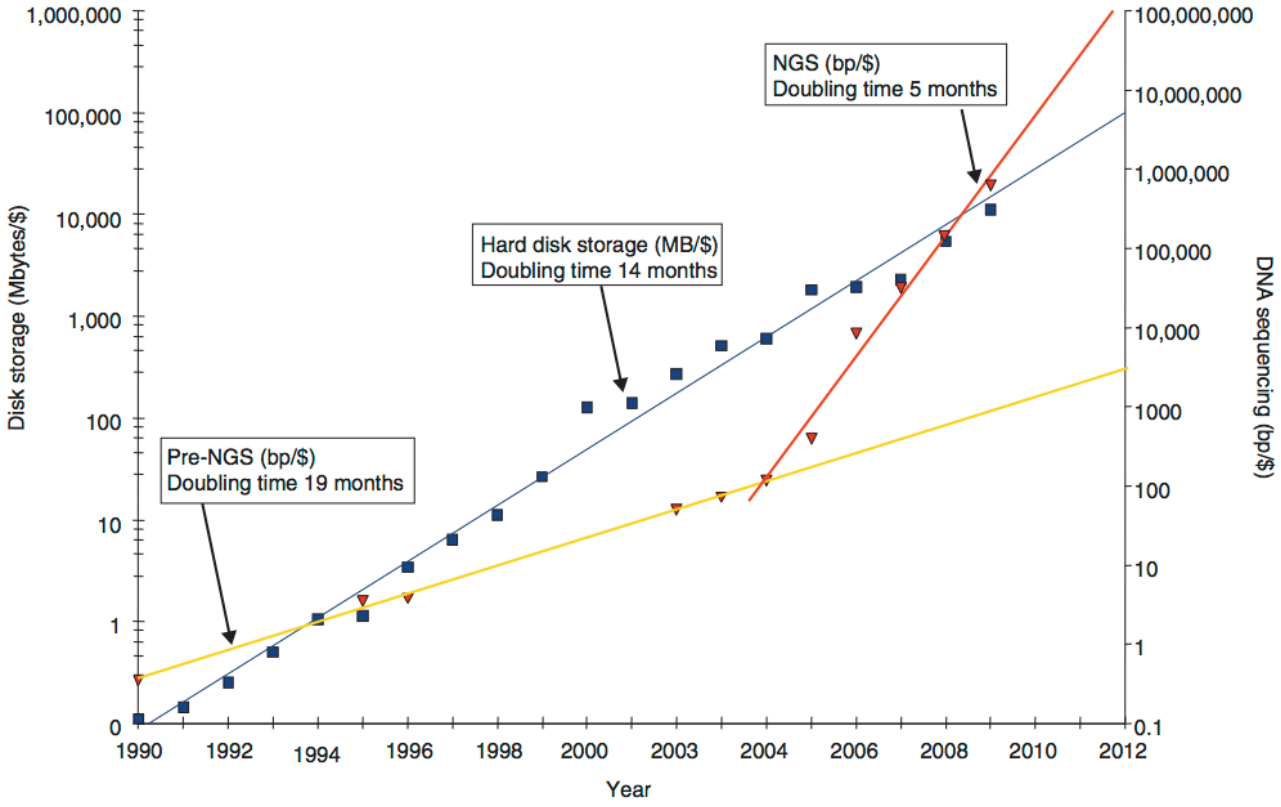
Run-Length and Markov Compression Revisited in DNA Sequences

Saurabh Kadekodi
M.S. Computer Science
saurabhkadekodi@u.northwestern.edu

Efficient and economical storage of DNA sequences has always been an interesting problem to solve. There is no particular solution termed as the "best" solution for this problem although people have tried to attack the problem from various angles. Today, the digital size of an uncompressed human genome can be as large as 285 GB[1]. With this explosion in DNA data, especially since the usage of next generation sequencing methods[2], its compression has become imperative. Below is a graph that summarizes the history of genome data and the evident need for DNA compression.



[Fig 1] - Comparison of disk costs in MB per US dollar to DNA costs in base pair per US dollar.

The blue squares describe the historic cost of disk prices in megabytes per US dollar. The long-term trend (blue line, which is a straight line here because the plot is logarithmic) shows exponential growth in storage per dollar with a doubling time of roughly 1.5 years. The cost of DNA sequencing, expressed in base pairs per dollar, is shown by the red triangles. It follows an exponential curve (yellow line) with a doubling time slightly slower than disk storage until 2004, when next generation sequencing (NGS) causes an inflection in the curve to a doubling time of less than 6 months (red line). These curves are not corrected for inflation or for the 'fully loaded' cost of sequencing and disk storage, which would include personnel costs, depreciation and overhead[2].

Classes of Compression:

Compression algorithms can broadly be classified into 3 categories, compression using run-length encoding techniques, dictionary based compression and Markov compression.

Random data is any compression algorithm's worst enemy. DNA sequences typically consist of 4 characters, viz. A, C, G and T, and their existence in a DNA sequence is close to random. There is very little that any compression algorithm can do in such a situation. Most algorithms today are based on the dictionary based approach, and are interesting extensions to the universal Lempel-Ziv compression algorithm[3]. We believe that it is due to the randomness and unstructured nature of genome sequences, that, till date, the dictionary based models have proven to be the best from the point of view of performance and size.

Splitting the Genome:

An interesting preprocessing technique shows that we can possibly have some order in this chaos making us revisit the run-length encoding schemes and Markov compression as possibly better options than the current compression techniques.

Splitting involves duplicating the DNA sequence into as many copies as the unique characters that are there in DNA, viz. 4. Let's call each copy by the character for which it is intended. For A's copy, we represent all occurrences of the rest of the three characters by a hyphen. In A's copy we now have only 2 characters, A and hyphen. We perform the similar operation on C's copy, G's copy and T's copy. We can now observe that if we superimpose the 4 copies on one another, we get the original sequence back.

Even though this preprocessing technique seems trivial and intuitively seems to make an already unwieldy genome sequence even larger in size (contradicting our original objective), it has some very interesting characteristics.

Run-Length Encoding:

Let us now look at how we can benefit from splitting of the genome from the run-length encoding perspective. To summarize, run-length encoding involves replacing the number of continuous occurrences of a certain entity by a single instance of that entity followed by a number denoting how many times it occurred.

Before mentioning how we can exploit splitting, let us first introduce 2 more concepts, Genome Encoding and Variable Integers (VINT).

Genome Encoding:

As mentioned previously, DNA sequences consist of A, C, G and T. Since there are only 4 characters, we can reduce the 1-byte representation of the characters by only a 2-bit representation, according to the table below. This is one of the most primitive compression methods.

A	00
C	01
G	10
T	11

[Table 1] - Genome Encoding of DNA characters

Variable Integers:

The usual size of an integer is 4 bytes. An integer (of 4 bytes) is thus capable of storing numbers up to 4294967296 in unsigned representation and -2147483648 to 2147483648 in signed representation. Many times, we do not need these many numbers and we end up wasting a lot of space, just to store 0's. To use integers more efficiently, we have the concept of Variable Integers. In VINT, we use only as many bytes to represent the number as required. 1 bit from each byte holding the number is reserved as a flag bit to indicate whether the number ends in the current byte or has overflowed into the next byte.

An example of a single byte integer:

Integer 6 in the usual representation:

00000000 00000000 00000000 00000110

Integer 6 in VINT:

00001101

An example of a multiple byte integer:

Integer 1722 in the usual representation:

00000000 00000000 00000011 01011101

Integer 1722 in VINT:

00000110 10111011

VINT reduces the effective length of each byte by 1 bit, but gives us the flexibility to use only as many bytes as needed, especially effective for small number representations.

Proposed Technique:

We propose the following technique for run-length based DNA compression:

Splitting + Genome Encoding + Run-Length Encoding + VINT

For a better explanation for the technique as a whole, let us take an example. In the example, let us do a comparative study of naive run-length encoding with the above mentioned technique.
Following is a sample DNA sequence to be compressed:

G G G G T C C C A A T T C A G T

Naive run-length compression consists of initially genome encoding the sequence before we run-length encode it using normal integers. On genome encoding the above sequence, we get the following bit representation:

1 0 1 0 1 0 1 0 1 1 0 1 0 1 0 1 0 0 1 1 0 1 0 0 1 0 1 1

On performing run-length encoding of the above bit sequence, we get the following:

11 01 11 01 11 01 11 01 12 01 11 01 11 01 11 02 12 01 11 02 11 01 12

We can observe that 1's and 0's (in black) are alternating. So, we can simply omit them (i.e. just make a note of whether to start with 1 or 0, but for the purpose of this discussion, omit them altogether) and just list their run-lengths to get:

1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 2 1 1 2 1 1 2

Considering 4 byte integers, the above sequence would be 92 bytes long.

Let us now work on the same sequence using the technique we introduced above. The first step is to split the original sequence to obtain the following copies:

A's copy: - - - - - - - - A A - - - A - -
C's copy: - - - - - C C C - - - - C - - -
G's copy: G G G G - - - - - - - - - - G -
T's copy: - - - - T T - - - - T - - - - T

To genome encode the sequence, we can now take advantage of the fact that there are only 2 characters per split sequence. We can thus represent the hyphen by 0 and the corresponding character in each sequence by the bit 1. On genome encoding the split sequence according to the modified scheme, we get:

A's copy: 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0
C's copy: 0 0 0 0 0 1 1 1 0 0 0 0 1 0 0 0
G's copy: 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0
T's copy: 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 1

If we run-length encode each of the above sequences, we get:

A's copy: 08 12 03 11 02
C's copy: 05 13 04 11 03
G's copy: 14 010 11
T's copy: 04 12 04 11 04 11

Again, since there are alternating 0's and 1's, we can simply omit them to get:

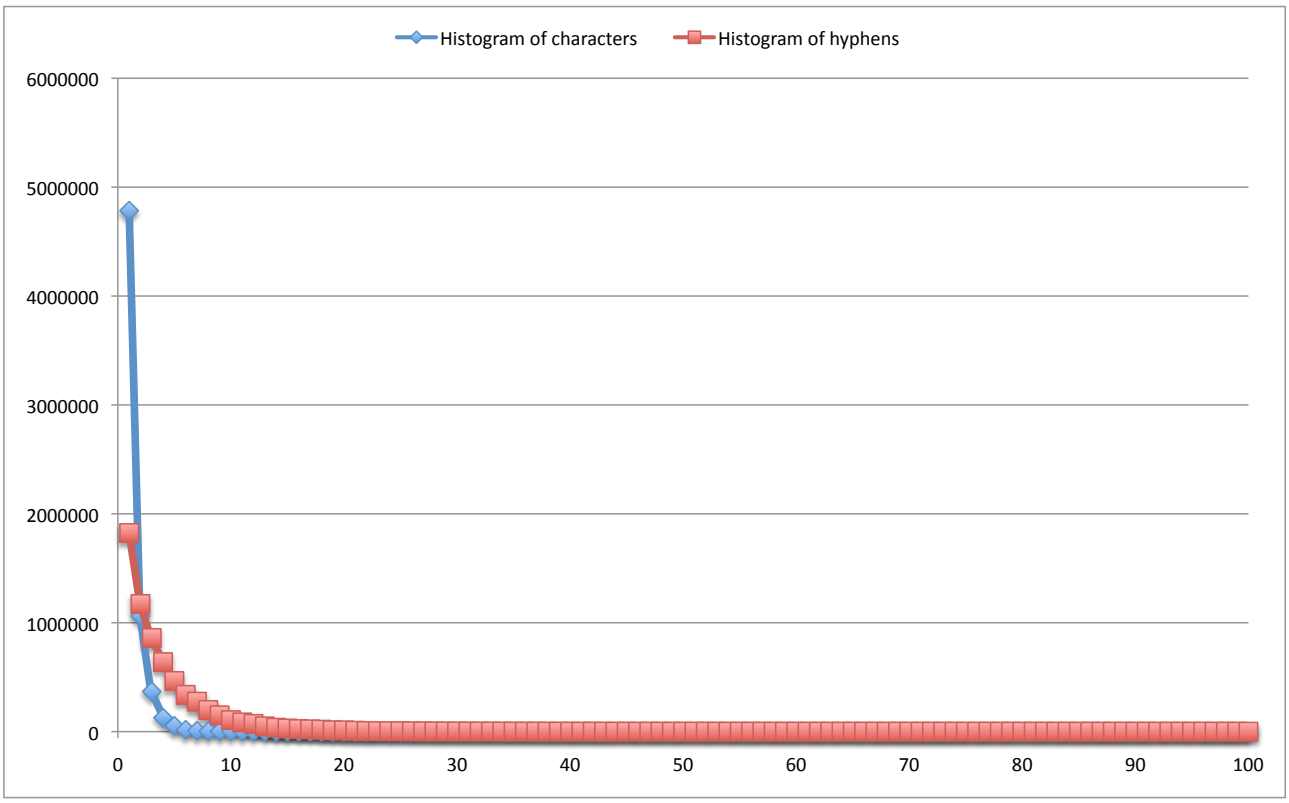
A's copy: 8 2 3 1 2
C's copy: 5 3 4 1 3
G's copy: 4 10 1
T's copy: 4 2 4 1 4 1

Considering 4 byte integers, to compare with the above results (despite having 4 copies), the total is just 80 bytes.

This simple algorithm perform better than naive run-length compression only because of splitting the sequence. VINT is our trump card in distinctly outperforming naive run-length. Before delving into the details of how we use VINT, let us first understand why is it that we chose VINT.

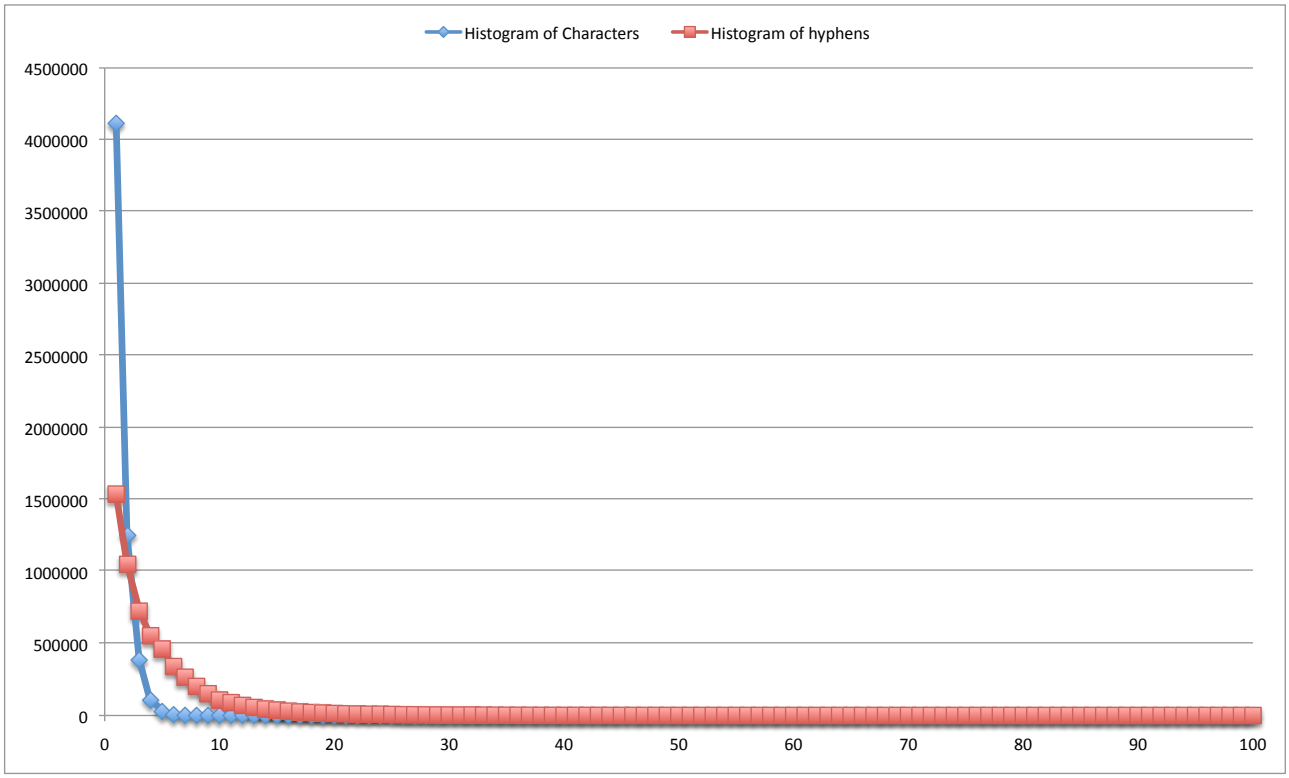
Motivation for VINT (Variable Integer):

To be able to efficiently encode the run-lengths of hyphens and characters, we need to know what the frequency of their various runs is. Let's plot the histogram of the occurrences of characters and hyphens within a split sequence. The split sequences used in the plot below are of chromosome 22 of the human genome[4]:

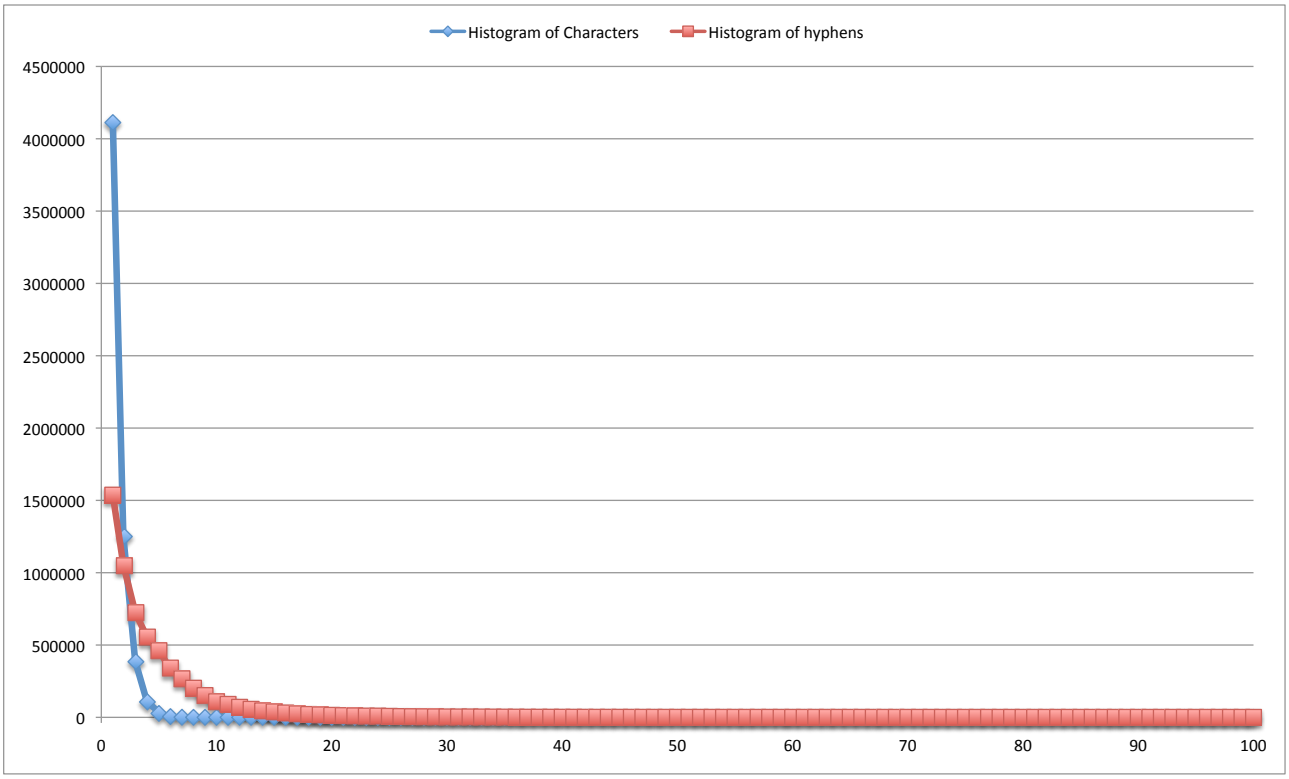


[Fig 2] - Histogram of Split Sequence for A

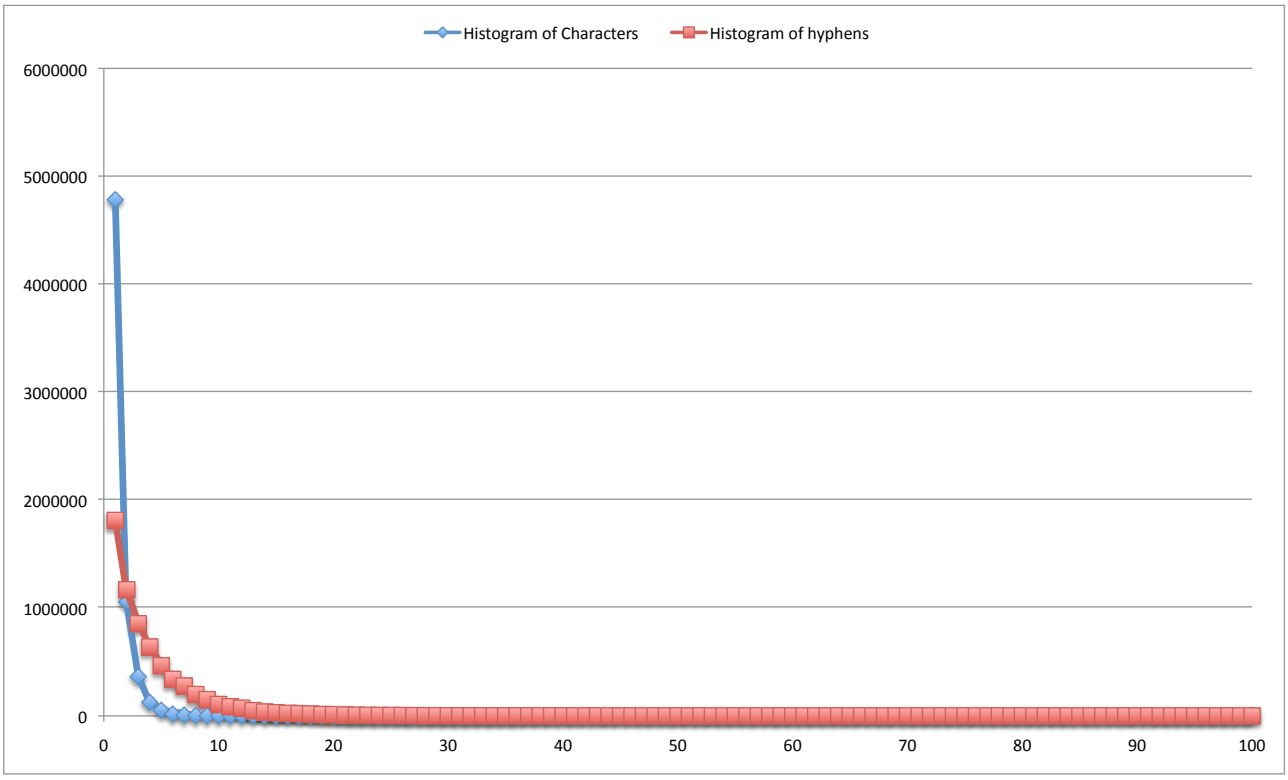
The above graph contains two histograms, one (red) for the occurrences of the hyphens within the split sequence and the other (blue) for the occurrences of the characters within the split sequence. The X axis denotes the number of continuous occurrences of hyphen / character and the Y axis shows the frequency of a articular run-length. The X axis is truncated to only runs of length 100 to emphasize on the interesting portion of the graph at the start. The tail of the graph tapers towards zero.



[Fig 3] - Histogram of Split Sequence for C



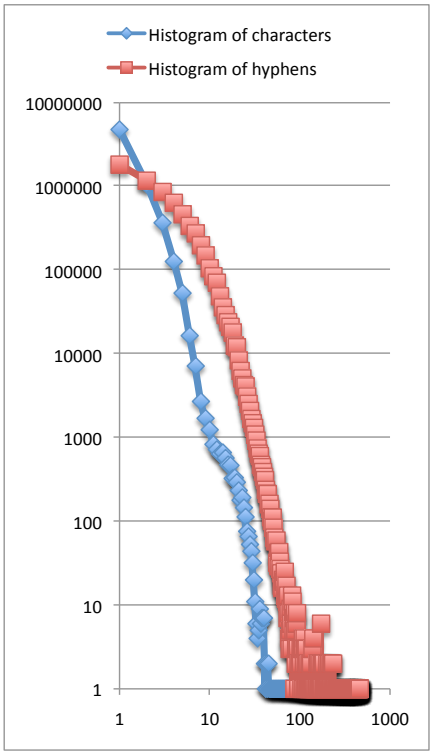
[Fig 4] - Histogram of Split Sequence for G



[Fig 5] - Histogram of Split Sequence for T

Since the histograms of all the copies are similar, we will consider A's copy for the rest of the discussion.

As soon as we see a histogram of this form, we are curious to find out what its distribution is like, viz. exponential, power law, etc. One of the main characteristics of any power law function is that its graphical plot on a log-log scale is linear. Following is the distribution of the above plot for character A:



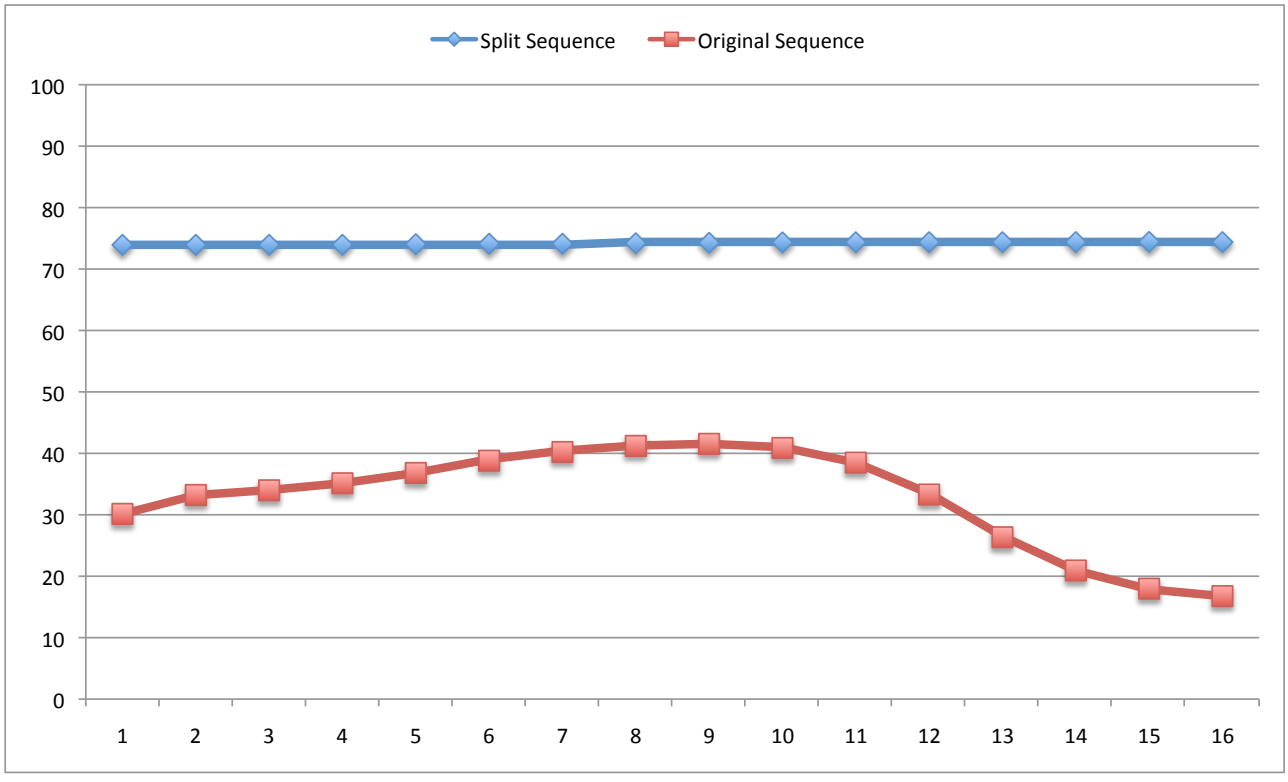
[Fig 6] - Histogram of Split Sequence for A

The above graph is a log-log plot of fig. 2. The linearity of the histograms, when plotted on a log-log scale reveal that in most likelyhood, they have a power-law distribution.

The interesting part is the extremely linear plot of the occurrences of the hyphens within a split sequence. Whenever there is a power law distribution that can be associated with an entity, the problem of compression of that entity is reduced to finding the variable-length code that corresponds to the distribution[7]. Thinking on the same lines, we are in the process of finding the function that fits this distribution to find the optimal variable-length code. VINT has a 1-byte granularity. For the purpose of effective compression, we need to make this granularity finer and think in terms of bits. As a result multiple codes can fit within a byte, possibly unveiling a much more efficient compression of genome sequences from the view of run-length compression algorithms.

Markov Compression:

Another avenue that the power law characteristic opens up, is the possible usage of Markov models to aid compression of these sequences. In this technique, we try to find the best order Markov model that fits our sequence and try compressing the probability state transition matrix that is built as a part of the model analysis. Currently, on analyzing the Markov models of order 1 through 16 on the original DNA sequence and the split DNA sequence for character A, we get the following results.



[Fig 6] - Markov Model Predictability on Original and Split Sequences

The X axis on the above graph is the order of the Markov model that was run on the original and split sequences of chr22. The Y axis denotes the percentage of correct predictions that could be done for each of order the Markov models from the X axis (1 through 16). The interesting aspect is the stability (and gradual increase) of predictability in the split sequence against the poor predictability of the original DNA sequence.

As seen above, the original sequence's predictability drops drastically after model 12, while the split sequence's predictability remains stable even till 16. The higher the model (assuming it is stable), the more contiguous set of characters are predictable. While better prediction is not directly related to better compression, we are in the process of finding out a way to harness the higher predictability, post splitting, to achieve higher compression.

On searching more about Markov compression techniques, we came across a relatively newer algorithm called LZMA[5], which essentially is the universal LZ (dictionary based) technique, but whose dictionary size is allowed to be greater than in the standard technique, and, also whose dictionary is built based on the Markov modeling. As shown above, there is a certain predictability in DNA sequences (less in unsplit ones, but there nevertheless). Intuitively, this algorithm should outperform completely dictionary based techniques viz. gzip and bz2.

On quickly running a few tests on split and original DNA sequences, we can see that in fact, it does perform better than both gzip and bzip2 (see table below). Also, like our graphs suggest, the compression is better in the split form than in the whole form. A gzipped split sequence performs slightly better than 100% when compared with the gzipped archive of the original sequence. That ratio is slightly reduced, but still in the high 90's when comparing the sequences using bzip2. The ratio is almost equally maintained by LZMA as well. When comparing amongst the split sequence archives in themselves, LZMA proves to be the algorithm providing maximum compression. Despite these results the overhead of 4 copies outweighs the advantage gained by better predictability. On this front, we are on the search of a more pure form of Markov compression for additional benefit.

	Chr 22 (Original)	**Chr 22 (Split - A)**
Uncompressed	34894545 (~33 MB)	34894545 (~33 MB)
GZIP	9685731 (~9.2 MB)	4987293 (~4.8 MB)
BZIP2	8790527 (~8.4 MB)	4210131 (~4 MB)
LZMA	7804508 (~7.4 MB)	3601064 (~3.4 MB)

[Table 2] - Comparison of popular compression algorithms on original and split sequences with LZMA

Parallelism in Markov:

In many situations, Markov models have much higher compression ratio than LZ techniques, but their runtimes are too slow to be used in practice. Due to splitting, we now have higher predictability and lesser states (as there are only 2 characters within a sequence). This will result in faster compression using Markov techniques. We can parallelize the operation(s) by simultaneously compressing the 4 sequences, since they are completely unrelated with each other for the purpose of compression.

DNA Generation - A side effect:

When predicting a sequence, Markov models construct a probability state transition matrix. Suppose we are running our sequence through a Markov model of order 12, and our sequence can consist of 2 characters, then the number of states in the state transition matrix are 2^12, i.e., on a general note, the number of unique characters raised to the number of the order that we are

considering. This produces all possible arrangements of the characters having length equal to the order we have decided.

Using this matrix, we can also construct sequences of arbitrary length having similar characteristics as that of the original. This can prove useful in situations where we want to create a large number of sufficiently random sequences that are based on the characteristics of one (or an average of multiple) sequences as a testbed. While generation of such data can prove helpful in the testing of a computational concept, its biological implications may be interesting (and worth exploring).

Future Work:

We are interestingly poised on two fronts regarding splitting the genome. On one hand, we are trying to find a function that fits the power-law distribution to construct the optimal variable-length encoding scheme to try and revive run-length compression as one of the best for DNA sequences. On the other hand, we are viewing Markov predictability (combined with parallelism) to be our possible answer for the best DNA compression algorithm.

Conclusion:

Splitting DNA sequences has given us an interesting view of genomic data. Preliminary tests and analysis are pointing at promising benefits in the area of DNA compression. With further study of this technique, we aim to unlock the answer for the smallest DNA footprint.

Acknowledgements:

The author thanks Prof. Ming-Yang Kao for his timely guidance and generous help during the course of this study. He also thanks Prof. Peter Dinda for his valuable insight into predictability and Markov techniques. Last, but not the least the author is grateful for the support and encouragement of Dr. Ankit Agarwal.

References:

[1] C. Kozanitis, C. Saunders, S. Kruglyak, V. Bafna, G. Varghese. “Compressing Genomic Sequence Fragments Using SlimGene”. Journal of Computation Biology. (2011) 18:401-413.

[2] L. Stein. “The case for cloud computing in genome informatics”. Biomed Central. (2010).

[3] J. Ziv, A. Lempel. “Compression of individual sequences via variable-rate encoding”. Information Theory, IEEE Transactions. (1978) 24:530-536.

[4] “Homo sapiens chromosome 22, alternate assembly CHM1_1.0, whole genome shotgun sequence”. - GenBank [http://www.ncbi.nlm.nih.gov/nuccore/NC_018933.1]

[5] I. Pavlov. “7-Zip”. Online - [http://www.7-zip.org]

[6] D. Salomon. “Variable-length Codes for Data Compression”. Springer. (2007) 2:92-93