Using Modern Compression Algorithms to Compress DNA Sequence Data

Connor Trotter

Candidate number: (006778 -0086)

May 2015

Computer Science - Bioinformatics

Abstract

Research Question: To what extent can the data within DNA sequences be compressed using modern compression algorithms?

Due to society's continuously growing dependence on technology, the appearance of big data in computing is becoming increasingly more common. From social networks to research labs, large amounts of data never cease to be sent, modified, or received. Data has always grown faster than technology, meaning that researchers are constantly dealing with data that is more complex than the technology used to processes it. Whenever we interact with exceedingly complex data compared to that of our system, it is necessary to compress that data.

Bioinformatics, for example, deals with the data within DNA sequences (as well as other biological sources of information) that are densely rich with data. Thus, in order for scientists to effectively interact with it, the data must first be compressed. However, DNA as a data source can be difficult to compress, seeing as the order of its values is seemingly random. There aren't very many predictable patterns. Fortunately, modern compression algorithms such as DNABIT Compress and BIND can still manage to compress DNA even though its complexity may normally be too large. Both the DNABIT and BIND algorithms make use of DNA's deceptively repetitive structure to more easily navigate streams of bioinformatic data.

This investigation will examine the extent to which these modern bioinformatic algorithms can compress the data present within DNA sequences and explain why the lossless BIND algorithm does so most efficiently. Additionally, the investigation will examine the implications of compression entropy and complexity in bioinformatic data, describing the issues

these intricacies create during compression. Through the use of modern compression techniques, bioinformatic compression algorithms overcome this complexity to make the transmission of data more efficient as a whole.

Contents

Introduction

In computing, compression is essentially performed by finding repeating patterns within a data stream, removing them from the data, and replacing them with pointers to their index within the stream. This method allows a program to reference a pattern only once, and replicate it at the positions of each of the pointers. For example, two common compression algorithms are the JPEG and GIF image compressors. Both algorithms take advantage of shortcomings in human perception by compressing an image without making it look too different from its original quality. The difference between the JPEG and GIF algorithms, however, is that JPEG is known as "lossy". This means that when JPEG compresses an image, it is actually removing information from the image and thereby directly degrading the quality. GIF is lossless and simply compresses the information within an image without losing any.

The effects of lossy algorithms appear when an algorithm wants to predict a string of data more than is possible with the given data. For example, if a JPEG algorithm wants to compress a picture of a forest, it might want to predict that every pixel of green on the leaves is the same shade of green. In reality, the picture has a large amount of different shades of green. In order to save efficiency, however, the JPEG algorithm will seek to reduce the number of green shades and therefore the amount of different pixels that have to be transferred. Depending on the degree to which an algorithm wants to predict a string of data, the effects of lossiness may or may not be felt.

The magic of compression arises not during the transfer of the data, but when it is received and replicated on the other side. If a file has been compressed in a lossless fashion, it can be easily uncompressed to the exact state in which it exists on the other end of the line.

However, if it has been compressed by a lossy algorithm, it can be restored only to an extent. Unfortunately, because the algorithm sacrificed some of the original data in favor of higher compression rates, it can never be fully uncompressed to its original state. In either case, using a compression algorithm to transfer a string of data removes lessens the amount of time required to send the information but increases the amount of time needed to process the data on the other end.

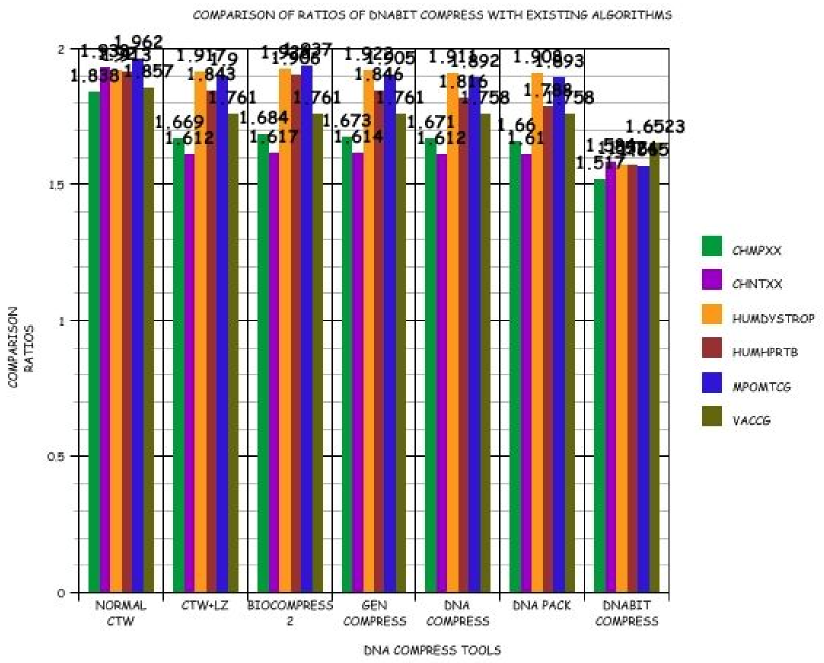
Ultimately, compressing data saves bandwidth during transmission, making the transfer of information more efficient. This concept is especially important in fields of scientific research, where big data is an everyday occurrence. There is a vast amount of data in the world, and it is measured every day in a variety of different fields of research. What makes scientific data different from data in other areas of computing is its predictability. Compression relies on being able to predict and replicate redundant values in a string of data, and data from the natural world is less likely to contain removable redundant patterns. The world's innate unpredictability requires that scientific compression algorithms find patterns in more complex ways. Bioinformatic algorithms, for example, take advantage of the redundant structure of DNA and other sources of genetic information by using multiple data streams and binary notation to find patterns that limit the efficiency of data transfer. The algorithms DNABIT and BIND are becoming increasingly popular in the field of bioinformatic compression, and this investigation will attempt to explain which algorithm performs its function most efficiently.

DNABIT Compressor

Compression is important in the context of scientific research for a variety of reasons. When analyses are being done on long strings of data, a computer must allocate a certain amount of memory for each value within the data. Thus, what makes compression valuable is the fact that it can minimize data to take up less memory, the rest of which can be used for quicker computation. Compressing bioinformatic data allows it to be more efficiently scanned, revealing information about its complexity, entropy, similarities to other strings of data, and patterns. Compressed bioinformatic data is useful as it can be compared with other data strings in multiple sequence alignment programs that search for shared patterns among various sequences. Removing redundancies in genetic data also highlights which information is important. If a particular pattern is not repeated, that likely means it is unique to a particular function.

As stated before, biological data comes in enormous quantities, and therefore cannot be effectively analyzed in an uncompressed state. DNABIT, created by Pothuraju Rajarajeswari and Allam Apparao, solves this problem. DNABIT functions by taking advantage of the limited range of possible values within a string of DNA sequence data. Because DNA is made of just 4 different values, any given value within a string of DNA sequence data is either A, C, T, or G. DNABIT represents each of these values as one of 4 binary bit codes (00, 01, 10, 11). This compresses the repetitive patterns within the sequence, in addition to saving memory (converting from char to byte).

DNABIT's method of binary representation is "a unique concept introduced in this algorithm for the first time in DNA compression"(Rajarajeswari). The compressor minifies patterns that are reverse repeats in addition to exact repeats. Due to this new method, DNABIT achieves a remarkable compression ratio. Most other bioinformatic compression algorithms claim ratios of approximately 1.72 bits/bases, but DNABIT achieves 1.58 bits/bases, a significant performance optimization. This allows a compressed sequence to be more easily compared with other compressed sequences in alignment programs that search for similarities and possible explanations for patterns.



A diagram of DNABIT performance compared to that of other popular compression algorithms

Since DNABIT achieves such great performance almost solely by taking advantage of repetitive patterns within a DNA sequence, one might argue that it is significantly less effective when used with a more complex (random) string of DNA. This argument is valid considering DNA is relatively unpredictable, but it fails to take into account the fact that "repetitive structures are present in over one-third of the human genome"(Rajarajeswari). Large chunks of DNA appear multiple times in order to specify a certain protein, and these large repetitive chunks can be removed during compression. Additionally, even if a given data sequence didn't have sufficiently repetitive patterns, DNABIT could still manage decent performance due to its binary notation method. Converting a sequence of ASCII characters to binary code enables a machine to compute the data faster, as it doesn't need to map 0's and 1's to a particular code like it does with letters. Due to this advantage, although much of DNABIT's method relies on repeat patterns, it could achieve results even for nearly non-repetitive sequences.

BIND Compressor

Upon learning about how DNABIT functions, it is easy to see that it achieves such great speeds due to the fact that it takes advantage of the small alphabet of DNA. However, because it can only compress sequences to one of four binary symbols (00, 01, 10, 11), DNABIT is a "lossy" compressor. This means that, like the JPEG algorithm, DNABIT does away with a certain amount of information upon compressing it. It must convert each pattern within a sequence to one of the four binary symbols, therefore not completely preserving the data within the original pattern. Thus, a DNA sequence compressed with DNABIT may take up less storage and be easier to interact with, but will also contain substantially less information.

In order to solve this problem, there must exist a bioinformatic compressor that behaves similar to the lossless GIF algorithm. That is to say that there needs to be a compressor that can compress bioinformatic sequences while also preserving their information. This is where the BIND compressor comes in, a lossless alternative "specialized for compressing nucleotide sequence data"(Mohammed). Created by Tungadri Bose and Monzoorul Haque Mohammed, BIND differentiates itself from other bioinformatic compressors (such as DNABIT) in two main ways. For one, it provides lossless compression by using binary bit encoding similar to that of DNABIT but separates the process into two "data streams." This is a parallel processing method that allows the program to iterate through two halves of the data stream at the same time, therefore encoding the common nucleotide bases (A, T, C, G) faster. The algorithm allocates these two streams by having one stream encode A's and T's as 0s and G's and C's as 1s. The second stream encodes A's and G's as 0s and C's and T's as 1s. The result of this allocation is two streams of binary data that consist of "blocks" of consecutive 1s and 0s. This is where BIND's second differentiation comes in. The algorithm then further encodes these blocks with a unary operation, representing an n-length block with n bits. It encodes the blocks such that the first n-1 bits all have the same value (0 or 1) and the final (n) bit has the opposite value. This ensures that the blocks are all organized and fit together in the final compressed state.

Additionally, BIND can take in data that contains more than just the ATCG alphabet. In fact, it can even handle input in lowercase letters. "This makes BIND a loss-less compression approach that is suitable for practical use"(Mohammed) say the creators. The fact that it supports a larger input alphabet makes BIND a more versatile algorithm than DNABIT with more potential applications in the bioinformatic world. Instead of just compressing DNA, BIND can also compress other sources of nucleotide sequence data, such as RNA. Subsequently BIND has the potential to last longer in the bioinformatic world, as it can continue to compress data even as the data changes in form. While DNABIT and BIND both have their separate advantages and disadvantages in the context of DNA compression, BIND exceeds DNABIT in efficiency and versatility.

The Difficulty in Compression

The main reason that bioinformatic compression functions so differently from general compression is that the data is inherently different. In compressing a typical text file, there will be numerous repeat patterns that can be easily taken out and referenced to. However, in compressing a source of information such as DNA, these patterns will be more difficult to find and in some cases will only be revealed once the data string is encoded and reorganized (as in the BIND compressor). In the past, data with a biological origin was widely believed to be random because there were no observable patterns. Nowadays in the age of significantly faster computing one can confirm that biological data is not in fact random. Of course there are already algorithms that can compress bioinformatic data, but true "randomness" can be theoretically defined in terms of computing using Kolmogorov complexity. The Kolmogorov complexity of a data string is defined as the length of the shortest possible description of the data string.

$$K(s) = |\inf \{d_n(s)\}|$$

Where *d(s)* is a description of s, a data string. For example, the string "123456789" is easily computed by incrementing a starting value of 1 and looping until an exclusive maximum of 10 is reached. Thus, it has a significantly low Kolmogorov complexity. On the other hand, a string such as "sk7wjJ7nF20Nl" contains no repetitions or observable patterns and would be difficult to compute and recreate. This means it would have a higher complexity. The higher a string's complexity gets, the closer the string gets to being random. In the context of Kolmogorov complexity, a string is defined as random if the length of its description is equal to the length of the original string plus a constant.

$$K(s) = |s| + C$$

Essentially, the string can only be described by repeating it value for value. The value of the constant changes based on the method one uses to describe it.

However, data can be described in a variety of different languages, symbols, and notations. The constant will change based on whatever method one uses to recreate the string. Unfortunately, this means that Kolmogorov complexity is not explicitly computable. In the end it still provides information concerning roughly how computable (or compressible) a data string is. This makes it useful in the context of compressing seemingly random data (i.e. bioinformatic data). That being said, even with nonrandom sets of data, there is always an absolute minimum to which you can compress something. This depends on the data's entropy, or in other words the amount of information contained within the data.

Essentially, a string's entropy is determined by the amount of uncertainty within each value of the string and thus how unique the string is as a whole. If there are more probabilistically unique values within the string, then the string theoretically contains more information. In order to determine the uniqueness of each value within a string, one can model the data within a Markov chain. In a Markov chain, the probability each value within a sequence is dependent on the value that precedes it. Therefore, as a string grows in length, it grows in the amount of information it theoretically carries and how much the information can be compressed. Markov chains can also be categorized by order, or how far back they depend on previous values in a sequence. An order 1 M-chain only depends on the symbol immediately preceding the one that is being examined. Higher order M-chains depend on values that precede the current one by 2, 3, and so on.

$$P(t) = \begin{pmatrix} p_{AA}(t) & p_{GA}(t) & p_{CA}(t) & p_{TA}(t) \\ p_{AG}(t) & p_{GG}(t) & p_{CG}(t) & p_{TG}(t) \\ p_{AC}(t) & p_{GC}(t) & p_{CC}(t) & p_{TC}(t) \\ p_{AT}(t) & p_{GT}(t) & p_{CT}(t) & p_{TT}(t) \end{pmatrix}$$

A Markov chain matrix

One can calculate the entropy of a string by using Markov's formula for finding the overall probability of a given string:

$$P(s) = P(s_n|s_{n-1})P(s_{n-1}|s_{n-2})...P(s_2|s_1)\pi(s_1)$$

In a given bioinformatic DNA sequence, all possible values (A, C, T. G) will make up a probability distribution that can be used to determine the individual probabilities of each value. Once these values are obtained, it's a simple chain of products to calculate the uniqueness of a DNA string. Biological compression algorithms can use the statistical model generated by a strings entropy to give more probable values a shorter output. Using Markov chains provides a computable manner in which one can determine the amount of entropy within a string, and therefore how much it can be compressed. The idea of entropy supports that of Kolmogorov complexity by providing the degree of computability that K-complexity lacks.

Conclusion

In the modern age of computation, all data is big data. This is especially true in the fields of research science, as data must contain the maximum amount of information possible. Data that contains vast amounts of information can sometimes be difficult to work with due to the exhaustive processing power it requires.

Biological compression algorithms, at their essence, are meant to give us complete control over bioinformatic data sequences. Now that the immense size of genetic data no longer poses a problem, researchers can begin to delve ever deeper into the data that makes us who we are. For example, it might be possible to transcode bioinformatic data into an even more efficient file type, thereby making bioinformatic data drastically easier to work with and encouraging research teams to collaborate through an accepted standard. However, with such a wide variety of software available for analyzing bioinformatic data, it might be difficult to limit the data to one file type. Additionally, while data corruption is generally frowned upon when dealing with important data, researchers might someday have the opportunity to investigate how controlled corruption in bioinformatic data can lead to artificial mutations and biological diversification. The implications of controlled corruption (or even purposefully lossy algorithms) could also include curing disease by corrupting the genetic makeup of a virus or creating vaccines that can correct corruptions within certain proteins. None of these further investigations would be possible without algorithms like DNABIT and BIND that make complex bioinformatic data compressible.

In researching bioinformatic compression, I wanted to examine the extent to which computers can analyze our inner workings. I have always been fascinated with the "human element," or what makes us different from other beings. Many people choose to investigate this concept through philosophy or psychology, but I have rarely seen research on the uniqueness of humanity from the perspective of computer science. Looking at myself and others through a pragmatic computer science in this investigation revealed so much to me, including the massive amount of data that goes into just creating the color of our eyes and the similarities we share with others in the code of our bodies. Since middle school I have had a passion for programming and mathematics, but I've never really applied it in this way. Building things in code is a refreshingly organic experience, because one coder can build nearly anything provided he/she has the drive to do to. The startling realization I made during my investigation is that human beings are little more than creatures of code. Our DNA (as well as other sources of bioinformatic information) makes up who we are, and it is the work of nature and evolution. This thought has inspired me to further investigate topics like the ones I listed above, particularly the controlled corruption of DNA.

Bibliography

Apparao, A., & Rajarajeswari, P. (2011). DNABIT Compress - Genome Compression Algorithm. *Bioinformation*, *5*, 350-360.

Bose, T., Mohammed, M., Dutta, A., & Mande, S. (2012). BIND - An algorithm for loss-less compression of nucleotide sequence data. *Journal Of Biosciences*, *37*(4), 785-789. doi:10.1007/s12038-012-9230-6

McGeoch, C. (1993). Data compression. *The American Mathematical Monthly*, *100*(5), 493-497. doi: 10.2307/2324310

Rodemich, E. R., & Posner, E. C. (1971). Epsilon Entropy and Data Compression. *The Annals of Mathematical Statistics*, *42*, 2079-2125.

Ryabko, B., Reznikova, Z., Druzyaka, A., & Panteleeva, S. (2013). Using Ideas of Kolmogorov Complexity for Studying Biological Texts. *Theory Of Computing Systems*, *52*(1), 133-147. doi:10.1007/s00224-012-9403-6