TJHSST Senior Research Project
Data Compression Through Duplicate Elimination and Tagging
2008-2009

Jeffrey Thomas

June 10, 2009

Abstract

Data compression is a valuable tool to save memory and time when sending data between computers. Although many different methods exist, I believe to have created a new method to compress data based on simple probability and the concept of effective infinite data. This algorithm will also potentially work with any kind of data, and will always compress a significant amount of data. I intend to test the effectiveness of this algorithm in terms of both time saved when sending large amounts of data and the proportion of data compressed, and find the optimal case of the inner variables.

1 Introduction - Problem Statement and Purpose

In this paper, we will explore fully a completely original data compression method. This method relies on two inner variables, and can be run multiple times on compressed data. The main will be to find the optimal state in one iteration, if one exists. It is possible that the algorithm will have no optimal state, as increasing the variables to infinity will infinitely enhance the performance.

The algorithm will be tested on how much data the algorithm compresses. The data files it compresses will be randomly generated to simulate large files. In theory, the algorithm should be able to compress large data files to an optimal proportion any time it runs. This paper will test that claim.

If the algorithm is a success, it has the potential to be exceptionally useful in the world of computing, as it can perform on any type of file, and should be able to optimally perform every time it runs.

2 Background

The main purpose of data compression is to replace larger patterns of data with smaller representations. In doing so, there are two main methods; lossless, and lossy compression. Lossless compression does not lose any data in the compression/decompression process. It is used for data that requires accuracy, such as program and text files. Lossy compression allows for some data to be lost in the process, and is used in compressing images and other visual mediums (Data-Compression.com).

A common method of data compression is Huffman coding. This takes repeating symbols or patterns and replaces them with a smaller pattern or number that represents that pattern (McGeoch).

This is valuable when compressing large files with repetitious data, such as text files. The downside is that if the probability of each word or pattern is equal, then the efficiency drops dramatically (Lelewer and Hirschberg).

Lossy compression is utilized in various image compression. If certain colors are reduced in quality or removed altogether, the human eye cannot tell the difference between the compressed version and the original version.

P	*W*	C_1	C_2
.40	not	110	11
.35	save	00	11111
.14	the	01	001
.06	trust	111	111
.05	queen	10	10

Figure 1. Two codes for the same set of source words. The first is a prefix code, the second is not.

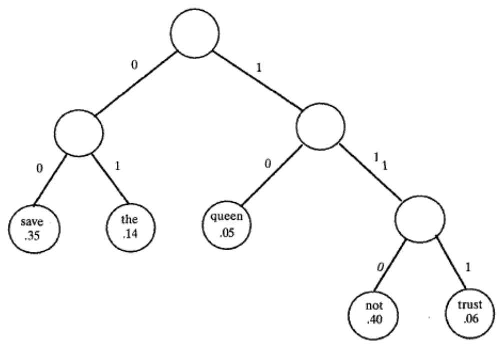


Figure 2. An encoding tree for code C_1.

Figure 1: Pictures taken from [1], Page 494-495

3 Development

3.1 The Algorithm

The algorithm I have developed works on the concept of data large enough to be considered effectively infinite and basic probability. Reading in four bits, there are sixteen different bit combinations that can be read. Of these sixteen, four are duplicates, in that the first two bits are identical to the last two in both order and value. Thus, one-fourth of the possible combinations are duplicates. This ratio holds true for any group of 2 to the n bits, provided n is greater than one. If the amount of data compared to the size of the bit pattern is very large, it could be considered effectively infinite. With nearly infinite data, basic probability dictates that each possible pattern of bits will have an equal chance of appearing. Thus, the duplicate patterns will appear in 1/4 of the bit patterns in the data.

For clarification, G is the number of groups analyzed at once, and B is the number of bits each group has.

G groups of size B are analyzed. Each individual group is checked for duplicity. If it is not a duplicate, nothing is done, and the next group is analyzed. If it is, than only half of the group is retained, and a marker is placed before the entire stream denoting which group, numerically, is a duplicate. The marker is of size log(G)/log(2), rounded up. After all of the groups are processed in this manner, then one final marker is added in front of all the data so far denoting how many groups were duplicates. The entire data stream is processed in this manner.

3.2 Data Storage

A large problem confronting this project was how to store the data that was being processed. Currently, 2 to the thirtieth power elements of data are being processed. With an inefficient method of storage, code runtime can easily exceed several hours. By using an ArrayList, I was able to cut runtime down to minutes. However, the amount of data exceeded the maximum size of the ArrayList. Two different solutions were attempted. The first involved using a single ArrayList filled with Strings. Each String would act as an array for characters. When this was implemented, runtime was over six hours. To remedy this problem, I placed the data in an ArrayList of ArrayLists, with each individual ArrayList holding one hundred thousand elements.

4 Testing

4.1 Methodology

The algorithm was judged by the total percentage of data saved from the initial set to the final set. Exponents of one through eight were tested with group sizes of 2 to the 0 through eigth power. Each set of values was tested twenty times, then averaged to find the final percentage of data saved.

4.2 Exceptions

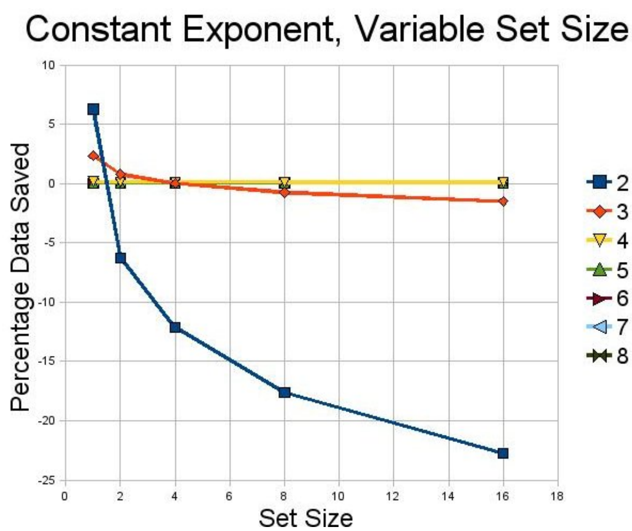
For values of the variables less than 3, the program encountered an OutOfMemoryError. With no way to increase the amount of memory allocated to the program, we were forced to lower the number of data elements. This should not impact results, as the final results are based off of a percentage.

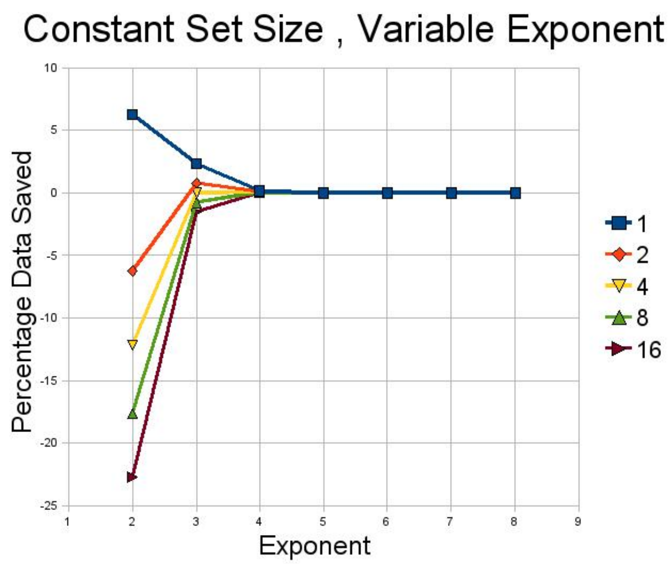
Additionaly, the data is large enough to be considered effectively infinite in all cases anyway.

5 Results

		Exponent						
		2	3	4	5	6	7	8
Set Size	1	6.250024	2.343305	0.171421	0.000715	0	0	0
	2	-6.248832	0.781598	0.121675	0.000575	0	0	0
	4	-12.184506	0.002896	0.097694	0.000561	0	0	0
	8	-17.646327	-0.764386	0.073257	0.000523	0	0	0
	16	-22.731542	-1.502715	0.04895	0.000508	0	0	0

This is the data. The columns are constant exponents, while the rows are contant set sizes.





6 Conclusions

After significant testing, we have concluded that this method of data compression is not a viable method for widespread use. The data shows that increasing either of the variables, contrary to our expectations, decreased the efficiency of the algorithm, eventually making it zero, or a negative percentage.

However, it is important to note that there is potential for additional testing. We were limited by the amount of memory we could devote to this program, which was automatically set by the system we used. If more memory could be devoted to this program, a greater amount of data could be tested. This may change the results of our larger variables. Due to time constraints, the program itself was not optimized. Should there be an error or inefficient method in the program, this would significantly limit the amount of data we could test the algorithm with, as more memory would be devoted to running the program itself and not storing the data.

Even with additional testing, it would be difficult for this method to become viable. For larger exponent values, the amount of data required to hold one copy of every possible pattern of bits becomes immense. (INSERT GRAPH HERE)

References

[1] McGeoch, Catherine C. ”Data Compression.” The American Mathematical Monthly 100: 493-497. 31 Oct. 2008 ”¡http://www.jstor.org/stable/2324310?seq=1&Search=yes&term=data&term=compression&list=hide&searchUri=%2Faction%2FdoBasicSearch%3FQuery%3Ddata%2Bcompression%26x%3D1%26y%3D10%26wc%3Don&item=15&ttl=16184&returnArticleService=showArticle&resultsServiceName=doBasicResultsFromArticle¿”.

[2] Data-Compression.com. EEF. 31 Oct. 2008 ¡http://www.data-compression.com/index.shtml¿

[3] Lelewer, Debra A, and Daniel S Hirschberg. ”Data Compression.” Data Compression. University of California. 31 Oct. 2008 ¡http://www.ics.uci.edu/ dan/pubs/DataCompression.html¿. Research Paper