Analysis of the copy model compression method

TAI - 2023/2024

Tiago Costa Nº 98601
João Mourão Nº 102578
João Rodrigues Nº 102487

Introduction

There are various methods to compress data, each with their own upsides and downsides. The copy model is a method that uses previous occurrences of a set of characters to try to predict the next characters from the text file. Correct predictions use a small amount of data to be stored while incorrect predictions require more data to be stored.

In this project, we will create a program that simulates a copy model to obtain an estimate of the amount of bits needed to compress various files, including variants with mutations (some characters are changed to other characters). We will then use this simulation program to compare the effectiveness of the copy model with other compression methods, such as gzip, bzip2 and zstd.

Copy model

The copy model is a file compressor that uses patterns, called kmers, that appeared previously and will be copied, being that copy, a prediction of the next characters in the file and then it stores the information needed in the compressed file to tell if the prediction is correct or not and if not, the information needed to get the right character. Correct predictions need a small amount of bits to store the information needed to identify the next character while wrong predictions need a bigger amount of bits to store this information. In the case where there is no repeated pattern that can be used, a second method is used, a fallback method, to compress characters until it finds a repeated pattern.

To implement a simulation of the copy model, a program was created that starts by reading a text file given by the user, storing the text and obtaining a list of all unique characters in the text. After that, it goes back to the start and starts processing the text. The program reads a set of characters,a kmer, and checks if it already exists in a hash table. The size of this chunk of characters is set by the user and the default size is 4.

If it exists, then it starts a repeat model for that set of characters, which consists of predicting the next character from the text file using the previous occurrence of that kmer, calculating the probability of this prediction being correct using the amount of hits and misses from previous predictions, then calculating the amount of bits needed to store the value based on the probability of the prediction being correct and updating the total amount of bits needed to compress the file.

The probability of having a correct prediction is calculated using the following formula:

$$P\ (hit) \approx \frac{Nh + \alpha}{Nh + Nm + 2\alpha}$$

Where ɑ is a smoothing parameter, which makes sure that even at the start of the copy model, the chance to predict the value correctly is never zero (it always starts at 50%).

The amount of bits needed to represent a new symbol, s, is then given by the following formula:

$$-\ log_2(P\ (s))$$

Finally, the set of characters that the program is looking at is moved forward by one and the kmer that the program was looking at is added to a hash table. This process is repeated at least 10 times and it stops when it hits the threshold of missed predictions allowed. The threshold depends on the method used, being the percentage of missed predictions in the last 10 predictions when the “last_ten” method is used and being the total percentage of misses for the current repeat model when the “last” method is used. In both cases, the minimum value for the threshold is 0, the maximum value is 1 and the default is 0.1. When the threshold is reached, the program checks the current kmer to see if it exists in the hash table to create a new repeat model.

If the kmer doesn't exist in the hash table, the program counts the next character as a missed prediction, the current set of characters is added to a hash table of kmers and the set of characters that the program is looking at is moved forward by one.

When the program reaches the end of the file, it outputs the total amount of bits that the compressed file would need.

Mutate

To create mutated variants of files, a program was created that obtains all unique characters in a text file given by the user and then goes through every character of the file and randomly replaces some of them with other characters from the list of unique characters.

To choose which characters to replace, this program uses a random value generated from a seed, with the seed being based on the time when the program is run. This value is a random number between 0 and 2 billion (2^31 - 1) which is divided by 100. The remainder of this division is then compared to the chance to mutate. if it’s smaller, then it mutates the character and if it isn't, the character stays intact. When mutating a character, a new random number is generated and divided by the amount of unique characters minus one. The remainder of this operation is then used to determine which character will replace the current character in the mutated file.

The chance to mutate a character is input by the user and the default value is 20%.

Other compression methods

To figure out how efficient the copy model is, we will compare it to 3 popular general purpose compression methods:

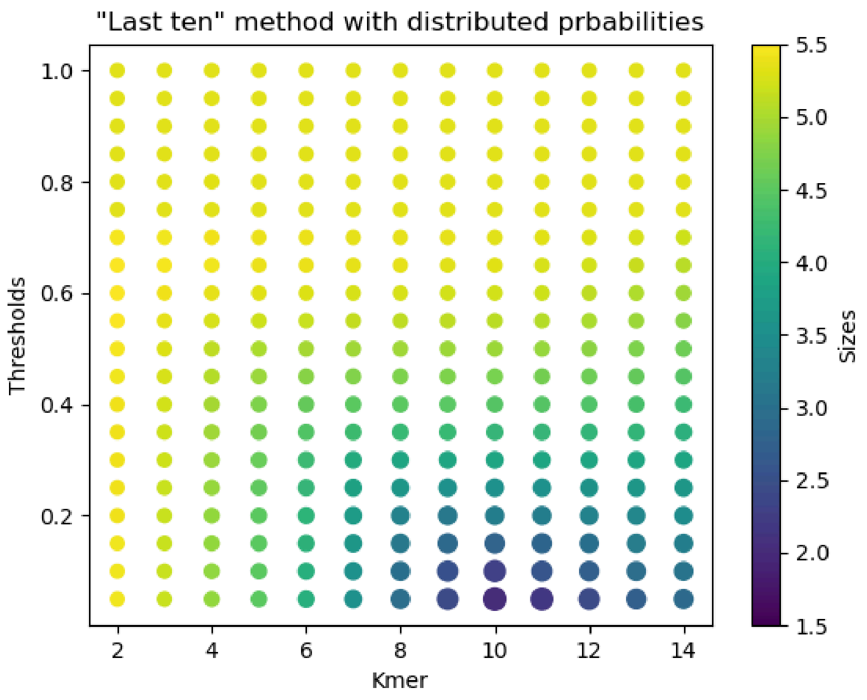
- gzip, a format that uses the DEFLATE algorithm and Huffman encoding to compress files.

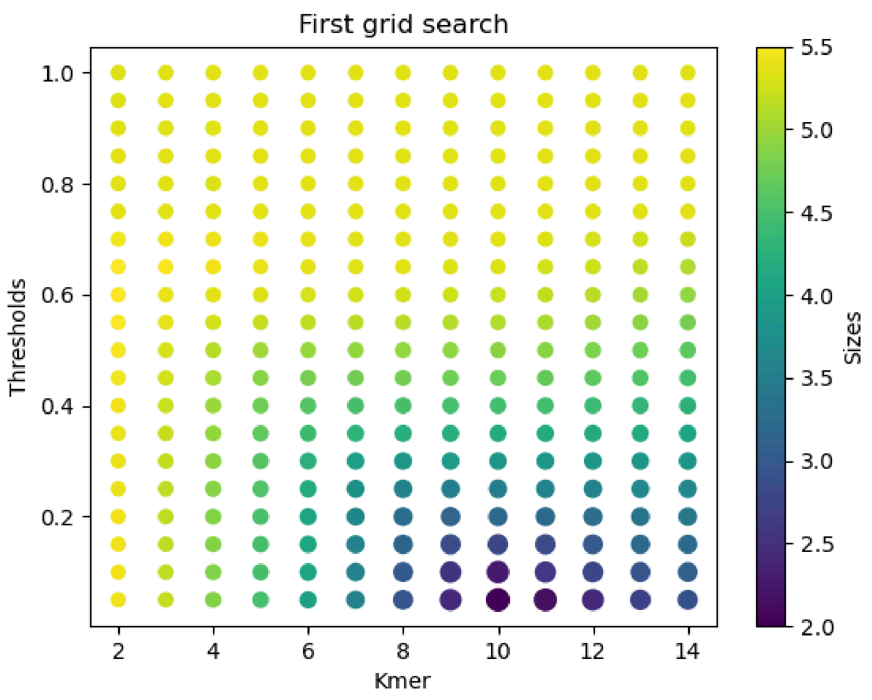
- bzip2, an algorithm that uses 9 layers of compression techniques (like run-length encoding, Burrows-Wheeler transform and move-to-front transform) to obtain better compression rates than other algorithms, at the cost of a slower run time.

- zstd, an algorithm that uses a dictionary-matching stage (LZ77 algorithm) and an entropy-coding stage (with both Huffman coding and finite-state entropy).

Results

After making the copy model simulation program, we tested various parameter values for both the chunksize (the size of each kmer) and the threshold (amount of fails before it picks a new kmer) on the chry.txt file, using both the “last ten”method and the “default” method, which got us the following results:





From this, we can conclude that a chunksize value of 10 to 11 characters and a threshold of 0.05 or 0.1 gives us the best compression rates. With the best parameters, the file becomes around 11 times smaller, going from 22MB down to 2MB or even around 1.6MB for the “last ten” method.

The execution time is a relevant part of the program, however, the method with better results (“last ten”), doesn’t oscillate too much given the different thresholds and chunksizes, so, we will simply choose the parameters with the best compression results.

Comparison with other methods

To compare the copy model to general purpose compressors, we used them all to compress the file chry.txt that was provided to us and the file RJ.txt which contains Romeo and Juliet, by William Shakespeare. For the copy model, we used the best chunk size and threshold values that we found on the previous test.

Finally, we used the mutate program to mutate both files with a mutate chance of 50% and redid the previous test on these new text files, which gave us the following results:

Analysis of the copy model compression method

	Original	Copy model	gzip	bzip2	zstd
Chry Normal	22.7MB	1.62MB	6.2MB	5.7MB	6.2MB
Chry Mutated	22.7MB	1.95MB	6.6MB	6.2MB	7.1MB
RJ Normal	146.7KB	95.1KB	58KB	46.4KB	61KB
RJ Mutated	146.7KB	110.5KB	107KB	109KB	105.4KB

From these results, we can see that the copy model has a significantly better compression rate on the chry.txt file when compared to other compressors. This file has a very limited set of unique characters, only having 4 unique characters. Because of this, randomly repeating patterns are more likely to happen and correct predictions are also more likely to happen, which increases the compression rate of the copy model.

For the RJ.txt file, the copy model had a significantly worse compression rate when compared to the more popular compressors. This is because while in language, there are repeating patterns that can show up often, the likelihood of having correct predictions is based entirely on the length of the word that's being repeated and the next words in the text, which decreases the amount of correct predictions drastically.

All of the compressors were affected by the mutations applied to the chry.txt file, which decreased their compression rate by a similar amount. Meanwhile on the RJ.txt file, the copy model gets affected by the mutation a lot less than the other compression methods.

Conclusion

The copy model is a very efficient way to compress text files for files with a lot of random repeating patterns and a small amount of unique characters, such as DNA sequences. On these types of files, the copy model beats the general purpose compression methods that we tested by a significant margin. On files with more unique characters and less random repeating patterns, since the repetition for these patterns is deliberate, the general purpose compressors have a better compression rate than the copy model.

Applying mutations to text files decreases the compression rate of all of the compression methods tested. For files with a small amount of unique characters and randomly repeating patterns, all of the compressors get affected by a similar amount, while for files with more unique characters and deliberately repeating patterns, the copy model gets affected less by the mutations than the other compression methods.

Contributions:

Tiago Costa Nº 98601 : mutate bash file, Mutate and report
João Rodrigues Nº 102487, João Mourão Nº 102578 : copy model bash file, copy model implementation and optimization , video and report graphs