The Chunk Compressor

Jason L Hutchens

August 5, 1997

Abstract

I have finished implementing a new type of compression system—one which uses two statistical models. One model is used to discover structure in the data being compressed, the other is used to encode the data for transmission. Results so far indicate that the method has merit; I was disheartened at first, but recent changes to the algorithm have improved things somewhat.

1 Introduction

As mentioned in my previous technical report, entitled "Does Chunking Work?", preliminary results from my Hybrid Data Compressor were discouraging to say the least. After moping around for a few days, I came up with a new implementation.

2 The New Implementation

The new technique is quite neat. It truely seperates the chunking from the parsing from the encoding. It may be broken up into two parts: encoding and updating.

2.1 Encoding

A greedy parser is used to segment up the input stream with respect to a dynamic dictionary. A symbol-string is read from the input stream by the parser; this string may in fact consist of a lone symbol.

The symbol-string must be encoded immediately, as the guy on the other end of the communication system must know what it is so he can update his models in the same way we will update ours. The encoder uses a statistical

model based around the same dynamic dictionary used by the parser to efficiently encode the symbol-string for transmission.

2.2 Updating

Once the symbol-string has been transmitted, we may use it to update our models. Firstly, the model used by the encoder is updated. As far as it is concerned, the symbol-string is an inseparable unit; it's just another item in it's dictionary.

Another model is maintained by the chunker. This model operates on the lowest-level symbols exclusively, which are usually the ASCII characters. To update this model, the symbol-string is parsed with respect to the dictionary used by this model. The symbols are used to update the model one-at-a-time, and a chunk is formed should the entropy rise above some threshold value. Any chunks found are added to the dictionary used by the encoder and parser.

3 Why Is It Better?

The new method is better primarily because it separates the three processes. Previously, the chunker was used as a parser. This turned out to be a silly idea, and the chunker rarely finds "good" chunks only; it tends to also find a lot of crap. The old method necessarily transmitted the crap too—the new method only uses chunks if the greedy parser finds them in the data stream. Hence the greedy parser acts as a crude chunk evaluator.

4 Results

As can be seen from the plot, results are encouraging. Admittedly, the model under scrutiny is a mere order-1 model—such a model would seldom be used in "real life". However, the vast improvement chunking offers over and above the hacks employed by the common PPMC compressor are encouraging.

The text used for evaluation was a 100kB portion of the Sherlock Holmes corpus. The process terminated sucessfully in under a minute for each entropy threshold. Without chunking, the text compressed to 43811 bytes; the chunk-compressor got it down to 33152 bytes. Amazingly, GZip could only manage 38743 bytes!

An entropy threshold of 4.6 bits resulted in this impressive result. 6894 chunks were discovered and added to the dictionary, although only 1921 of

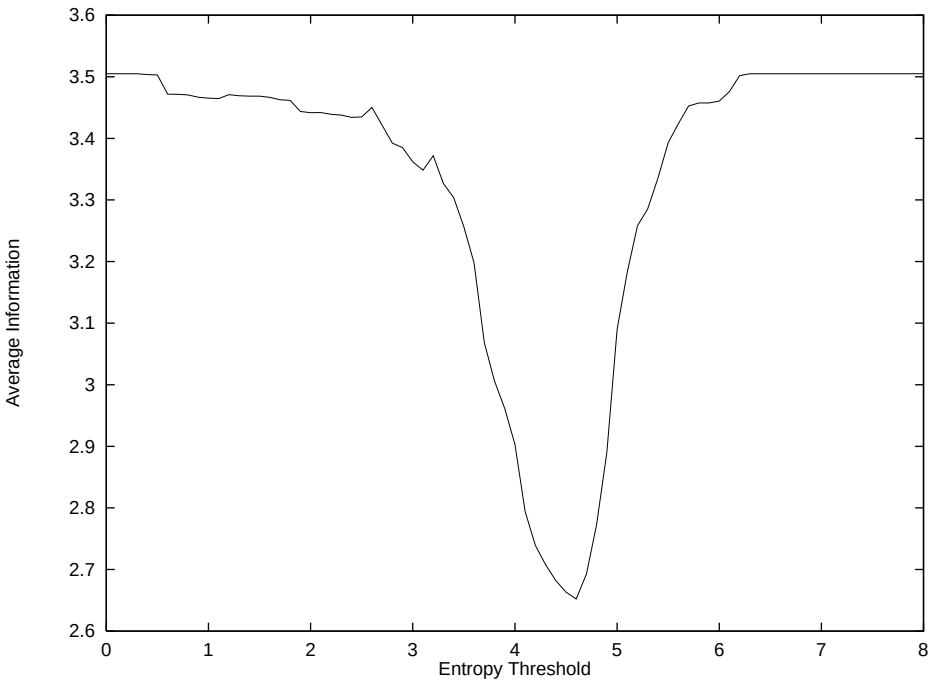

Figure 1: Compression performance as a function of Entropy threshold

Chunk	**Count**
the*	897
and*	478
a*	465
of*	461
to*	456
I*	448
,*	371
in*	324
en	278

Table 1: The most frequently used chunks

them were used to compress the data. The most popular chunks (i.e. the ones which were used most often by the encoder) are shown in the table, where * denotes whitespace.

5 Future Work

At the moment I am working on adding this compression technique to the GIT program (GIT, by the way, stands for "Grammatical Inference Toolkit"). This program provides a graphical interface to make experimentation a painless and fun process. I also need to add an off-the-shelf arithmetic coder to double-check my results; it shouldn't change the numbers drastically.

Once I've added the compressor to GIT, I plan to tinker around with it, trying different models and dictionaries, to see how far I can get. Of particular importance is a way of determining a good entropy threshold automatically.

Once I've gathered good results for the standard compression corpora, I will publish our findings.