What Hutter Prize winning entries reveal about descriptive complexity and truly universal lossless compression algorithms

Alexander Rhatushnyak

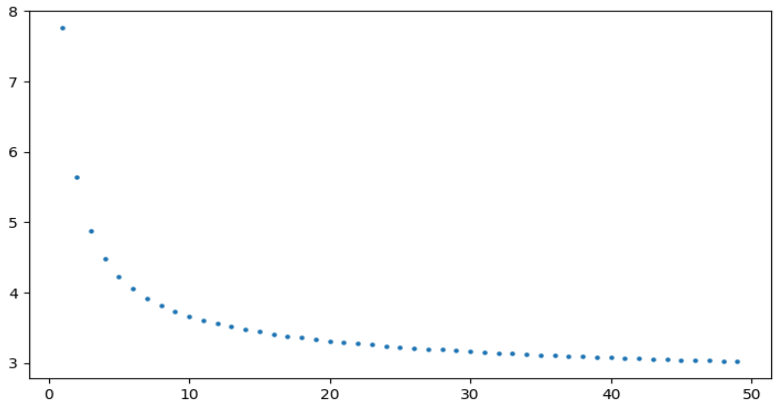
Hutter Prize is the only ongoing data compression competition with a cash prize: it awards 500 euros for each 1% improvement (with 50,000 euros total funding)[1] in the compressed size of the file enwik8, which is the first 100 million characters of a specific version of English Wikipedia.

We suggest that one of the most important observations, possibly the most important one, revealed by the Hutter Prize winning entries and the associated work, is that **descriptive complexity levels** are as numerous and well visible in artificial systems as they are in the physical world.

In the physical world, hadrons are composed of quarks, then atoms are composed of hadrons, molecules are composed of atoms, then the cells of living entities, multicellular organisms, and so on. On each new level the instances have a higher descriptive complexity, on average, than instances on the previous level.

In a data compression scenario such that an improvement in compressed size is desired at (almost) any cost, it seems natural to apply a **bitwise** algorithm, like [2] and [3], an algorithm that calculates for each bit **the probability that this bit is one**, making use of all of the already processed bits of input. If input is big enough, then if we plot the **compressed size** as a function of **computational resources** (memory and CPU time) used by our bitwise algorithm, the plot looks *somewhat similar* to the plots of functions

y = Const + 1/x and **y = C + 1/log(x)**.



Every doubling of x, the available resources, improves compressed size **y** by a smaller and smaller amount. **y = C1 + z1(x)** where z1(x) converges to zero when x runs to infinity.

It can easily be discovered that in the input, bits are the building blocks of units, most often fixed-size units: symbols in case of qualitative data,

samples or numbers in case of quantitative data. If input is enwik8, symbols are more often 8-bit, less often 16-bit symbols. If input, another example, is raw (uncompressed) audio, units are 16-bit samples, as a rule.

After we apply the knowledge of peculiarities of the symbolic system(s) used in our input, and plot the **computational resources** versus **compressed size** for the *improved* compression algorithm, we see again something similar to **y = C2 + z2(x)**, but this time the constant appears to be smaller, C2 < C1.

Next, we can discover what is composed of symbols. In the case of English text, as in enwik8, the units on the next complexity level are English words. After we apply this knowledge, for example, build a dictionary of English words and transform the input accordingly, the new plot of the improved compression algorithm looks like **y = C3 + z3(x), and C3 < C2**. Here is another example: in case of executable CPU code, the analogues of UTF-16 symbols are the elementary CPU instructions, and analogues of English words are the typical sequences of CPU instructions, for example: a comparison followed by a conditional jump, calculation of the maximum of two values, pushing a set of registers into stack, a dot product of two vectors.

Next, we can discover what is composed of typical sequences of symbols, which are English words in case the input contains mostly English text, as in enwik8.

Sentences of English words are the next complexity level. And after we discover and make use of the rules behind sentences, including **parts of speech** and **punctuation rules**, the plot is like **y = C4 + z4(x) where C4 < C3**. In case of CPU code, there are also certain rules for what is possible and what is not, what is more likely and what is unlikely in the input when it is executable CPU code. For example, "words" like "push many registers into stack" are likely at the headers of functions, and "words" like "pop many registers from stack" at the function footers.

In enwik8 there are two more complexity levels: paragraphs, and then the complete Wikipedia articles, also known as Wikipedia pages. Paragraphs composed of English language sentences are on the same complexity level as other Wikipedia specific entities, for example, lists "See also", "References", and "Other languages". Here you could possibly discover that switching individual models on and off, depending on the types of paragraphs, could be very beneficial for the compression algorithm, for arriving at y **= C5 + z5(x) such that C5 < C4**. A set of models for URIs should be rather different than a set of models for sequences of English sentences.

Finally, Wikipedia articles are at the top of the stack of complexity levels in enwik8. If you discover and make use of rules on how the articles are composed, once again you get a plot like **y = C6 + z6(x) and C6 < C5**. For example, every Wikipedia article has a header and a footer. Also, you could try to group similar articles together: "Persons" and "Places" articles, for example.

Do C1, C2, …, C6 correspond to descriptive complexity, also known as Kolmogorov complexity? The Kolmogorov complexity of an object is defined as the length of the shortest computer program, in a predetermined programming language, that produces the object as output. It is important that if the programming language is as simple as a CPU instruction set, that's one case, if it contains knowledge of the symbolic system properties (e.g. UTF-16), that's another case,

and if it contains also the knowledge of how thousands of English words are composed of UTF-16 symbols, that's a third case. Descriptive complexity of input E decreases if complexity of the programming language increases, and if information inserted into the programming language is relevant to information in E. When using the definition of Kolmogorov complexity, we should take into account the complexity of programming language. So the answer is yes, C1…C6 may be close to descriptive complexities, if **the sizes of decompression programs** are taken into account. Note these sizes must somehow take into account the complexity hidden in hardware, including CPU, in standard libraries of the programming language, and in the operating system.

It is worth mentioning that the original bitwise compression algorithm, after all the complexity levels are taken into account, is "aware" of entities at different levels, and does not always predict a bit at a time (by the way, not necessarily the immediately next bit). When it is appropriate, it predicts a complete symbol, or even a complete word, and possibly a complete sentence, or at least something about the type of sentence. This allows shifting the y=f(x) curve closer to the x=0 axis, which works very similarly to shifting it closer to the y=0 axis, if the function f(x) is like C + z(x).

Another issue worth mentioning briefly is that a very straightforward (and very slow) bitwise compression algorithm, for example multi-level artificial neural network, could eventually discover all the rules "naturally", especially if there are approximately as many ANN levels as the number of complexity levels in the input, and more neural cells than rules, on each level. However, applying the most important rules "manually" seems to push the f(x) graph significantly closer to the x=0 axis. This should improve as the state of ANN art improves, and the top-level ANN supervisor algorithm is able to switch between architectures, turn on and off stacks of levels, dynamically re-adjust the number of cells on each level, reconsider the batch sizes, and so on.

A truly universal **lossless data compression algorithm targeting a higher compression quality** should discover homogeneous portions and all the **complexity levels** in each portion, before building and applying a complex model, a model that should take into consideration all of the complexity levels, and other discovered properties, including data dimensionality, references to standard symbolic systems, etc. *Note in general it is important to split input into homogeneous portions; consider this example:* ***executable machine code*** *intermixed with* ***plain English text*** *and with* ***raw audio data.***

References

1. Website of the Hutter Prize: http://prize.hutter1.net

2. Matt Mahoney. The ZPAQ Compression Algorithm.

http://mattmahoney.net/dc/zpaq_compression.pdf

3. Byron Knoll. CMIX version 18. http://byronknoll.com/cmix.html