Nervous System: Putting Data on a Diet: How Lossless Compression Makes Information Skinnier

BY DAVID KALAT

With the aggressive pace of technological change and the onslaught of news regarding data breaches, cyber-attacks, and technological threats to privacy and security, it is easy to assume these are fundamentally new threats. The pace of technological change is slower than it feels, and many seemingly new categories of threats have been with us longer than we remember.

Nervous System *is a monthly series that approaches issues of data privacy and cyber security from the context of history—to look to the past for clues about how to interpret the present and prepare for the future.*

The act of "zipping" a computer file or directory into a smaller format has become routine, even though the practice may seem like some dark art of wizardry. Compressing a stack of blankets for storage using a vacuum sealer makes the overall volume appear smaller, for example, because removing excess air from the blankets keeps the part that matters and discards something else. But how exactly does one take an electronic document and make it smaller without losing some aspect of it? The electronic document, at root, consists of a sequence of 1s and 0s. There is no excess *something else* to remove. Compressing the data inevitably means discarding some portion of those 1s and 0s.

In 1977, Israeli data scientists Jacob Ziv and Abraham Lempel tackled this challenge and came up with a way to remove binary data from a document while retaining the information that binary data represented. Their theory of Universal Data Compression was based on "run-length encoding." In this context, "run" refers to a sequence of bytes in the electronic data source that recur more than once; "length" is a measurement of the size of any given run; and "encoding" means to substitute some sort of code in place of the run. Although run-length encoding works wonders for any type of binary data, it is perhaps most easily explained in terms of compressing written text.

When computers were first gaining widespread use in American businesses in the 1960s and 1970s, the standard byte length was 7 bits. Each bit is an individual instance of a binary value, basically a 1 or a 0. Each additional bit doubles the possible values the sequence can potentially hold. A 7-bit byte therefore allows for 128 possible values per byte. For the kinds of applications then in use, this was sufficient to cover the entire twenty-six-letter English alphabet in both upper and lowercase, the numerical values from 0 to 9, a variety of punctuation and symbols, and some legacy control codes left over from the era of Teletype machines. This system of mapping the 128 possible values represented by 7-bit computing to each of these unique characters was called ASCII (American Standard

Code for Information Interchange). In the interest of preserving backward compatibility, this core set of character mapping was maintained even as the advent of 8-bit computing doubled the number of possible values.

In practical terms, this means that each individual character in written text—including the spaces between words—occupies an entire 8-bit byte of storage, and certain foreign alphabets and specialized character sets like emojis require multiple bytes. The preceding sentence consists of characters comprising 251 bytes. Each new character added to the document increments the need for storage by one more byte, one at a time (hence the modern trend toward using a single space after a period, because double spaces require literally twice as much data storage).

The key to lossless compression is to replace redundant runs of data with a pointer that occupies less storage space. For example, the preceding text includes six instances of the string "character"; in ASCII, each one requires 9 bytes. That is 54 bytes already occupied by just this one run. If the document replaced each recurrence with a pointer back to the first one, as long as the pointer itself used fewer than 9 bytes, the overall amount of storage needed would be reduced without having lost any information.

In a 2004 interview, Ziv amusingly described his invention in terms of an analogy that was already dated at the time, but decidedly prehistoric now. Ziv compared his algorithm to *TV Guide*, which used to exist as a printed publication. As Ziv observed, if the same movie was shown several times in a week, the editors would give a description of the movie only once, and subsequent instances would refer back to the previous page. One could describe Ziv and Lempel's creation perhaps more universally by noting that the use of pronouns to replace names and concepts with simplified references is as old as human language.

Ziv and Lempel's original version of this algorithm in 1977 used a fixed-length pointer that identified 1) how far back to look to copy 2) how many bytes. Subsequent variations improved the efficiency of the algorithm by improving the design of the pointer. Later tweaks included developing a variable-sized pointer that could accommodate longer lookbacks for longer runs; changing how the system distinguishes pointers from uncompressed "literal" data; and developing an internal dictionary of runs that can assign shorter codes to more frequently used runs.

As groundbreaking as the invention was, it was perhaps too avant-garde in 1977 to find widespread use. Much of the early research into Universal Data Compression in the 1970s and 1980s was academic more than practical. Lempel and Ziv foresaw the commercial applications and hoped to patent the idea, but their employers at Bell Labs were disinterested at first. At that time, it had not yet been established that software even could be patented. Eventually, Bell Labs and Sperry-Rand obtained a patent on a hardware-based implementation of the algorithm. As Ziv later acknowledged, though, the underlying idea was both fairly simple and rooted in familiar concepts of how language works, so the limited hardware-based patents meant little, as engineers around the world could easily code their own variations. As the age of internet-based communications heightened the urgency to compress data without losing information, the marketplace for variant compression algorithms flourished. In general terms, this is how all forms of lossless compression work and still forms the basis for ZIP compression and other lossless approaches used today.

Legaltech news

This article was originally published in Legaltech News on July 7, 2022.