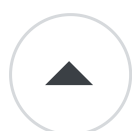


# What is the best file compression of random binary data that you can achieve? [closed]

Asked 13 years, 11 months ago   Modified 7 years, 8 months ago

Viewed 85k times



42



**Closed.** This question does not meet [Stack Overflow guidelines](#). It is not currently accepting answers.



This question does not appear to be about programming within the scope defined in the [help center](#).

Closed 11 years ago.

[Improve this question](#)

Specifically, what programs are out there and what has the highest compression ratio? I tried Googling it, but it seems experience would trump search results, so I ask.

random

compression

binary-data

Share

Improve this question

Follow

edited Jan 17, 2016 at 14:48



Brian Tompsett - 汤莱恩

5,875 ● 72 ● 61 ● 133

asked Jan 17, 2011 at 17:45



ox\_n

667 ● 2 ● 7 ● 12

---

32 Truly random data can't be compressed. ;-) The more helpful answer is longer: what are the properties of the data being compressed? (sound, image, video, binary executable, etc.) Can you tolerate loss of information? – [Throwback1986](#) Jan 17, 2011 at 17:47 ✎

---

4 Apparently random data is impossible to compress. Imagine that. Impossible. So I shouldn't be able to do it? That's disappointing. – [ox\\_n](#) Jan 18, 2011 at 8:04

---

8 "Truly random data can't be compressed." Haha. False. Truly random data *that is perfectly spread* cannot be compressed. Compression relies on redundancy, and redundancy is very possible in random data. Granted, you likely won't get much compression out of random data due to its tendency to be much more evenly spread, but it is certainly possible. – [Andrew](#) May 30, 2015 at 1:29

---

2 @Andrew Are you saying that: if you compress every possible n byte file, the average size of the compressed results will be less than n bytes? I'm pretty sure you can disprove that mathematically. – user354134 Dec 8, 2016 at 15:25

---

2 Very, very slightly, yes. True randomness != perfectly non-redundant distribution; therefore, due to expected redundancies of even a slight amount, an optimal compression algorithm applied will result in very slightly smaller file sizes. – [Andrew](#) Dec 9, 2016 at 17:09

---

2 Answers

Sorted by:

Highest score (default)





68



If file sizes could be specified accurate to the bit, for any file size  $N$ , there would be precisely  $2^{N+1}-1$  possible files of  $N$  bits or smaller. In order for a file of size  $X$  to be mapped to some smaller size  $Y$ , some file of size  $Y$  or smaller must be mapped to a file of size  $X$  or larger. The only way lossless compression can work is if some possible files can be identified as being more probable than others; in that scenario, the likely files will be shrunk and the unlikely ones will grow.

As a simple example, suppose that one wishes to store losslessly a file in which the bits are random and independent, but instead of 50% of the bits being set, only 33% are. One could compress such a file by taking each pair of bits and writing "0" if both bits were clear, "10" if the first bit was set and the second one not, "110" if the second was set and the first not, or "111" if both bits were set. The effect would be that each pair of bits would become one bit 44% of the time, two bits 22% of the time, and three bits 33% of the time. While some strings of data would grow, others would shrink; the pairs that shrank would--if the probability distribution was as expected--outnumber those that grow (4/9 files would shrink by a bit, 2/9 would stay the same, and 3/9 would grow, so pairs would on average shrink by 1/9 bit, and files would on average shrink by 1/18 [since the 1/9 figure was bits per pair]).

Note that if the bits actually had a 50% distribution, then only 25% of pairs would become one bit, 25% would stay

two bits, and 50% would become three bits.

Consequently, 25% of bits would shrink and 50% would grow, so pairs on average would grow by 25%, and files would grow by 12.5%. The break-even point would be about 38.2% of bits being set (two minus the golden mean), which would yield 38.2% of bit pairs shrinking and the same percentage growing.

Share Improve this answer

edited Mar 31, 2017 at 15:40

Follow

answered Jan 17, 2011 at 18:13



supercat

80.8k ● 9 ● 174 ● 220


---

5 I take it that is a simple explanation of the Kolmogorov complexity. Not bad. – [ox\\_n](#) Jan 17, 2011 at 18:46

---

1 More detailed explanations would tend to make many readers' eyes glaze over. Although the approach of compressing two bits at a time to 1-3 output bits is simple, I think it conveys pretty well the nature of the challenge. Compressing 1-3 input bits into 2 output bits would be another approach e.g. (000, 001, 01, 1) but computing the associated probabilities would be harder. – [supercat](#) Jan 17, 2011 at 19:57

---

Excellent explanation of "why" compression works. I have always been a victim of the eye-glazing. +1 – [Steven Lu](#) Sep 15, 2011 at 23:52 

---



10

There is no one universally best compression algorithm. Different algorithms have been invented to handle different data.



For example, JPEG compression allows you to compress images quite a lot because it doesn't matter too much if the red in your image is 0xFF or 0xFE (usually). However, if you tried to compress a text document, changes like this would be disastrous.



Also, even between two compression algorithms designed to work with the same kind of data, your results will vary depending on your data.

Example: Sometimes using a gzip tarball is smaller, and sometimes using a bzip tarball is smaller.

Lastly, for truly random data of sufficient length, your data will likely have almost the same size as (or even larger than) the original data.

Share Improve this answer

answered Jan 17, 2011 at 17:57

Follow



[helloworld922](#)

10.9k ● 6 ● 51 ● 90

---

There has to be one universally best compression algorithm. I think that logic would demand that be true, unless there were multiple algorithms of equal compression ratios tied for the best. – [ox\\_n](#) Jan 17, 2011 at 18:49

---

There are indeed many methods which could be considered "tied" for the best compression ratio for a particular type of data, as well as many methods which are specialized for a

particular type of data which offer better performance for these types of data over generic methods (audio, picture, movie, etc.). You need to determine what assumptions can you make about your data, with the more assumptions usually (but not always) resulting in higher compression ratios for that particular type of data. – [helloworld922](#) Jan 17, 2011 at 21:08

---

---