Comparative Evaluation of Geospatial Raster Data Compression Techniques

Pavel Vladimirovich Shebarshin, *BSE184-2 Student*

Abstract—In the modern world, raster data surrounds us almost everywhere. They are used in photos that we see every day on websites, in weather forecasting and a variety of studies. It is difficult to overestimate their importance and the need to develop approaches to their processing and analysis. In my work, I consider NDVI floating-point raster data that I received after calculating data from the Landsat 8 research space satellite, as well as GeoTIFF raster data with positive integer values. These data will be processed by six specialized algorithms for compression and decompression of raster data in order to compare their speed, analyze the feasibility of their use in the practical application of data transmission and compare their effectiveness by compression ratio. For a more visual comparison of the effectiveness of algorithms, tables, graphs and diagrams of various indicators for comparing algorithms will be presented in the conclusions.

1 Introduction

Geospatial data is information that is tied to a specific location in the space of objects (in certain cases and time). Geospatial data is usually stored in the form of geographical coordinates (latitude, longitude, altitude, or projection coordinates) and auxiliary data. The amount of such data is growing at an ever-increasing pace, since they are mainly created by various electronic devices and applications. They are generated by phones, PDAs, CCTV cameras, satellites, sensor networks and vehicles. With the development of the Internet of Things and its industrial analogues, the volume of geospatial data will only grow. This kind of information is used in Geoinformation systems. A geoinformation system is a system that collects, stores, analyzes and visualizes spatial geo data and related information about necessary objects. The concept of a geoinformation system is also used in a narrower sense — as a tool (software product) that allows users to search, analyze and edit both a digital terrain map and additional information about objects. A geoinformation system can include spatial databases (including those under the control of universal DBMS), raster and vector graphics editors, and various means of spatial data analysis. They are used in cartography, geology, meteorology, land management, ecology, municipal administration, transport, economy, defense and many other fields. Scientific, technical, technological and applied aspects of the design, creation and use of geoinformation systems are studied by geoinformatics. [1], [2]

By territorial coverage, geoinformation systems are divided into global, subcontinental, national, often having the status of state, regional, sub-regional, local, or local. In some cases, such territorial GIS can be placed in open access on the Internet and are called geoportals [1] .

According to the subject area of information modeling, urban (municipal), subsurface use, mining and geological information systems (GGIS), environmental protection, etc. are distinguished; among them, land information systems have received a special name, as particularly widespread. [3]

Geoinformation systems can also be classified by problem orientation — solved scientific and applied tasks. Such tasks can be audit of resources, including cadastre, analysis, evaluation, management, planning, geomarketing. In addition, integrated geoinformation systems combine the functionality of image processing systems, as well as remote sensing data in a single environment. Multiscale or scale-independent geoinformation systems based on multiple representations of spatial objects in time, providing graphical or cartographic reproduction of data at any levels of a scale series based on a single data set with priority spatial resolution; spatio temporal geoinformation systems operating with spatio temporal data. Data in geoinformation systems describe, as a rule, real objects, such as roads, buildings, reservoirs, forests. Real objects can be divided into two abstract categories: discrete (houses, territorial zones) and continuous (relief, precipitation level, average annual temperature). Vector and raster
data are used to represent these two categories of objects. [4]

Multi-scale or scale-independent geoinformation systems based on multiple or multi-scale representations of spatial objects, providing graphical or cartographic reproduction of data at any of the selected levels of the scale series based on a single data set with the highest spatial resolution; spatio-temporal geoinformation systems operating with spatio-temporal data.
Data in geoinformation systems describe, as a rule, real objects, such as roads, buildings, reservoirs, forests. Real objects can be divided into two abstract categories: discrete (houses, territorial zones) and continuous (relief, precipitation level, average annual temperature). Vector and raster data are used to represent these two categories of objects. [4]

Raster data is stored in the form of sets of values arranged in the form of a rectangular grid. The cells of this grid are called pixels. The most common way to obtain raster data on the Earth's surface is remote sensing, conducted using satellites and UAVs. Raster data can be stored

• *Pavel Vladimirovich Shebarshin is with School of Economics Faculty of Computer Science Software engineering, Research University Higher, Email: pvshebarshin@edu.hse.ru*

in graphic formats, such as TIFF or JPEG. [1]

Vector data is usually smaller than raster data. They are easy to convert and perform binary operations with them. Vector data allows you to perform spatial analysis, for example, to find the shortest path in the road network. The most common types of vector objects are points, polylines, polygons.

Dots are used to indicate geographical features for which location is important, not their shape or size. The ability to designate an object as a point directly depends on the scale of the map. While it is advisable to designate cities as point objects on the world map, on the city map the city itself is represented as a set of objects. In GIS, a point object is depicted as a small geometric shape (a square, a circle, a cross), or a pictogram that conveys the type of a real object [1] .

Polylines are used to represent linear objects. A polyline is a polyline consisting of segments. Polylines represent roads, rivers, and streets. The reliability of multilinear images of objects also largely depends on the scale of the map. For example, a river on the scale of a continent can be depicted as a linear object, and on the scale of a city, its image as an object requires the characteristics of an area. The main characteristic of a linear object is its length.
Polygons are used to mark areal objects with clear boundaries. Examples are lakes, parks, buildings, countries. They are characterized by area and perimeter. Such data can be associated with vector images: for example, on a map of territorial zoning, the characteristic of the zone type can be associated with areal objects representing zones. The data structure is defined by the user. Based on the numerical values assigned to vector objects on the map, a map can be constructed on which these values are indicated by colors in accordance with the color scale, or circles of different sizes. Continuous fields of quantities can be represented by vector data. One of the ways to represent a relief is a triangulation grid. Such a grid is formed by a set of points with bound values. The values at an arbitrary point inside the grid are obtained by interpolating the values at the nodes of the triangle into which this point falls. [4], [5]

Semantic data can be linked to vector data: for example, on a map of territorial zoning, a characteristic of the zone type can be linked to areal objects representing zones. The structure and data types are defined by the user. Based on the numerical values assigned to vector objects on the map, a thematic map can be constructed, on which these values are indicated by colors in accordance with the color scale, or circles of different sizes. Continuous fields of quantities can be described by vector data. The fields are represented in the form of isolines or contour lines. One of the ways to represent a relief is an irregular triangulation grid. Such a grid is formed by a set of points with bound values (in this case, height). The values at an arbitrary point inside the grid are obtained by interpolating the values at the nodes of the triangle into which this point falls [5], [6] .

The most important elements of GIS are additional information related to geospatial data. There is also the concept of a data layer, which is a combination of attributes and coordinates, in some cases in time. For example, a map on which objects are plotted can act as a layer, and images of the same place at different times are temporary layers. Geospatial data analysis is used to get answers to questions about the real world — about the current situation at a given point in time, the vector of changes associated with various factors, with modeling and forecasts. [7] .

What does geospatial analysis give organizations?
Alternatively, thanks to it, you can visualize social media activity during certain events — for example, an environmental disaster. Matching tweets, posts, or blog activity with a geographic location can help rescuers or special services. Obviously, tracking such data sources with the help of people is unrealistic: during any large-scale incident, the number of tweets, messages and just fake news is very large. Therefore, complex analytical tools are needed to organize rescue operations, they are similar to the tools that marketers use to work with clients. Telecommunications companies can use geospatial data to monitor the outflow of users, as social networks provide information that a customer has moved to another area or country. In this case, the system monitors the availability of services in this area and manages to offer new services. The energy industry derives significant benefits from geospatial data. Performing spatial analysis, companies calculate what the peak energy consumption may be, based on the geographical location and equipment in a particular area. And the most developed area of application of geospatial data is, of course, advertising and marketing. The ability to offer certain services to the consumer exactly when he is in close proximity to the store is a tasty morsel for marketers.

As we can see, analytical tools partially based on methods of analyzing GIS data — terrain topology and information about objects have turned into complex algorithms that provide a number of different applications. Today, geoinformation systems combined with numerous data from user devices are a powerful tool for marketers, analysts, IT companies and many other organizations.

Compression algorithms are an important part of this whole system. Existing data compression algorithms can be divided into two large classes – lossy and lossless. Lossy algorithms are commonly used to compress images and audio. These algorithms allow you to achieve high compression rates due to selective loss of quality. However, by definition, it is impossible to restore the original data from the compressed result. Lossless compression algorithms are used to reduce the size of the data, and work in such a way that it is possible to restore the data exactly as it was before compression. They are used in communications, archivers and some algorithms for compressing audio and graphic information. The basic principle of compression algorithms is based on the fact that in any file containing non-random data, the information is partially repeated. Using statistical mathematical models, it is possible to determine the probability of repeating a certain combination of symbols. After that, you can create codes for the selected phrases and assign the shortest codes to the most frequently repeated phrases. To do this, different techniques are used, for example: entropy encoding, repeat encoding, and dictionary compression. With their help, an 8-bit character, or an entire string, can be replaced with just a few bits, thus eliminating unnecessary information [8] .

In the scientific study of climate and weather forecasting, the accuracy and timeliness of data collection is important, so it is very important to find the optimal method of transmitting them. Currently, different countries and companies use their chosen methods and algorithms for transmitting raster geodata. There is no single standard for this, and researchers usually rely on existing articles and research when choosing. Sometimes there may be a situation where it will be more profitable to transfer data without any processing or compression, since the time spent on data compression and transmission will be higher than the time spent simply on transmitting the same data. All this is due to the limitation of network bandwidth, since now its potential is very limited and it is impossible to transfer all the data at once, and the computing power installed in such environments may not be enough to quickly compress the data itself. [9]

With the development of modern technologies, specialized methods of compression and decompression of raster geodata began to appear, aimed at high efficiency in such environments. Despite their narrow focus on raster data, they exhibit very high compression ratios and speeds that are superior to simpler or traditional compression algorithms. As a result, scientists are beginning to give more preference to such algorithms. However, the problem is that over the past 15 years, a large number of new specialized algorithms have been developed, and their creators usually compare them with non-specialized algorithms that cannot reflect the full picture of their applied efficiency and usefulness for use in scientific environments with not very high throughput. [10], [11]

My work is aimed at comparing the effectiveness of specialized compression and decompression algorithms for raster geodata. Its peculiarity is that both lossy and lossless compression algorithms will be compared. At the time of the approval of the topic, no similar works were found on the Internet. This approach is due to the fact that high measurement accuracy is not important for all data, for example, the NDVI indicator, the data for which is used in my work to compare the effectiveness of algorithms, has values from -1 to 1, and a large number of decimal places has no applied value, which means that you can safely get rid of them. [12]

My work is aimed at comparing the effectiveness of specialized compression and decompression algorithms for raster geodata. Its peculiarity is that both lossy and lossless compression algorithms will be compared. At the time of the approval of the topic, no similar works were found on the Internet. This approach is due to the fact that high measurement accuracy is not important for all data, for example, the NDVI indicator, the data for which is used in my work to compare the effectiveness of algorithms, has values from -1 to 1, and a large number of decimal places has no applied value, which means that you can safely get rid of them get rid of it. This index is actively used in agriculture to solve a wide range of tasks. [12] NDVI is calculated by the absorption and reflection of red and near infrared rays of the spectrum by plants. The vegetation index values range from 0.20 to 0.95. The better the vegetation is developed during the growing season, the higher the NDVI value. Thus, NDVI is an index by which one can judge the development of the green mass of plants during the growing season. The most famous application of NDVI is for assessing the development of cultures. According to the NDVI distribution map, it is possible to estimate where the values in the field are very low, and where they are above average. For visual assessment, a color scale is used: gray means the condition of plants below critical (below 0.25), red-yellow-green mean, respectively, poor, average or excellent biomass development. These data should be able to be interpreted taking into account the phase of vegetation and the type of crop on the field. During the growing season, the indicator grows, reaches its peak around 0.80-0.85 (for cereals, this is the moment of earing) and then begins to decline. A decrease in the index at the end of the growing season reflects the process of maturation of agricultural crops, and if the index value is less than 0, this means that inanimate objects such as rivers, stones, buildings and the like are visible in the picture. Therefore, for example, for several fields of grain crops, according to the NDVI index, it is possible to determine the most optimal order of harvesting fields, because the lower the index, the drier the grain. Regular inspection of the fields from the images helps to notice changes in the event of the emergence of foci of infections or the appearance of pests. Due to the timely identification of such problems, protective measures can be carried out with the greatest efficiency [13] .

The data used in the study are raster images from the Landsat 8 satellite. This is an American Earth remote sensing satellite, the eighth under the Landsat program. The satellite was built on the basis of the LEOStar-3 platform by Orbital Sciences Corporation. The payload of the spacecraft was created by Ball Aerospace and NASA, the launch was made by the United Launch Alliance.

2 Description of the selected problem-solving methods and algorithms

Before moving on to the problems and their solution, let's talk about Basic definitions, designations, abbreviations.

NDVI - normalized relative vegetation index is a very simple and convenient indicator of the amount of photosynthetically active biomass (usually called the vegetation index). One of the most widely used and used in practice indexes for solving problems using quantitative estimates of the vegetation cover of the Earth's surface.

Calculated by the following formula:

$$NDVI = \frac{NIR - RED}{NIR + RED}$$

where,

NIR is the reflection in the near infrared region of the spectrum

RED is the reflection in the red region of the spectrum

The compression ratio is the main characteristic of the compression algorithm. It is defined as the ratio of the volume of the original uncompressed data to the volume of compressed data, that is:

$$k = \frac{S_o}{S_C}$$

where k is the compression ratio, S_o is the volume of the source data, and S_c is the volume of the compressed data.

Thus, the higher the compression ratio, the more efficient the algorithm is.

Predictor – parameter prediction function.

Raster data is data consisting of a matrix of cells (or pixels), which is organized into rows and columns (grid), where each cell contains a value that carries some information, such as temperature.

Throughput is a metric characteristic showing the ratio of the maximum number of passing units (information, objects, volume) per unit of time through a channel, system, node.

GeoTIFF is an open format for the representation of raster data in TIFF format together with metadata about geographical reference (georeferentiated raster). It uses the TIFF 6.0 specification, to which it adds several types of geotags that define the type of cartographic projection, geographical coordinate system, geoid model, datum, and any other information necessary for accurate spatial orientation of the satellite image.

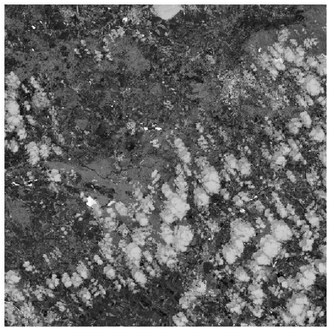
Fig. 1. Example of NDVI raster data

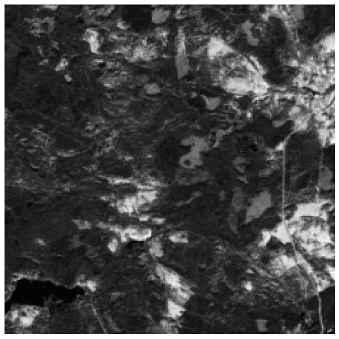
Fig. 2. Example of GeoTIFF raster data

Review and analysis of sources, analogues, selection of research methods

Overview of sources

The article [14] describes the lossless compression algorithm FPC, it was developed specifically for working with raster data in low-bandwidth environments. Its feature is the use of predictors to predict more efficient combinations for compression.

The BitGrooming algorithm is described in the article [15]. This is a lossy compression algorithm designed to compress data whose high accuracy is not important, initially it reduces bits using a special NSD coefficient, and then uses the deflate algorithm [16] to compress the resulting array.

The third BitShaving compression algorithm is described in the article [17]. This is a fairly old and simple floating-point data compression algorithm, it just zeroes a certain number of bits after the decimal point, and then compresses the data using the deflate algorithm [18] .

The K2raster algorithm, which is the most difficult to implement, is described in the articles [19] and [20]. K2raster is a new algorithm based on creating additional tables and K2Tree data structures [21], [22], [23] .

The Digits Routing algorithm, which is described in the article [24], calculates the quantization coefficient q, which is a power of 2, to set the bits. DigitRouting adapts the quantization coefficient to each sample value to achieve a given relative accuracy of NSD digits.

The SZ algorithm is described in the article [25]. It has many modifications for different tasks and needs, I chose an option capable of compressing data both lossily and without depending on the error that the user set. The main essence of his work is the use of three predictors to predict the next element of the array and subsequent data compression using median quantization of IEEE 754 floating-point numbers [26] .

The basis of the work is the IEEE 754 standard described in the article [27], all algorithms, as well as the conclusions of the work are based on its content.

Overview of analogues

In the articles [14], [15], [19], [24] comparisons of the presented algorithms with others are also given. The problem is that these algorithms are compared either with non-specialized algorithms, or with non-specialized and specialized algorithms, which, like the algorithms presented in the articles, compress with or without loss. In my work, I compare both lossy and lossless compression algorithms, and they are all specialized for working with raster data.

Selection of research methods

Also, the problem is that almost all raster data compression algorithms are implemented in C++ or Python, and almost no specialized algorithm is implemented in Java. So for my research, Java was chosen as the main language for implementing algorithms, and the results comparison mechanism was also implemented in Python. To create the research code, we used: the Maven library for building the project, geotols for converting raster data, log4j2 for convenient logging of the program, junit for testing the code, zxing for working with bit matrices, and I also implemented methods for byte type translation. Libraries were used to perform calculations in Python: matplotlib for plotting graphs, numpy for implementing mathematical logic and working with big data, pandas for data analysis. All data for the study were taken from the website earthexplorer [28] from the satellite Landsat 8. For the K2Raster algorithm, the GeoTIFF format data was not converted, for other algorithms, the data was converted by the GeoTIFF library scripts in Python to NDVI.

An important part of the work is the implementation of the algorithms themselves in the Java programming language. Each algorithm is implemented in a separate package and has an interface class for interacting with it. There are also classes in the work for implementing a mechanism for launching and obtaining measurement results, class diagrams in the resource folder. The code has a structure similar to a class library, but it has a mechanism for running a program in order to obtain measurements of algorithms. The paths to the data folders are specified as command-line arguments or are used by default if there are no input arguments. The results are recorded in a CSV file for more convenient parsing and data analysis. The CSV file contains information about the algorithm that made the measurement, namely: the size of the file being processed, the compression ratio, the execution time, the name of the algorithm, the parameters of the algorithm, if any, and the type of processing (compression or decompression). Further, the results obtained will be analyzed using various libraries of the Python programming language. The results will include graphs comparing all algorithms, lossy compression algorithms, lossless compression algorithms in terms of speed and compression coefficients, and algorithms for processing bits and storing bits with different parameter values will also be compared. All measurements are performed on square format images .tiff from 32 to 8182 pixels from the Landsat 8 satellite. These are satellite photos specially processed by the GeoTIFF Python library, they show the NDVI coefficient for all algorithms except K2Raster, original satellite images are used for it.

Also, the results of all algorithms except the K2Raster algorithm were submitted to the Deflate algorithm for compression of encoded data, since the result of their work is encoded data that needs to be compressed after processing. And K2Raster itself is a compression algorithm.

The structure of the main classes for implementing algorithms:

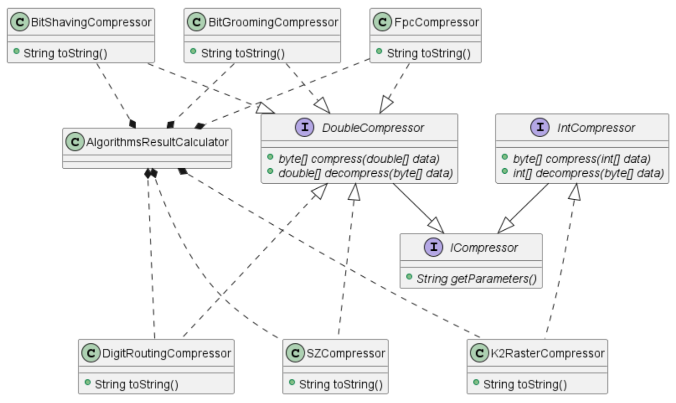

Fig. 3. Compressor uml diagram

As can be seen from the diagram, all compressors implement Int Compressor or Double Compressor interfaces, which in turn inherit the ICompressor interface. Each of the compressors contains an implementation of interface methods, this will be seen from the analysis of the diagrams below. Also, all compressors are in relation to the composition with the AlgorithmsResultCalculator class, which performs calculations of all measurements. It is also used for reading and processing spatial raster geodata.

FPC

FPC is a lossless compression algorithm for 64-bit floating-point data, it satisfies the bandwidth requirements of scientific computing environments. The main essence of the algorithm is to use two predictors of DFCM and FCM.

```
unsigned long long true_value, fcm_prediction, fcm_hash, fcm[table_size];
...
fcm_prediction = fcm[fcm_hash];  // prediction: read hash table entry
fcm[fcm_hash] = true_value;       // update: write hash table entry
fcm_hash = ((fcm_hash << 6) ^ (true_value >> 48)) & (table_size - 1);
```

Fig. 4. The pseudocode of the FCM predictor[14]

```
unsigned long long last_value, dfcm_prediction, dfcm_hash, dfcm[table_size];
...
dfcm_prediction = dfcm[dfcm_hash] + last_value;
dfcm[dfcm_hash] = true_value - last_value;
dfcm_hash = ((dfcm_hash << 2) ^ ((true_value - last_value) >> 40)) &
  (table_size - 1);
last_value = true_value;
```

Fig. 5. The pseudocode of the DFCM predictor[14]

With their help, a floating-point number deterministically converted to an integer receives a "prediction" and is entered into a hash table. After that, the most suitable prediction is selected as an integer, that is, the one after which the resulting number has most of the leading zero bits in binary representation and is compared with this number, then the predictor information (0 or 1) is recorded bitwise, the number of zero bytes and the remaining non-zero bits are transmitted as is.

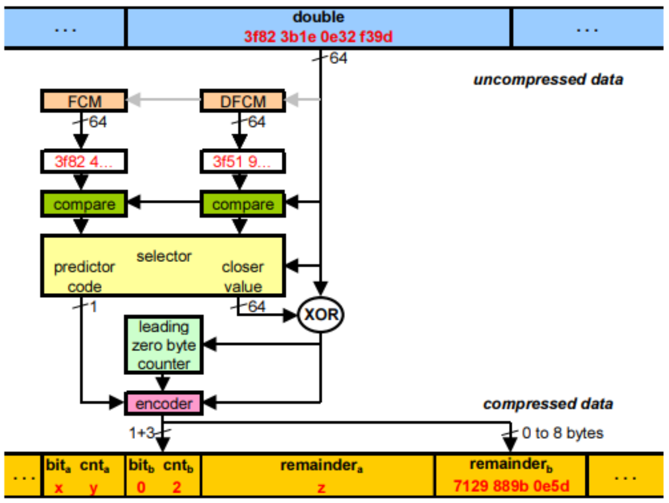

Fig. 6. Algorithm operation scheme[14]

FPC in the Java implementation uses ByteBuffer to implement the logic of the algorithm. DfcmPredictor and FcmPredictor are implemented as separate classes with hashtables. The main logic of the algorithm is in the FPC class, and it also uses the methods of the TypeUtils auxiliary class to convert primitive types of the Java language.

BitGrooming

BitGrooming is a modern and high-performance algorithm for getting rid of false precision in floating-point numbers. Its main application is the processing of raster data from satellites and other survey tools with the possibility of obtaining false bits in numbers (false accuracy). It can also be used simply to trim the precision of floating-point numbers when the data transfer rate is much more important than their accuracy. The algorithm is based on the

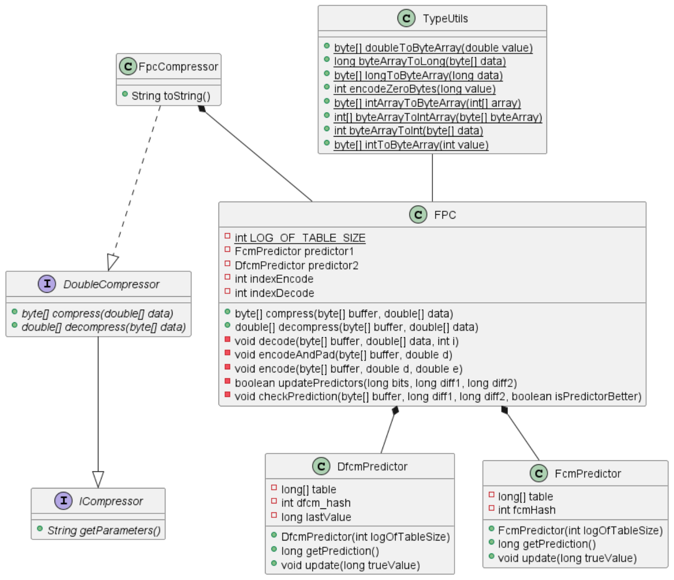
FpcCompressor
String toString()
TypeUtils
byte[] doubleToByteArray(double value)
long byteArrayToLong(byte[] data)
byte[] longToByteArray(long data)
int encodeZeroBytes(long value)
byte[] intArrayToByteArray(int[] array)
int[] byteArrayToIntArray(byte[] byteArray)
int byteArrayToInt(byte[] data)
byte[] intToByteArray(int value)
FPC
int LOG_OF_TABLE_SIZE
FcmPredictor predictor1
DfcmPredictor predictor2
int indexEncode
int indexDecode
byte[] compress(byte[] buffer, double[] data)
double[] decompress(byte[] buffer, double[] data)
void decode(byte[] buffer, double[] data, int i)
void encodeAndPad(byte[] buffer, double d)
void encode(byte[] buffer, double d, double e)
boolean updatePredictors(long bits, long diff1, long diff2)
void checkPrediction(byte[] buffer, long diff1, long diff2, boolean isPredictorBetter)
DoubleCompressor
byte[] compress(double[] data)
double[] decompress(byte[] data)
ICompressor
String getParameters()
DfcmPredictor
long[] table
int dfcm_hash
long lastValue
DfcmPredictor(int logOfTableSize)
long getPrediction()
void update(long trueValue)
FcmPredictor
long[] table
int fcmHash
FcmPredictor(int logOfTableSize)
long getPrediction()
void update(long trueValue)

Fig. 7. Diagram of the FPC algorithm

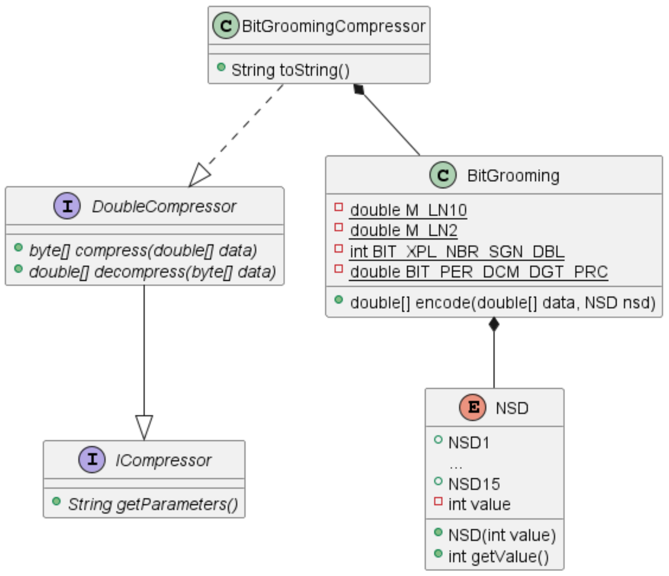

Fig. 8. UML diagram of the BitGrooming algorithm

construction of two bit masks, they depend on the selected type of encoded numbers (Float32 or Float64), as well as on the input parameter NSD. The number of significant bits removed is calculated as ceil(3.32 * NSD) + 2, where NSD is in the range [1;15] [15]. Then two bit masks of the length of the number of bits of the input numbers are constructed, one zero, the second one, then they are shifted to the left by the counted number of significant bits. Further, the numbers under even numbers are multiplied bitwise by a single bit mask, and under odd numbers are added bitwise with a zero bit mask. The peculiarity of the algorithm is that when removing false accuracy due to bit quantization, it strives to preserve the necessary accuracy as much as possible with maximum bit encoding for more convenient data compression. You can see an example in Table 1.

TABLE 1
Example of BitGrooming operation

Sign	Expo-nent	Fraction (Significand)	Decimal	Notes
0	10000000	10010010000111111011011	3.14159265	Exact
0	10000000	10010010000111111011011	3.14159265	NSD8
0	10000000	10010010000111111011010	3.14159262	NSD7
0	10000000	10010010000111111011000	3.14159203	NSD6
0	10000000	10010010000111111000000	3.14158630	NSD5
0	10000000	10010010000111100000000	3.14154053	NSD4
0	10000000	10010010000000000000000	3.14062500	NSD3
0	10000000	10010010000000000000000	3.14062500	NSD2
0	10000000	10010000000000000000000	3.12500000	NSD1

In the Java programming language, BitGrooming is implemented in the class of the same name. The arguments of the algorithm are implemented as an NSD enumeration with 15 possible variants of the argument. The BitGrooming class is in composition with its compressor.

Unlike BitShaving, which simply changes a certain number of bits to zeros, BitGrooming changes the significant bits, which allows you to increase the speed of work.

BitShaving

BitShaving is based on reducing the precision of floating-point numbers by a certain number of bits. Unlike BitGrooming, logical multiplication by zeros comes from the very end of the binary representation of a number. That is, in the case of BitShaving, with numbers with low accuracy, it may happen that the accuracy will not be cut from the numbers, which will make the algorithm completely useless. That is, for Float32 numbers, the algorithm parameters will be in the range [1;23] [14], and for Float64 - [1;52] [14] .Another bad feature of the algorithm is the lack of bit quantization, as a result of which the algorithm very roughly cuts off the accuracy, as a result of which it may turn out that BitShaving cuts off the accuracy of data that could have physical meaning when using this data.

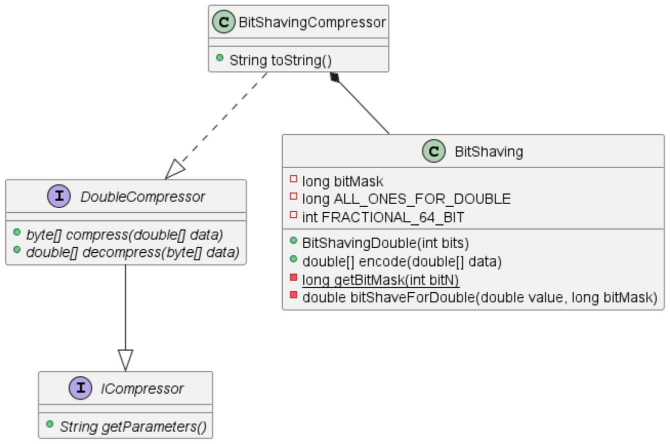
BitShavingCompressor
String toString()
DoubleCompressor
byte[] compress(double[] data)
double[] decompress(byte[] data)
ICompressor
String getParameters()
BitShaving
long bitMask
long ALL_ONES_FOR_DOUBLE
int FRACTIONAL_64_BIT
BitShavingDouble(int bits)
double[] encode(double[] data)
long getBitMask(int bitN)
double bitShaveForDouble(double value, long bitMask)

Fig. 9. UML diagram of the BitShaving algorithm

In Java, BitShaving has a very simple implementation, this is due to its prostate. The implementation of all the logic of the algorithm is in the BitShaving class. According to the structure, it has several constants, auxiliary methods and the main method of removing false precision.

DigitRouting calculates the quantization factor q, which is a power of 2, to set the bits. DigitRouting adapts the quantization coefficient to each sample value to achieve a given relative accuracy of NSD digits.

DigitRouting

DigitRouting uses uniform scalar quantization

$$\widetilde{s_i} = sign(s_i) * \left(\left\lfloor \frac{i|}{q_i} \right\rfloor + 0.5\right) * q_i$$

Where $\widetilde{s_i}$ - quantized value of sample values s_i. The quantization error is limited:

$$_i - \widetilde{s_i}| \le \frac{q_i}{2}$$

Number of digits s_i before the decimal separator in the value s_i equally:

$$d_i = \lfloor log_{10\,i}| + 1 \rfloor$$

We want to keep the significant digits of the sample value of s as well. This is roughly equivalent to keeping the rounding error less than half of the last tenth digit. Thus, the quantization error must be less than or equal to:

$$_i - \widetilde{s_i}| \le 0.5 * 10^{d_i - nsd}$$

This condition ensures that the rounding algorithm always preserves a relative error less than or equal to half the value of the smallest significant digit. We are looking for the highest quantization coefficient q_i such that:

$$\frac{q_i}{2} \le 0.5 * 10^{d_i - nsd} \; or \; log_{10} q_i \le d_i - nsd$$

Moreover, in order to reduce computational costs and improve compression efficiency, we are looking for a quantization coefficient equal to the power of two. This allows bit masking instead of splitting and creates a sequence of zero bits.:

$$q_i = 2^{p_i}$$

So, we are looking for the largest integer p_i such that:

$$p_i \le (d_i - nsd) * log_2 10$$

Finally, we take the value p_i such that:

$$p_i = \lfloor (d_i - nsd) * log_2 10 \rfloor$$

Then the obtained data is substituted into the formula:

$$\widetilde{s_i} = sign(s_i) * \left(\left\lfloor \frac{(s_i)}{q_i} \right\rfloor + 0.5\right) * q_i$$

DigitRouting, like BitGrooming, by quantizing bits, strives to maintain maximum accuracy while encoding bits as efficiently as possible for efficient compression.

In Java, DigitsRouting has a similar structure to BitShaving, but the implementation itself is more complicated. The algorithm class also has auxiliary methods and constant fields, but in addition, the MantissaExponent class is implemented to work separately with the mantis and exponent of floating-point numbers according to the IEEE754 standard. This is due to the fact that in the standard library of the Java language there are no methods for dividing a number into a mantissa and an exponent, unlike C++.

K2Raster

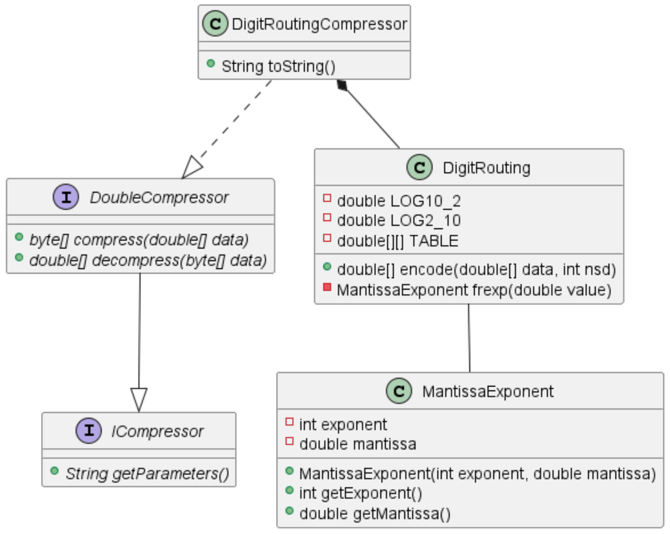

Fig. 10. UML diagram of the DigitRouting algorithm

The K2Raster algorithm is based on the K2Tree data structure, the operation of the Morton curve and the construction of additional tables. Initially, a matrix of positive integers comes to the input, then, according to the Morton curve Z algorithm, it is converted into a one–dimensional array, after which a bit matrix is constructed, the rows of which are all the values of the input numbers, and the columns are the values of the Morton code, that is, an adjacency matrix is obtained. At the intersection of numbers with Morton codes there are single bits, where there are no numbers – there are zero bits. Next, the K2Tree data structure is constructed using the bit matrix.

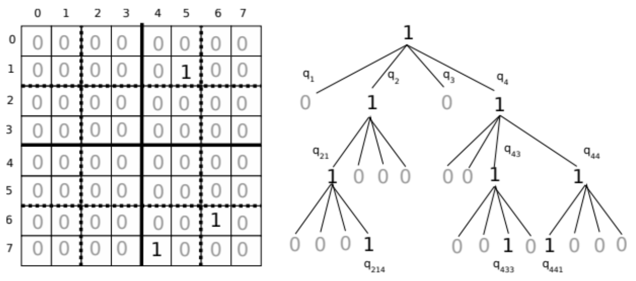

Fig. 11. K2Tree[5]

K2Tree – represents the binary adjacency matrix of a graph using a short representation for quadrotrees. K2 True builds an unbalanced k2-ary tree from a graph by recursively dividing the adjacency matrix into k2 submatrices of the same size. It starts by dividing the original matrix into k rows and k columns of submatrices of size k/2 by k/2. Each of the resulting submatrices k2 will generate a child element of the root node whose value is 1 if there is at least one 1 in the cells of this submatrix. The child element 0 means that the submatrix has all 0, and, therefore, the decomposition of the tree ends there. The method is executed recursively for each child element with a value of 1 until it reaches a submatrix full of 0 or reaches the cells of the original matrix.

The implementation of K2Raster has a rather compli-

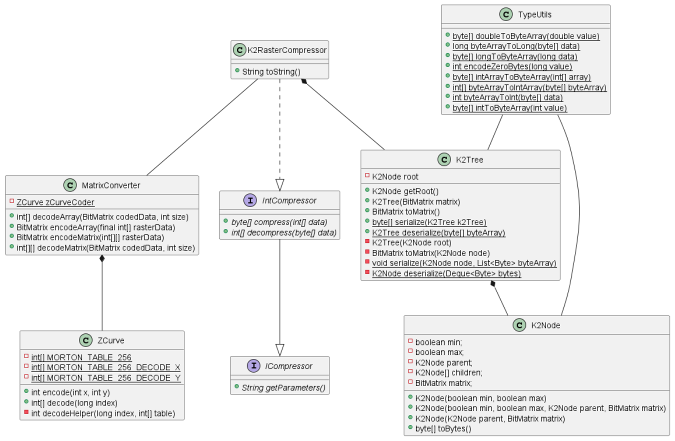
TypeUtils
byte[] doubleToByteArray(double value)
long byteArrayToLong(byte[] data)
byte[] longToByteArray(long data)
int encodeZeroBytes(long value)
byte[] intArrayToByteArray(int[] array)
int[] byteArrayToIntArray(byte[] byteArray)
int byteArrayToInt(byte[] data)
byte[] intToByteArray(int value)
K2RasterCompressor
String toString()
MatrixConverter
ZCurve zCurveCoder
int[] decodeArray(BitMatrix codedData, int size)
BitMatrix encodeArray(final int[] rasterData)
BitMatrix encodeMatrix(int[][] rasterData)
int[][] decodeMatrix(BitMatrix codedData, int size)
IntCompressor
byte[] compress(int[] data)
int[] decompress(byte[] data)
K2Tree
K2Node root
K2Node getRoot()
K2Tree(BitMatrix matrix)
BitMatrix toMatrix()
byte[] serialize(K2Tree k2Tree)
K2Tree deserialize(byte[] byteArray)
K2Tree(K2Node root)
BitMatrix toMatrix(K2Node node)
void serialize(K2Node node, List<Byte> byteArray)
K2Node deserialize(Deque<Byte> bytes)
ZCurve
int[] MORTON_TABLE_256
int[] MORTON_TABLE_256_DECODE_X
int[] MORTON_TABLE_256_DECODE_Y
int encode(int x, int y)
int[] decode(long index)
int decodeHelper(long index, int[] table)
ICompressor
String getParameters()
K2Node
boolean min;
boolean max;
K2Node parent;
K2Node[] children;
BitMatrix matrix;
K2Node(boolean min, boolean max)
K2Node(boolean min, boolean max, K2Node parent, BitMatrix matrix)
K2Node(K2Node parent, BitMatrix matrix)
byte[] toBytes()

Fig. 12. UML diagram of the K2Raster algorithm

cated structure in Java. For the operation of the first stage of the algorithm with the transformation of the matrix of integers into a linear representation, a class has been created that implements the Z algorithm of the Morton curve ZCurve. It is compositively used by the MatrixConverter class, which in turn builds an adjacency matrix from the linear representation of the original matrix and the Morton code. Next, in the algorithm compressor, it feeds it to the input for the K2Tree class, the leaves of which are K2Node. During serialization and deserialization of the tree, static methods from the TureUtils class are used to work with Java primitives.

SZ

The SZ algorithm is an algorithm that compresses floating–point data considering the compression error entered by the user. It turns out that if the error is not zero, then it will compress lossily, if the error is zero, then lossless compression occurs.

The algorithm is divided into 3 stages. In the first, data comes to the input of the algorithm – a compression error and the matrix of values itself, then the matrix of values is uniquely converted into an array of values using the Morton curve Z algorithm, as in the case of K2Raster.

In the second stage, data is processed using predictors. There are only three of them: the predictor of the preceding neighbor, the predictor of the linear curve fit, the predictor of the quadratic curve fit. Next, the best solution is sought, considering the error entered, and the best result is saved. C if none of the three options meets the requirements of the error, then the information about it is saved as a separate option. The pseudocode of the second and third stages of the algorithm is presented below:

In the last step, the median value among all the input data is calculated. It is subtracted from all values for which none of the predictors fit. This happens because bit quantization during compression is better for floating-point values close to zero according to the IEEE 754 standard.

The implementation of the algorithm in Java is presented below. As can be seen from the UML diagram, the algorithm does not require additional data structures, it only uses TypeUtils to work with Java language features and, like other algorithms, composes with its compressor.

If we summarize the work of all algorithm compressors,

```
Create a bit-array (denoted $\upsilon$) with $M$ elements, each occupying 2 bits.
Compute the value range size $r$ (= $\max(V_i) - \min(V_i)$).
for ($V_i$, $i$=1,2,$\cdots$,$M$) do
  $X_i^{(N)} \leftarrow X_{i-1}$ /*Preceding Neighbor Fitting*/
  $X_i^{(L)} \leftarrow 2X_{i-1} - X_{i-2}$ /*Linear-Curve Fitting*/
  $X_i^{(Q)} \leftarrow 3X_{i-1} - 3X_{i-2} + X_{i-3}$ /*Quadratic-Curve Fitting*/
  $bestfit\_sol = \underset{Y=\{(N),(L),(Q)\}}{\arg\min} (|X_i^Y - V_i|)$.
  if ($|X_i^{bestfit\_sol} - V_i|$ meets error bounds) then
    switch ($bestfit\_sol$)
    case ($N$):
      $\upsilon[i]$ = $01_{(2)}$ /*denote Preceding Neighbor Fitting*/
    case ($L$):
      $\upsilon[i]$ = $10_{(2)}$ /*denote Linear-Curve Fitting*/
    case ($Q$):
      $\upsilon[i]$ = $11_{(2)}$ /*denote Quadratic-Curve Fitting*/
    end switch
    $X_i \leftarrow X_i^{bestfit\_sol}$/*record the predicted value for next prediction*/
  else
    $\upsilon[i]$ = $00_{(2)}$ /*denote unpredictable data*/
    Compress $V_i$ by binary-representation analysis.
    $\rho[j{+}{+}] \leftarrow V_i'$ /*$V_i'$ is the binary decompressed value of $V_i$*/
    $X_i \leftarrow V_i'$
  end if
end for
```

Fig. 13. Pseudocode of the SZ operation

then in the case of compressors of algorithms for encoded compression of floating-point numbers, namely: BitGrooming, BitShaving, DigitRouting during compression, the input data passes through the algorithm to get rid of false accuracy, then floating-point numbers are compressed using DeflaterUtils by the Deflate algorithm and the output is just an array of bytes, in in the case of unpacking, the byte array simply passes through the DeflaterUtils methods and turns into a floating-point scale with lost false precision using the Inflate algorithm. In the case of compressors of algorithms for encoded compression of the entire set of numbers (SZ, FPC) during compression, the input array of numbers after processing by the algorithm returns the encoded byte array and then the byte array is compressed using Deflate. During decompression, the Inflate algorithm is used and the compressor algorithm itself decodes the data into floating-point numbers. In the case of the K2Raster algorithm, the compressor directly compresses and decompresses data using it, and DeflaterUtils is used solely for the purpose of calculating compression coefficients.

The DeflaterUtils class also plays an important role in the operation of all compressors, which includes calculating the compression ratio and working with the zip data compression library, in particular with the Deflate and Inflate algorithms, which are used to compress encoded data, however, since K2Raster is a compression algorithm, not an encoding algorithm, its compressor does not use Deflater and Inflater. K2RasterCompressor uses DeflaterUtils exclusively to calculate the data compression ratio.

3 Analysis of Obtained Results

After the program is running, we get a CSV file with the results of all algorithms on all input data, as a result, we got more than a hundred thousand measurements of the work of all algorithms on different data. Then we process

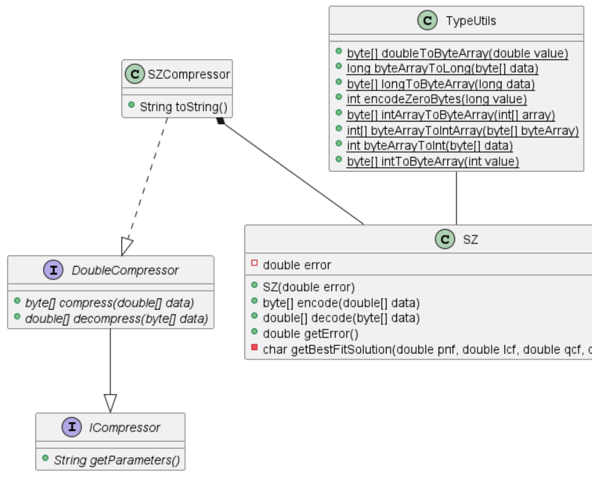
TypeUtils
byte[] doubleToByteArray(double value)
long byteArrayToLong(byte[] data)
byte[] longToByteArray(long data)
int encodeZeroBytes(long value)
byte[] intArrayToByteArray(int[] array)
int[] byteArrayToIntArray(byte[] byteArray)
int byteArrayToInt(byte[] data)
byte[] intToByteArray(int value)
SZCompressor
String toString()
SZ
double error
SZ(double error)
byte[] encode(double[] data)
double[] decode(byte[] data)
double getError()
char getBestFitSolution(double pnf, double lcf, double qcf, double value)
DoubleCompressor
byte[] compress(double[] data)
double[] decompress(byte[] data)
ICompressor
String getParameters()

Fig. 14. UML diagram of the SZ algorithm

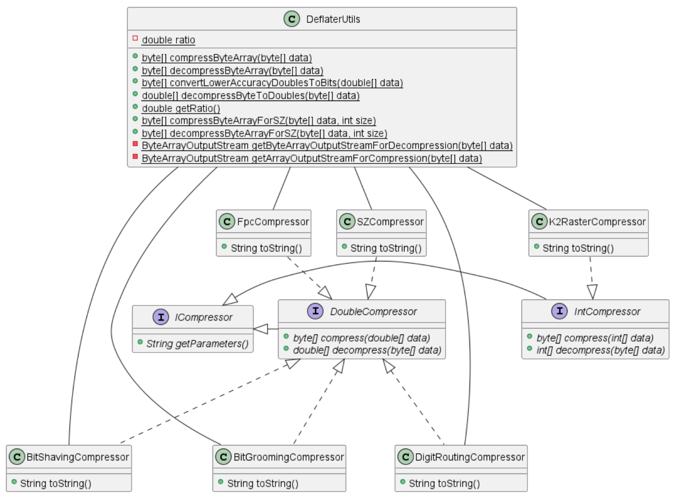
DeflaterUtils
double ratio
byte[] compressByteArray(byte[] data)
byte[] decompressByteArray(byte[] data)
byte[] convertLowerAccuracyDoublesToBits(double[] data)
double[] decompressByteToDoubles(byte[] data)
double getRatio()
byte[] compressByteArrayForSZ(byte[] data, int size)
byte[] decompressByteArrayForSZ(byte[] data, int size)
ByteArrayOutputStream getByteArrayOutputStreamForDecompression(byte[] data)
ByteArrayOutputStream getArrayOutputStreamForCompression(byte[] data)
FpcCompressor
String toString()
SZCompressor
String toString()
K2RasterCompressor
String toString()
ICompressor
String getParameters()
DoubleCompressor
byte[] compress(double[] data)
double[] decompress(byte[] data)
IntCompressor
byte[] compress(int[] data)
int[] decompress(byte[] data)
BitShavingCompressor
String toString()
BitGroomingCompressor
String toString()
DigitRoutingCompressor
String toString()

Fig. 15. UML diagram of compressor connection with DeflaterUtils

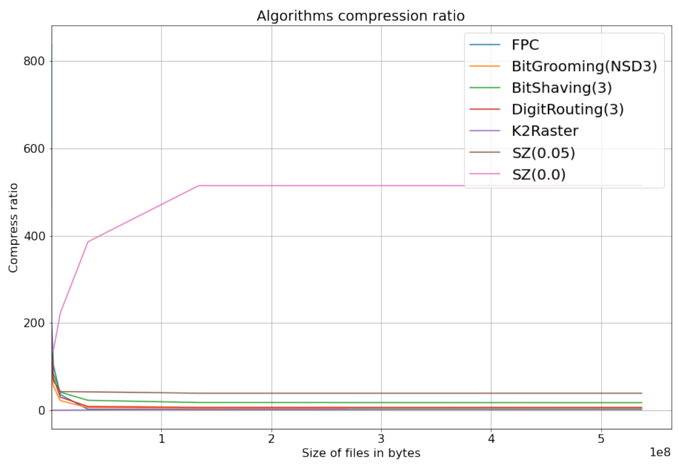

Fig. 16. Graph of the dependence of the compression ratio on the amount of data for all algorithms

Next, consider the dependence of compression time on the amount of data.

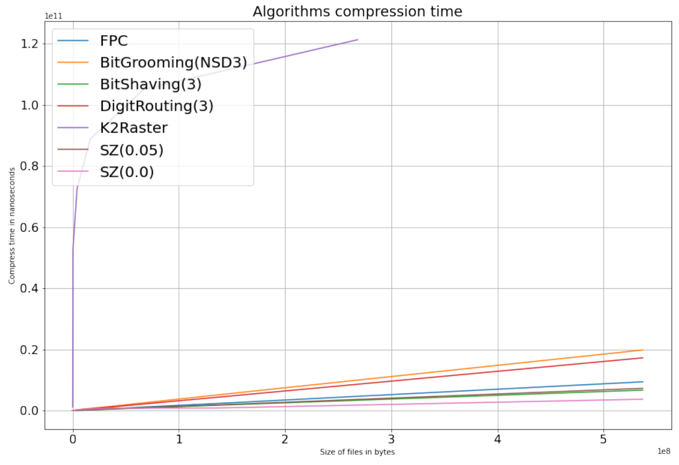

Fig. 17. Graph of compression time versus data volume for all algorithms

them using Python libraries and get graphs to analyze the algorithms. The parameters of lossy compression algorithms selected on the general graphs are due to the specifics of the NDVI data, since the numbers after the third decimal place have no applied physical meaning and can be discarded, therefore, the parameters that best satisfy this fact were selected. These parameters do not allow you to lose unnecessary information, but at the same time they will allow you to compress data as efficiently as possible.

Let's consider the obtained results of all algorithms:

Consider the graph of the dependence of the compression ratio on the amount of data.

A graph with such large data is not so easy to analyze, however, the best compression coefficients show SZ graphs with and without losses. For SZ(0.0), this result is due to the fact that with zero error, bits are quantized in the binary representation of a number by subtracting the median average value from all numbers, and predictors almost do not work, as a result of which it can be observed that the Deflate algorithm applied to the encoded data compressed the input data as efficiently as possible from the presented algorithms.

As we can see in Fig 17. the K2Raster algorithm showed the worst results on this graph, and the length of its curve along the axis showing the size of the data can be explained by the fact that for all algorithms, the maximum data size was 8192x8192 images, however, for all algorithms except K2Raster, Float64 floating point values were used, and for K2Raster images with UInt16 values, this is due to the that the rest of the algorithms are designed to compress floating-point numbers, and K2Raster for positive integers. Also, the data types in the pictures are determined by the GeoTIFF format.

Next, consider the dependence of the unpacking time on the amount of data.

It is also seen here that the K2Raster algorithm shows the worst result to the point that it is not possible to analyze other algorithms, for a more correct assessment, the K2Raster algorithm will be excluded from the following graphs, and the size of satellite photos will be reduced to 4096x4096, since the behavior of the algorithms after this mark remains the same. As follows from the analysis of Fig. 18.

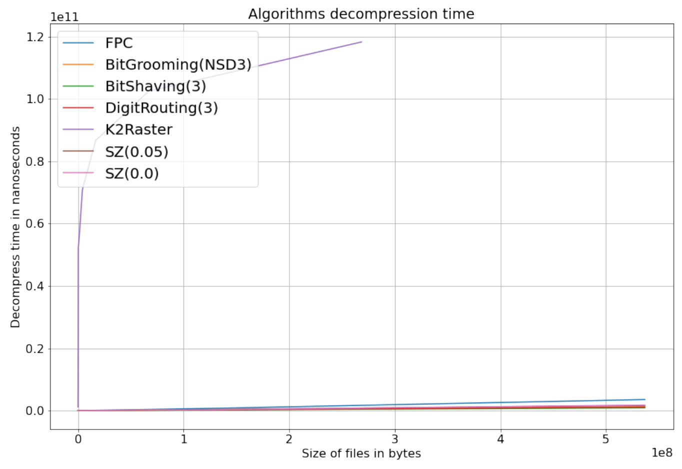

Fig. 18. Graph of the dependence of the unpacking speed on the amount of data for all algorithms

Now also consider all the algorithms at once, namely graphs of compression coefficients, compression rates and data decompression.

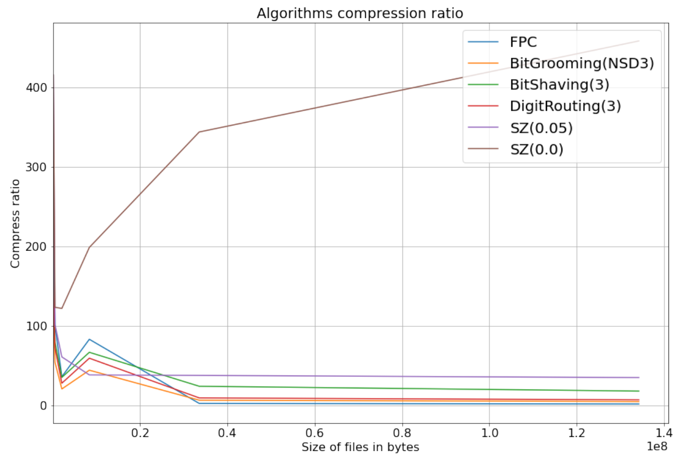

Fig. 19. Graph of the dependence of the compression ratio on the amount of data without K2Raster

So we can observe on Fig. 19 a more interesting picture, we can see the obvious superiority of SZ(0.0) over other algorithms due to bit quantization. An interesting result is shown by other algorithms in photos ranging in size from 128x128 to 1024x1024. For example, in the pictures 512x512 FPS shows the best result, except for SZ(0.0), but on average the SZ algorithm holds the championship regardless of the parameters.

SZ(0.0) shows on Fig.20 an unusual curve, this is due to the quantized bits and the compression feature of their Deflate algorithm. The BitShaving algorithm is so fast due to its prostate and the usual bit-cutting of the number.

On Fig. 21 the BitShaving algorithm here shows the highest speed due to its simplicity and simplicity. And FPC is the worst time since it uses predictors not only for compression, but also for decompressing data, unlike SZ.

Now we will separately consider the results of lossy compression algorithms, since among lossless compression algorithms, the obvious leader is SZ(0.0).

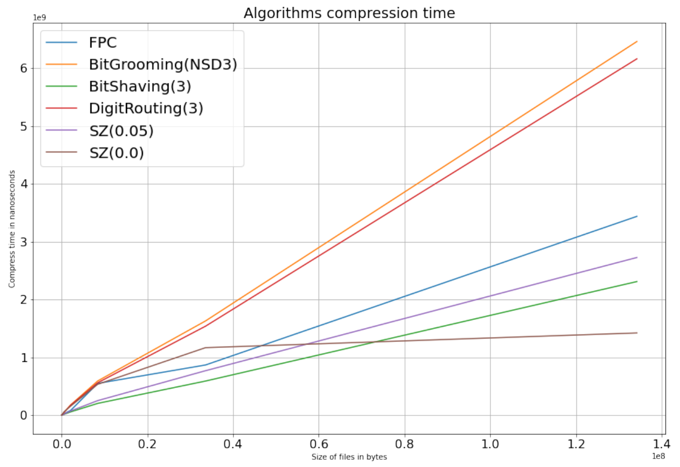

Fig. 20. Graph of compression time versus data volume without K2Raster

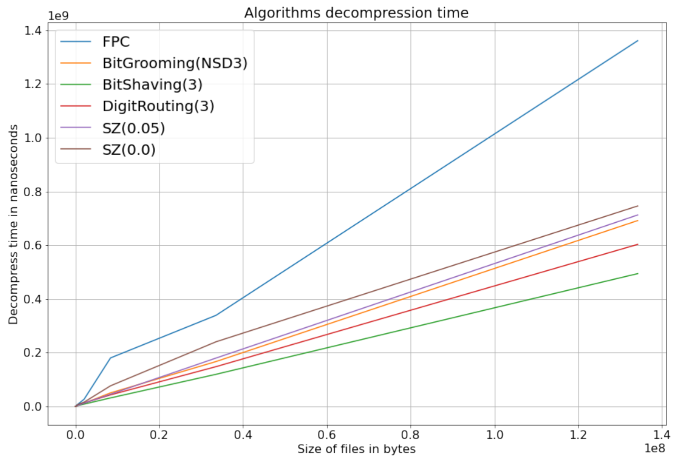

Fig. 21. Graph of the dependence of the unpacking speed on the amount of data without K2Raster

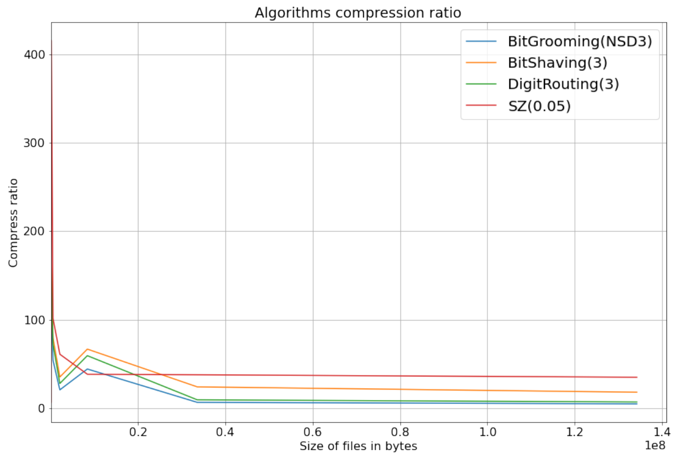

Fig. 22. Graph of the dependence of the compression ratio on the amount of datafor lossy compression algorithms

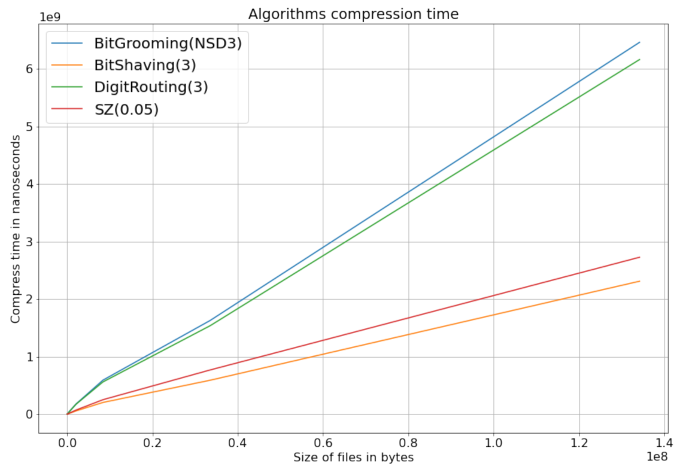

Fig. 23. Graph of compression time versus data volume for lossy compression algorithms

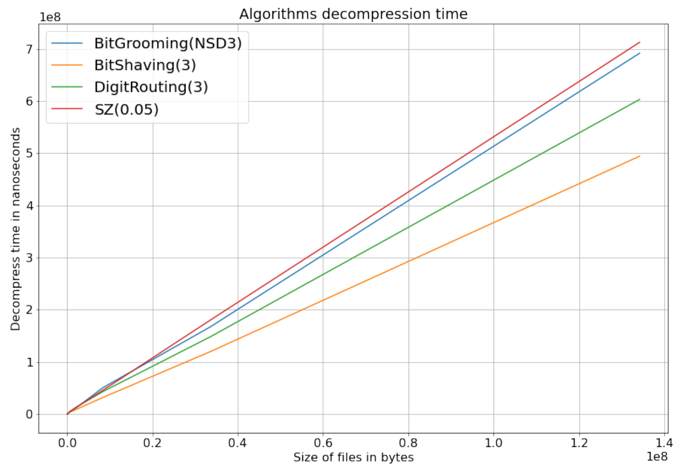

Fig. 24. Graph of the dependence of the unpacking speed on the amount of data for lossy compression algorithms

According to these figures 22, 23, 24, it can be seen that the SZ(0.05) algorithm is superior to others in terms of decompression speed and compression ratio, however, it, like BitShaving, does not maintain accuracy well and such a situation may happen on certain data that it can even change the rank of tens. BitGrooming and Digits Routing are much better at preserving the uniformity of data and their physical meaning, since lossy compression goes with maximum preservation of the accuracy of numbers. Therefore, based on the data obtained and the importance of maintaining approximate accuracy, DigitRouting can be called the best lossy compression algorithm, since it surpasses BitGrooming in decompression speed and compression ratio.

4 Conclusion

Summing up, we can conclude that modern raster data compression algorithms have something to strive for. Among lossless compression algorithms, SZ showed the best result, despite the fact that it is older than FPC and K2Raster. Which is not surprising, since in the case of lossless compression, the SZ algorithm tends to bring all numbers closer to zero by median quantization of bits. This leads to the fact that when the encoded data is compressed by the Deflate algorithm, a lot of repetitions appear, which makes it possible to compress data extremely efficiently. This is largely due to the specifics of the NDVI source data, since floating-point numbers in this case take values from -1 to 1 and the process of median quantization of data shows a truly impressive result. The FPC algorithm, in turn, is equally effective for any floating-point data of the Float64 format, this is due to the fact that it uses predictors with hash tables in the case of which the difference in the value of the numbers does not matter. So, for example, in the case of data where most of it will have a large value of integer parts of a number, and the other part tends to zero SZ without rubbing can show a compression ratio even lower than 1. K2Raster showed such poor results because K2Raster is part of the K3Raster algorithm, which was designed to compress three-dimensional raster temporal geo-data, but it was not considered in this study. Also, initially, when developing K2Raster, emphasis was placed on compressing UInt8 numbers, and in the case of UInt16, it shows high efficiency with data with a small value, but images from the Landsat 8 satellite do not have this feature. Among lossy compression algorithms, DigitRouting showed the best results in terms of aggregate compression speed, compression ratio, and the ability of the algorithm to maintain maximum accuracy while encoding data for compression as efficiently as possible was also taken into account. However, SZ(0.0) exceeds both the compression rate and the decompression rate, as well as the average compression ratio. Therefore, if, for example, an application developer requires the maximum allowable preservation of the accuracy of floating-point data, it is preferable to use DigitRouting, since SZ(0.0) will not show such high results in the case of large data entropy. If the application does not require high-quality preservation of accuracy, then you can use the SZ algorithm with error parameters, since it is faster both during compression and during decompression. Also, if it is possible to neglect the maximum accuracy margin and if you need to store raster spatial geodata, you should use the SZ algorithm with error parameters or BitShaving, since it shows the highest data compression ratio among lossy compression algorithms.

Consider the table of average compression coefficients for all algorithms:

TABLE 2
Table of the average compression ratio for all algorithms

	FPC	SZ(0.0)	K2R	SZ(0.05)	DR(3)	BG(NSD3)	BS(3)
Average compression ratio	119.31	238.7	0.88	83.79	88.91	67.37	97.36

The table only confirms the above conclusions. The FPC algorithm showed a very high average compression ratio, which shows its effectiveness in comparison with SZ(0.0) on data with high entropy. Ultimately, the choice of the algorithm should be made based on the task at hand. So, for example, considering the situation with DVI, the logical conclusion would be the choice of SZ(0.0), since it not only preserves accuracy, but on big data it works even faster than lossy compression algorithms. If the decisive factor is speed and there is no need to transmit data with high accuracy, and the data is very small, i.e. images smaller than 256x256, then BitShaving should be preferred. In general, when it is assumed that data can have high entropy, then in the case of lossless compression, the FPC algorithm will be preferred, in the case of lossy compression, everything again depends on the importance of preserving accuracy as much as possible, if this is important, then our choice is DigitRouting, otherwise BitShaving since it the fastest and has the highest compression ratio among algorithms with parmeters that allow you to maintain accuracy up to the third decimal place.

In the future, a good solution would be to compare algorithms on highly entropic data for a more accurate assessment of the conclusions regarding the FPC algorithm, even if it was developed specifically for such situations. Also expand the list of algorithms very popular such as fpZip, ZFP and possibly MGARD. It is also worth implementing K3Raster for a more objective assessment of the feasibility of the existence of K2Raster.

References

[1] [Online]. Available: https://www.researchgate.net/publication/228450765_Temporal_Analytics_on_Big_Data_for_Web_Advertising
[2] *IEEE Standard for Floating-Point Arithmetic*, vol. 29, 2008.
[3] [Online]. Available: https://iopscience.iop.org/article/10.1088/1755-1315/421/4/042001/pdf
[4] [Online]. Available: https://courses.lumenlearning.com/geophysical/chapter/geospatial-technology/
[5] [Online]. Available: https://www.gislounge.com/geodatabases-explored-vector-and-raster-data/
[6] [Online]. Available: https://www.researchgate.net/publication/323313402_Building_a_local_spatial_data_infrastructure_SDI_to_collect_manage_and_deliver_coastal_information
[7] [Online]. Available: https://www.nationalgeographic.org/encyclopedia/map/
[8] A. H. Baker, H. Xu, J. M. Dennis, M. N. Levy, D. Nychka, and S. A. Mickelson, "A Methodology for Evaluating the Impact of Data Compression on Climate Simulation Data," *The ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC14)*, pp. 203–214, 2014.
[9] S. Galati, 2006.
[10] Y.-J Chen, N. Chiang, and Memon, pp. 124–126, 2005.
[11] [Online]. Available: http://www.gzip.org
[12] Gislab and Gis. [Online]. Available: https://gis-lab.info/qa/ndvi.html
[13] *GIS Reference Books and Materials Caitlin Dempsey*, 2015.
[14] M. Burtscher and P. Ratanaworabhan, "High Throughput Compression of Double-Precision Floating-PointData," CSL-TR-2006-1047, 2006.
[15] C. S. Zender, "Bit grooming: Statistically accurate precision-preserving quantization with compression," *Geoscientific Model Development*, vol. 9, pp. 3199–3211, 2016.
[16] ——, "Analysis of self-describing gridded geoscience data with netCDF Operators (NCO)," *Environ. Modell. Softw*, vol. 23, no. 10, pp. 1338–1342, 2008.
[17] [Online]. Available: https://www.unidata.ucar.edu/blogs/developer/en/entry/compression_by_bit_shaving
[18] J. Caron, pp. 27–27, 2014. [Online]. Available: http://www.unidata.ucar.edu/blogs/developer/entry/compression_by_bit_shaving(lastaccess
[19] F. Silva-Coira, J. R. Paramá, G. Bernardo, and D. Seco, "Space-efficient representations of raster time series," *Information Sciences*, vol. 566, pp. 300–325, 2021.
[20] S. Ladra, J. R. Paramá, and F. Silva-Coira, "Compact and queryable representation of raster datasets," *Proceedings of the 28th International Conference on Scientific and Statistical Database Management*, 2016.
[21] N. R. Brisaboa, S. Ladra, and G. Navarro, "Compact representation of web graphs with extended functionality," *Information Systems*, pp. 152–174, 2014.
[22] N. R. Brisaboa, A. G. Brandon, G. Navarro, and J. R. P. a, "GraCT: A Grammar-based Compressed Index for Trajectory Data," *Information Sciences*, vol. 483, pp. 106–135, 2019.
[23] N. R. Brisaboa, S. Ladra, and G. Navarro, pp. 392–404, 2013.
[24] X. Delaunay, A. Courtois, and F. Gouillon, "Evaluation of lossless and lossy algorithms for the compression of scientific datasets in NetCDF-4 or HDF5 files," *Geoscientific Model Development*, vol. 12, no. 9, pp. 4099–4113, 2019.
[25] S. Di and F. Cappello, "Fast error-bounded lossy HPC data compression with SZ," *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2016.
[26] J. D. Silver and C. S. Zender, "The compression-error trade-off for large gridded data sets," *Geosci. Model Dev*, vol. 10, pp. 413–423, 2017.
[27] *IEEE Standard for Floating-Point Arithmetic*, 2008.
[28] U. S. G. S, - U S G Survey, Earthexplorer, and Earthexplorer. [Online]. Available: https://earthexplorer.usgs.gov/