COMPARISON OF OPEN SOURCE COMPRESSION ALGORITHMS ON VHR REMOTE SENSING IMAGES FOR EFFICIENT STORAGE HIERARCHY

A. Akoguz[a], S. Bozkurt[a], A. A. Gozutok[a,*], G. Alp[a], E. G. Turan[b], M. Bogaz[a], S. Kent[c]

[a] Center for Satellite Communications and Remote Sensing, ITU, Istanbul, Turkey - (alper, sadik, armagan, gulsah)@cscrs.itu.edu.tr
[b] Department of Geophysical Engineering, ITU, Istanbul, Turkey - turanel@itu.edu.tr
[c] Department of Electronics and Communication Engineering, ITU, Istanbul, Turkey - kents@itu.edu.tr

Commission IV, WG IV/1

KEY WORDS: Lossless Data Compression, LZMA, LZO, BWT, PPMd, GeoTIFF, VHR, SPOT, open source

ABSTRACT:

High resolution level in satellite imagery came with its fundamental problem as big amount of telemetry data which is to be stored after the downlink operation. Moreover, later the post-processing and image enhancement steps after the image is acquired, the file sizes increase even more and then it gets a lot harder to store and consume much more time to transmit the data from one source to another; hence, it should be taken into account that to save even more space with file compression of the raw and various levels of processed data is a necessity for archiving stations to save more space. Lossless data compression algorithms that will be examined in this study aim to provide compression without any loss of data holding spectral information. Within this objective, well-known open source programs supporting related compression algorithms have been implemented on processed GeoTIFF images of Airbus Defence & Spaces SPOT 6 & 7 satellites having 1.5 m. of GSD, which were acquired and stored by ITU Center for Satellite Communications and Remote Sensing (ITU CSCRS), with the algorithms Lempel-Ziv-Welch (LZW), Lempel-Ziv-Markov chain Algorithm (LZMA & LZMA2), Lempel-Ziv-Oberhumer (LZO), Deflate & Deflate 64, Prediction by Partial Matching (PPMd or PPM2), Burrows-Wheeler Transform (BWT) in order to observe compression performances of these algorithms over sample datasets in terms of how much of the image data can be compressed by ensuring lossless compression.

1. INTRODUCTION

Remote sensing has been an essential approach for the Earth observation from the beginning of space age. Since the space technology and complex space-borne sensor systems are still being developed rapidly, we can reach sub-meter spatial resolutions and wider swath width with satellite data acquired nowadays. Moreover, big data definition is getting more and more popular since the invention of more complex data acquisition systems and generation enormous amount of data from the beginning of digital era. One main part of remote sensing is that every image sensed by high resolution sensors occupies hundreds or thousands of megabytes in storage when it is in main processing level which is described as raw data. When we consider the amount of image acquired by remote sensing satellites each day and going further with downlinking the acquired satellite data to ground station in near-realtime, this amount is sure to be grown up to terabytes and then petabyte (10^{15} Bytes) scale in short time interval when hundreds of individual image is recorded over a single day. Furthermore, this may lead to insufficient storage area for ground receiving stations operating with these high resolution earth observation satellites after short period of time.

Within this advancing volume of data the challenge becomes more complicated that one should ask how to store this vast amount of data more effectively in receiving stations and how to transfer this data over a web structure in shorter time intervals? In order to avoid these kind of problems that the solution may be simplified as more storage area can be satisfied by having larger storage capacity; however, lossless data compression has being a great importance for storage of imagery as getting complicated remote sensing technology. On the other hand, preserving original quality of an image after compression and decompression operations play the most crucial role for data modeling, for example classification and feature extraction applications needs the spectral data as it is for generating more precise models. Therefore, lossless data compression should be preferred to compress remote sensing data for archiving in order to keep their information unchanged and then retrieve the original information back. In addition, data compression algorithms consist of three main categories which are lossy compression, near-lossless compression and lossless compression which will be described in algorithms and methods section.

Data compression concept firstly appeared on the fields of information theory (Shannon, 1948). The main concept behind data compression was generation of probability relations through data source. Theory behind data compression is achieving the elimination of redundant information by using an encoder and to fix up the initial information back in more compressed sizes to reduce initial volume of data. Inside the context of remote sensing, some image file formats that are standardized as .TIF (Tagged Image File), .img etc. are the main file formats that are used appropriately to store today. In order to compress large .TIF formatted files, one must consider having implemented lossless compression and decompression on their data to protect the original image information for further analysis.

Tagged Image File Format (TIFF) is one of the well-known raster image archiving file format around the world. One of the application areas of TIF format is producing and monitoring the raster image data geographically for various types of images including remote sensing data (Ritter and Ruth, 1995). Many Geographical Information System (GIS) softwares support TIF format as their user demands improved so far. The GeoTIFF is specified between the tags that are compatible with existing GIS software.

As we consider lossless data compression, we will investigate

*Corresponding author

open-source compression algorithms and programs which are supported by storage architecture to observe how fast and how much compression can be satisfied by to store remotely sensed data more efficiently over archiving architecture. Compression algorithms that are implemented on sample 16 bit-depth .TIF datasets are Deflate & Deflate64, Lempel-Ziv-Welch (LZW)), Lempel-Ziv-Markov Chain Algorithms (LZMA & LZMA2), Burrows & Wheeler Transform (BWT), Prediction by Partial Matching II (which is called shortly as PPMd) & Lempel-Ziv- Oberhumer (LZO). These algorithms will be defined in the following in basics.

2. METHODS & ALGORITHMS USED

2.1 Deflate

Deflate is a widely known compression algorithm which was developed by Philip Katz and initially used in PKZip program written by himself and further used in Gzip software. The algorithm was also implemented on zlib library, which is a portable and open-source compression library for lossless data compression (Salomon et al., 2010). The deflate is an algorithm based on LZ77 and Huffman code combination that tries to locate duplicate strings from the input information which are the repetition of the sequences, then replaces all the secondary situations where the characters exists by a pointer to retrieve previous information in archive. While locating and encoding the sequences from the input stream, Huffman code is also applied for entropy coding. Deflate algorithm is an essential compression part and base of Gzip and PigZ programs and also supported by 7z. PigZ program uses available threads in parallel by breaking the input data up to 128 KB chunks.

Theory behind deflate algorithm is basically a sliding window that stores the record of information about the characters which went before, for example, a 16K sliding window means that the last 16384 (16*1024) characters are recorded by compressor while archiving and decompressor while extracting. From here, when the next character stream or sequence is going to be compressed, it is retrieved from the sliding window, which stores the location of occurrences as pointers. The sequence is then changed to two integer numbers, a distance and a length information. The distance means that there is a further way from where the sequence starts into sliding window, the length means the quantity of characters from the sequence. The sliding window can be set up to 32K for this algorithm (Sayood, 2002). Deflate64 algorithm is modified version of deflate and dictionary size is extended to 64K to store more symbols in window while faster computational operations. Program release versions of Pigz and Gzip are 2.3.3 & 1.5 relatively.

2.2 Lempel-Ziv-Welch (LZW)

Another compression algorithm based on LZ77 is LZW, the algorithm is basically constructed around a translation table which consists of strings, the algorithm tries to find all of the common substrings and puts a variable size code where they exist and makes them assigned to the generated table which contains previous strings that have been encountered in the message that is to be compressed (Welch, 1984). In this research, a simple LZW algorithm that is implemented on default compression program of CentOS $compress$ was used for lossless compression of sample datasets.

2.3 Lempel-Ziv-Markov Chain Algorithm (LZMA-LZMA2)

Lempel and Ziv's consecutive compressors are the base of mostly practiced and used lossless compression algorithms (Salomon et al., 2010). Lempel-Ziv-Markov Chain Algorithm (which is abbreviated as LZMA) is the main algorithm running behind programs like 7z (or 7-zip) and XZ to compress data using a dictionary based compression scheme to achieve better compression ratios and faster decompression intervals. First thoughts of adaptive dictionary based compression algorithms are put forth by Abraham Lempel & Jacob Ziv and come out one as an algorithm in 1977 (LZ77) & another in 1978 (LZ78). Then the LZMA algorithm was created by Igor Pavlov and further developed since it is released. It is an improved and optimized version of LZ77 and its compression procedure has similarities between deflate algorithm however, it uses range encoding which is a variant of arithmetic coding rather than Huffman coding (Ziv and Lempel, 1977). As a result, encoder becomes more complex while better compression ratios are being achieved. This high compression ratio levels of LZMA basically provided by two computation concepts, sliding dictionaries or windows and Markov models. First phase of compression process is Lempel-Ziv coding which locates and reduces the redundancy by transforming data chunks to distance-length couples. Then the second phase comes which is a range encoding process that uses distinctive probability models for different samples data.

Considering the 7z program, the dictionary size can be set up to 4 Gb, however it is currently limited to 1 Gb to implement on. Also multi-processing is supported if hardware is appropriate for parallel compression on CPU. However, multi-threading is not supported for decompression. LZMA2 algorithm is a modified and improved version of LZMA which provides better multithreading results (Pavlov, 1999). 7zip & XZ programs support both LZMA and LZMA2 algorithms with different archive types as .7z and .xz. 7z software release used is this research was 9.20 (64-bit), Xz version was 5.1.2alpha.

2.4 Burrows-Wheeler Transform (BWT)

The Burrows-Wheeler transform (BWT) (Burrows and Wheeler, 1994) is a lossless data compression method which is basically a block-sorting algorithm. The transformation part does not achieve any compression but it changes the input data to make it more simple to compress input data easily, this will lead to significant increase in compression times. The BWT does not compress data sub-sequentially however, it divides the input stream to blocks as units. These units are consisting in the initial block's characters in different sort.

BZIP2 is an open-source lossless data compressor which uses Burrows-Wheeler algorithm and Huffman coding in compression background. In earlier times when bzip was developed, arithmetic coding was used more effectively, however it was reduced due to patent problems of arithmetic coding. The compression results seeming to be better and way more faster than LZ77-LZ78 based compressors in general and nearly on the path of performance level where PPM sort of statistical compressors stand (Gilchrist, 2004).

Linux based bzip2, pbzip2, which is parallel-bzip2, and 7zip softwares are the programs that are consisting in Burrows-Wheeler Transform within their source code. Compression parameters for the programs are file block size that reduces file into chunks, is variable from 100 KB to 900 KB, number of threads (for pbzip2 and 7z) and compression level that can be set from 0 to 9. Pbzip2's release version was v1.0.5 while Zip's version 3.0.

2.5 Prediction by Partial Matching (PPMd)

Prediction by partial matching (PPM) algorithm is adaptive statistical lossless data compression method based on modeling and

prediction, created by Cleary and Witten in 1984 (Cleary and Witten, 1984). The models basically predicts the next character in stream using a set of preceding symbols. First PPM algorithms have had higher memory usage while having longer compression intervals. These algorithms are well-known state of art compressors for lossless data compression (Mahoney, 2005). An algorithm called PPMd (or PPM2) was developed by D. Shkarin from PPM structure and having high compression ratios for all types of data. PPMd algorithm is supported by 7zip open-source software. In general, PPM algorithm basically encodes the input stream by using the longer context patterns where the previous symbols have appeared within defined model order in size (Howard, 1993).

PPM method is basically predicting the probability distribution of the *n*th symbol considering previous *k* symbols x_{n-k}, ... , x_{n-1}. Parameter *k* here is the maximum size or the order for the model up to $(n-k)^{th}$ symbol to predict *n*th symbol. PPMd parameters tested with 7z software are memory size in format as 2^{size} (2^{26}-2^{27}-2^{28} Bytes), prediction model order (orders selected 2-8-16) and compression level that differs from 0 to 9.

2.6 LZO (Lempel-Ziv-Oberhumer)

LZO is another modification of Lempel-Ziv 1977 (LZ77), compressing data with sliding window scheme, mainly optimized to use integer arithmetic in computer hardware to gain advantages of computational architecture. Since that floating point operations, which takes more time to compute (Kane and Yang, 2012), generate higher memory usage and consume more time, usage of integer arithmetic should avoid this kind of problems, however it has a performance trade-off which results in achieving lower compression ratios. There are implemented examples of LZO in various technologies like NASA's Mars Rovers(Oberhumer, 1996). Linux based lzop software has suitable LZO libraries to implement the algorithm on our dataset. Lzop release used in compression tests was v1.03, Nov 1st 2010.

3. IMPLEMENTATION

To begin with implementation of the algorithms on sample .TIF datasets, initially TIF products were generated from different test sites chosen from various land cover types as sea, which was chosen from Marmara region, settlement (chosen from urban sites in Ankara), forest site taken from Zonguldak region where dense forest flora exist and agricultural site selected from Konya plain. Quicklook files can be found in Figure 1 & Figure 2 for pansharpened (PMS) and panchromatic (PAN) products, for multispectral (MS) datasets, sample quicklook display looks same as PMS. Also product file size parameters can be retrieved from the Table 1 for generated sample TIF data. For SPOT satellite imagery products, there are two different product processing levels which are defined as primary (Pri) and orthorectified (Ort). The product level primary is basically described as the processing level which is closer to the original image that is acquired by sensor and calibrated radiometrically. After performing the resampling of the acquired image according to the projection mapping, which is previously calculated on ground-projected images, is the method called ortho-rectification (Leprince et al., 2007).

SPOT6 & 7 products used in this study provide natural color, 1.5 m spatial resolution imagery as standard. These Earth observation satellites have five spectral bands that are red, green, blue, near-infrared and panchromatic between the range of 0.450 and 0.890 micrometer of electromagnetic spectrum. SPOT6 & 7 imagery products are collected along 60 kilometer swath width and provides mapping at the scale of 1:25000. In this research, 16 bit encoded multispectral (MS) with 6 meter spatial resolution, panchromatic (PAN) with 1.5 meter spatial resolution and pansharpened (PMS which is defined as MS + PAN by image fusion technique) having 1.5 meter spatial resolution, generated as primary and orthorectified geotiff imageries have been selected from agricultural, settlement, sea and forestry areas in order to evaluate compression performance of deflate, BWT, PPMd, LZW, LZMA & LZMA2, LZO algorithms. These imageries tiled to approximately 20x20 km width while being produced. Additionally, these geotiff data have different sizes depending on spectral bands information about the selected locations.

Product #	MS	PAN	PMS
Agriculture (Ort)	86828830	347230894	1388921934
Agriculture (Pri)	78712486	314896716	1259586062
Forest (Ort)	80288606	320998116	1283990822
Forest (Pri)	58291990	233208004	932831214
Sea (Ort)	82663070	330570330	1322279678
Sea (Pri)	57170758	228722692	914889966
Settlement (Ort)	81765318	326903162	1307611006
Settlement (Pri)	54522006	218126884	872506734

Table 1: File sizes of different land cover datasets (Bytes).

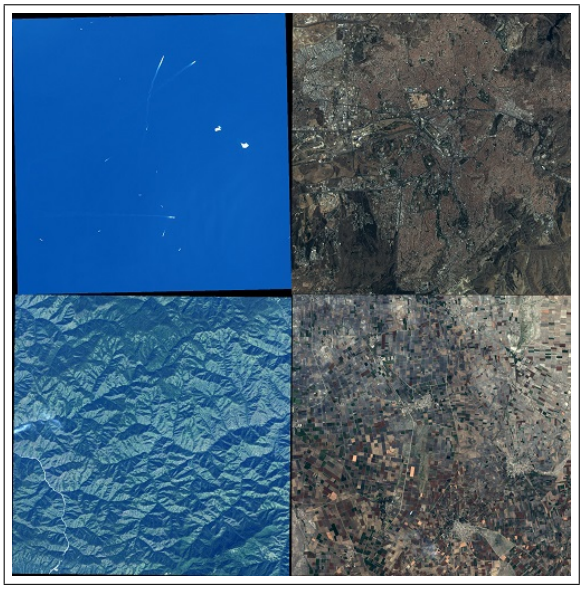

Figure 1: Sample quicklook display of pansharpened SPOT 6 & 7 images (upper left: sea, upper right: settlement, lower left: forest, lower right: agriculture) (PMS). © SPOT Image 2015, CNES.

In algorithm application section, compression parameters typically have some specific inputs which differ among compression programs that are tested. Considering the file compression, the main program input parameters can be listed as compression method, compression level, output archive type, dictionary size, file block size, number of threads to use in parallel processing, model order (only for PPMd implemented by 7z). For decompression of the compressed archive, there are no parameters used except execution of extraction process. All of the compression input parameters used in this study are displayed on Table 2. At that point to evaluate the compression performance, compression ratio is defined as $1-(output\_size/input\_size)$ which allows us to decide how much of the input data is compressed and how can we get free space after compression procedure.

$$Compression\ Ratio = 1 - \frac{Output\ size}{Input\ size} \qquad (1)$$

This contribution has been peer-reviewed.
doi:10.5194/isprsarchives-XLI-B4-3-2016

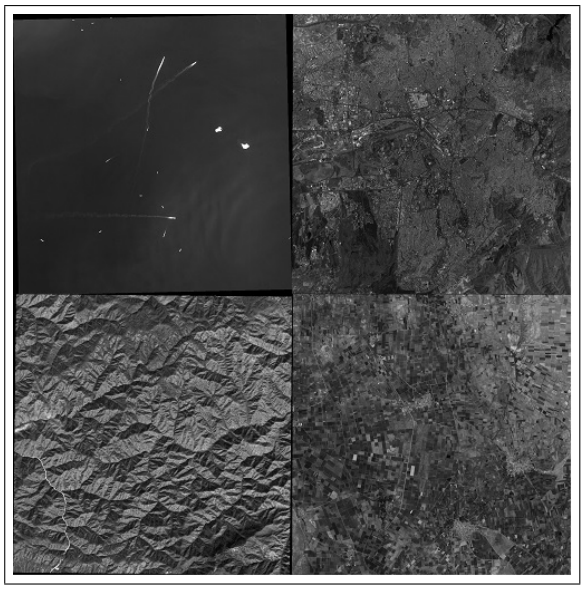

Figure 2: Sample quicklook display of panchromatic SPOT images (upper left: sea, upper right: settlement, lower left: forest, lower right: agriculture) (PAN). © SPOT Image 2015, CNES.

Test hardware specification includes Intel®Xeon®E3-1225 V2 model 58 @3.2 GHz - 4 core / 8 thread CPU, 8 GB 1333 MHz DDR3 RAM, 240 GB (520 MB - 350 MB/s read & write speed) SATA3 SSD hard disk and generated scripts & codes that were implemented on operating system CentOS Linux release 7.1.1503 (Core), kernel version 3.10.0-229.el7.x86_64.

4. RESULTS

The results of compression-decompression scripts written in bash (bourne-again shell) having input parameters listed in Table 2 will be discussed in this section. Before that discussion, in order to determine the compression performance relatively among with the programs and algorithms implemented on sample TIF datasets, we have and should describe a parameter called compression efficiency, E_{comp}. This parameter was basically derived from the compression ratio which is achieved at the end of archiving and the time (T_{comp}) which was spent during compression. We have found this parameter as dividing the achieved ratio by time elapsed.

$$E_{comp} = Compression\ Ratio/T_{comp} \quad (2)$$

According to the results obtained, typically the fastest compression scenario was proven to be compression level 3 on our sample imagery. Considering the speed of compression, LZOP have achieved its best compression efficiency of 110.441 while having a ratio 24.0662. Also Deflate algorithm implemented by Pigz has proven to be the most efficient algorithm by having 179.525 of efficiency while getting 48.04 compression ratio in a very short amount of time. This result was achieved by multi-threading of 8 threads by Pigz. Another multi-threading supporter Pbzip2 program achieved a maximum efficiency of 53.7075 by getting 62.5972 of compression ratio and the last program that was used for multi-threaded approach Xz was outperformed by other parallel compressors that it could only get an efficiency around 7. More detailed compression efficiency results can be retrieved from Table 3.

In terms of multi-threading support of the programs 7z, XZ, Pbzip2 & Pigz, our testbed workstation CPU usage have risen up to 800% while compressing the datasets with number of threads to compress input data were up to 8 to achieve much shorter compression intervals. In addition, more complex TIF data which consists large amount of land cover that consisting urban architecture and complicated structures took more time to compress while a small number of land cover types including sea took less time to archive. Observed decompression times were much lower than the compression time among every program, archiving types and for each algorithm as shown in Figure 8 & Figure 9. LZO archive files were easier to decompress while bz2, lzma and xz files that are compressed by Pbzip2 and XZ softwares were harder to extract as seen from these figures. As the compression level increases up to "ultra compression" level which is equal to 9, archive becomes more complex for these algorithms to decode and thus, decompression process took more time. Also the programs which have gone down below 1 second to decompress the archive are 7z, LZOP, Gzip & *compress* as seen from Figure 9, therefore, there were no significant differences in decompression time.

Considering the archive types, .xz and .lzma extensions not much have different effect on both compression and decompression times as the program XZ was used for archiving as seen from the Figure 4. For .gz format, output archive format also do not have any impact on the compression ratio and time when we look at both Figure 3 & Figure 4. Both programs Gzip and PigZ do not have ended up with different compressed sizes for gzip format since Deflate algorithm. Also, when 7z is used for lossless compression with its own archive format .7z, it will mostly result in its top performance.

According to the Figure 3, the most successful compression algorithm and program which has the highest compression ratio among the others is LZMA compressed by 7z. LZMA and LZMA2 algorithm have the highest compression ratio in general however, the are almost equal to each other. LZMA satisfied a little bit more space by compressing slightly more data than LZMA2 while LZMA2 was faster at making the archive as seen from Figure 7.

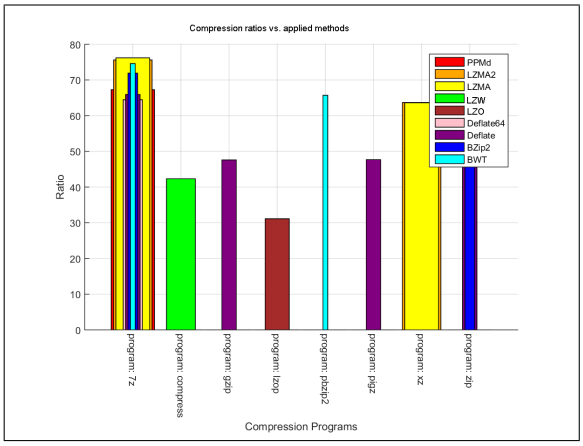

Figure 3: Graph of average compression ratio of each applied method.

5. CONCLUSION

Compression algorithms that are compared in this research mainly seek to reduce redundant information in the datastream. For instance, in order to reduce size of the data that is encoded with 16 bit integer, lossless compression algorithms encode the data

Program	Algorithms	Archive type	Comp. level	Dictionary size	File block size	#thread
Pigz (v2.3.3)	Deflate	.gz	1-3-5-7-9	32 KB		1-2-4-8
Pbzip2 (v1.05)	BWT	.bz2	3-5-9		100-500-900 KB	1-2-4-8
XZ (5.1.2 @alpha)	LZMA & LZMA2	.xz, .lzma	1-3-5-7-9	64 KB - 64 - 1024 MB		1-2-4-8
LZOP (v1.03)	LZO	.lzo	1-3-5-7-9			-
7Z (v9.20)	LZMA LZMA2 BWT Deflate Deflate64 PPMd	.7z, .xz, .gz, .bz2	3-5-9	64 KB 8 MB 64 MB 1024 MB 1536 MB	100-500-900 KB	1-2-4-8
GZIP (v1.5)	Deflate	.gz	1-3-5-7-9	32 KB		-
ZIP (v3.0)	Deflate, BWT	.zip	1-3-5-7-9	32 KB		-
COMPRESS	LZW	.Z	no-level	-		-

Table 2: Compression parameters for programs used.

Program	7Z	Compress	Gzip	LZOP	Pbzip2	Pigz	XZ	Zip	7z	7z
Algorithm	LZMA	LZW	Deflate	LZO	BWT	Deflate	LZMA2	Deflate	PPMd	Deflate64
E_comp	49.5224	57.070	43.3717	110.441	53.7075	179.525	7.469	47.641	27.2852	21.1975
Achieved Ratio	55.7294	43.570	48.0316	24.0662	62.5972	48.0401	38.555	48.0316	64.09	50.2950

Table 3: The best efficiency value achieved by each program and corresponding algorithm.

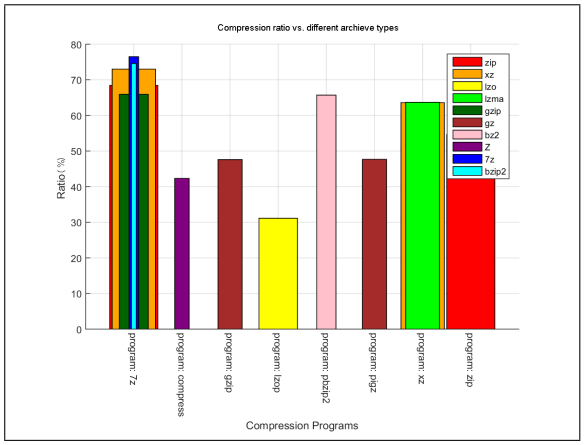

Figure 4: Average compression ratio vs. different archive types.

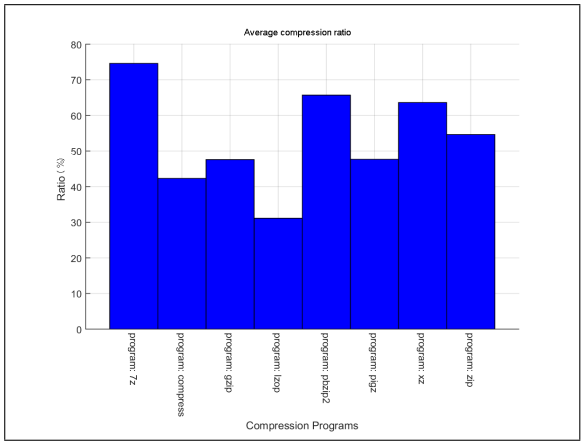

Figure 5: Average compression ratio among programs.

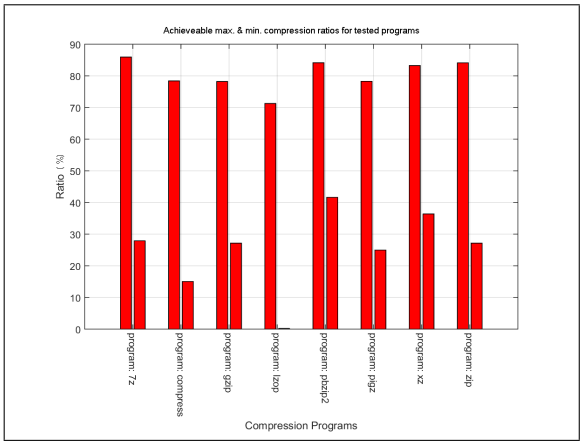

Figure 6: Graph of maximum and minimum of the achieveable compression ratio.

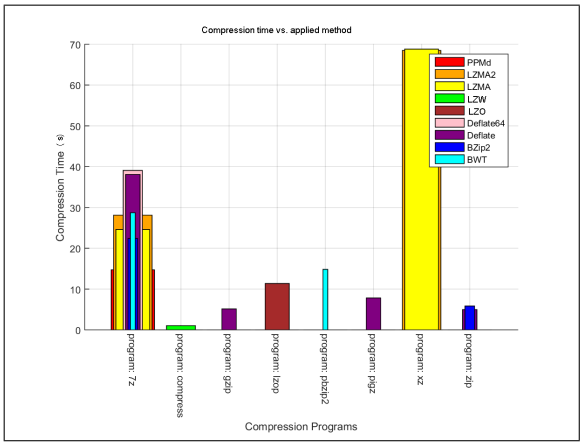

Figure 7: Average compression time vs. applied method.

reducing number of bit integer as much fewer as possible. The LZMA, which is based on LZ77 using a delta filter to sort data for better compression among others, is also an open sourced compression algorithm and it was proved to be effectively used to achieve higher lossless compression ratios. LZO algorithm has proven to be more quick to compress the same data however, it could not achieve higher compression ratio. Another algorithm PPMd, basically using a few characters at the end of the input stream to predict next streams very first characters and then encodes the datastream according to it achieved also not different

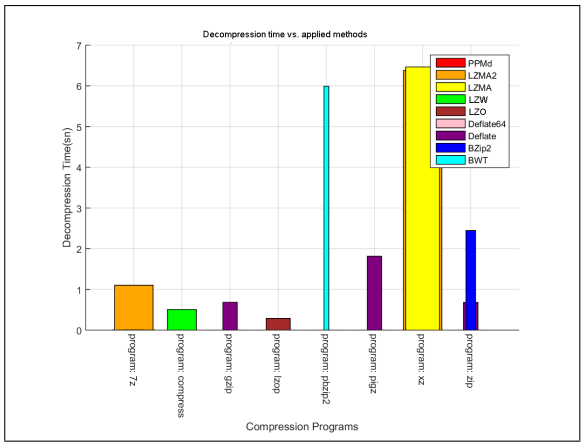

Figure 8: Average decompression time among applied methods.

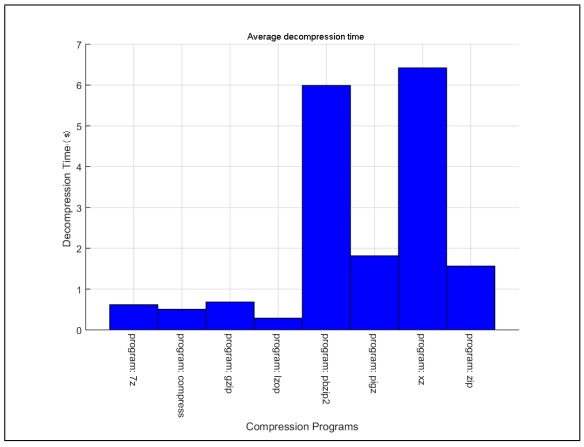

Figure 9: Average decompression time.

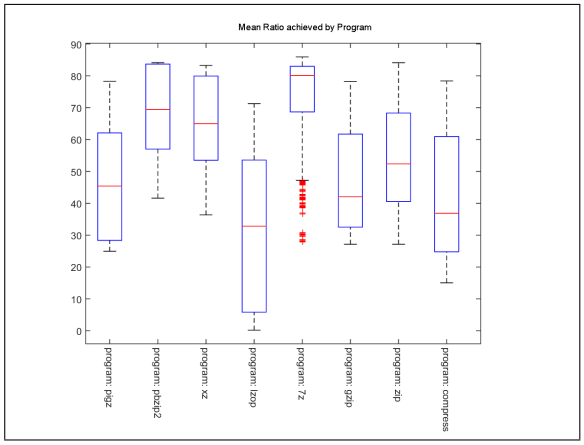

Figure 10: Box and whiskers plots showing statistics and achieved mean ratio per program.

compression ratio however as the PPMd's model prediction order increase the computation time will also increase, we have seen from the results that order 2 model was way more faster than order 8 & 16 model.

To validate the compression procedure was always lossless, after the extraction of the compressed archive, md5 hashes were also generated and compared to the input datasets md5, consequently they appeared as exactly the same. Thus, lossless compression algorithms in this research reduce the size of data which will be stored in a warehouse by no loss of information.

Since that each data compression algorithm have different compression rate and efficiencies, therefore, there have been examined several compression techniques on several images in order to evaluate which compression technique is the best and the most efficient one to achieve high performance for satellite image archiving. In this study, satellite images have been selected from different regions that have different land cover types such as forestry, residential, agricultural areas and marine etc. Images selected from different regions have been produced into two types as orthorectified and primary data and these produced satellite images have been compressed by using different lossless data compression algorithms and compared according to their effectiveness in order to reveal the best lossless data compression algorithm. The results of the compression algorithm implementation will lead us to perform much better satellite imagery archiving strategy and architecture in terms of both storage and file transfer time.

6. FURTHER RESEARCH

According to the results obtained, next step of this research will include the compression of TIFF products of both Pléiades 1A & 1B satellite data which has 0.5 m. GSD, European Space Agency's (ESA) Copernicus Project satellites, called Sentinel, which also include both radar imagery and optical satellite data and products in TIF format and METEOSAT 3rd & 4th generation meteorology satellite data that acquires atmospheric data from GEO. Furthermore, local or land cover based compression approach can also be analyzed in order to define an algorithm for much better compression ratio and less time. Also, algorithm and program variety will be implemented on sample GeoTIFF data. Downlinked imagery will also be produced via different file formats generated and processed by different remote sensing packet programs as .img, .rrd, .pix etc. to decide which compression algorithms are more suitable to store these processed data. For further implementation of the algorithms defined with different approaches, various compression algorithms could be developed in near future.

ACKNOWLEDGEMENTS

SPOT6 & SPOT7 extracted scenes were kindly provided by Istanbul Technical University Center for Satellite Communication and Remote Sensing (ITU-CSCRS), the first satellite ground receiving station of Turkey, established in 1998.

REFERENCES

Burrows, M. and Wheeler, D. J., 1994. A block-sorting lossless data compression algorithm. Technical report, Digital Equipment Corporation.

Cleary, J. and Witten, I., 1984. Data compression using adaptive coding and partial string matching. IEEE Transactions on Communications 32(4), pp. 396–402.

Gilchrist, J., 2004. Parallel data compression with bzip2. In: Proceedings of the 16th IASTED international conference on parallel and distributed computing and systems, Vol. 16, pp. 559–564.

Howard, P. G., 1993. The Design and Analysis of Efficient Lossless Data Compression Systems. PhD thesis, Brown University, Providence, RI, USA. Available as Technical Report CS-93-28.

Kane, J. and Yang, Q., 2012. Compression speed enhancements to lzo for multi-core systems. In: Computer Architecture and High Performance Computing (SBAC-PAD), 2012 IEEE 24th International Symposium on, pp. 108–115.

Leprince, S., Barbot, S., Ayoub, F. and Avouac, J. P., 2007. Automatic and precise orthorectification, coregistration, and subpixel correlation of satellite images, application to ground deformation measurements. IEEE Transactions on Geoscience and Remote Sensing 45(6), pp. 1529–1558.

Mahoney, M. V., 2005. Adaptive weighing of context models for lossless data compression.

Oberhumer, M. F., 1996. oberhumer.com: LZO real-time data compression library. http://www.oberhumer.com/opensource/lzo/.

Pavlov, I., 1999. LZMA SDK (Software Development Kit). http://www.7-zip.org/sdk.html/.

Ritter, N. and Ruth, M., 1995. Geotiff format specification, revision 1.0. Jet Propulsion Laboratory, Cartographic Application Group.

Salomon, D., Bryant, D. and Motta, G., 2010. Handbook of Data Compression. Springer London.

Sayood, K., 2002. Lossless Compression Handbook. Communications, Networking and Multimedia, Elsevier Science.

Shannon, C. E., 1948. A mathematical theory of communication. The Bell System Technical Journal 27(3), pp. 379–423.

Welch, T. A., 1984. A technique for high-performance data compression. Computer 17(6), pp. 8–19.

Ziv, J. and Lempel, A., 1977. A universal algorithm for sequential data compression. IEEE Transactions on Information Theory 23(3), pp. 337–343.