The enhancement of the operating speed of the algorithm of adaptive compression of binary bitmap images

Borusyak A.V.

Research Institute of Applied Mathematics and Cybernetics Lobachevsky Nizhni Novgorod State University National Research University

Abstract. The paper deals with the problem of the enhancement of the operating speed of the algorithm of adaptive compression of binary bitmap images (BBI) on the basis of entropy coding using context simulation. The influence of the size of the maximal order context on the compression rate is considered. The enhancement of the speed of the algorithm depending on the number of the threads used is considered.

Keywords: compression, optimization, binary images, context-based modeling

Citation: Borusyak A.V. The enhancement of the operating speed of the algorithm of adaptive compression of binary bitmap images. Proceedings of Information Technology and Nanotechnology (ITNT-2015), CEUR Workshop Proceedings, 2015; 1490: 262-267. DOI: 10.18287/1613-0073-2015-1490-262-267

1. Introduction

The development of specialized algorithms, targeted for a specific data type, remains relevant in many areas. These algorithms use the knowledge of the internal data structure to achieve a greater degree of compression. One of the important areas is the compression of monochrome black-and-white binary images (MBI). Among the most effective methods of compression of data with a certain structure and specificity there are contextual data compression techniques based on the PPM technique (Prediction by Partial Matching).

The PPM algorithm involves entropy coding using individual context models (CM) for each context, encountered in the stream of encoded data. Each CM includes the counters for all the characters, encountered after the corresponding context. At the same time, the implicit weighting of estimates is used for the application of the context models of several orders. The PPM model itself only predicts the value of a character, while the encoding itself is performed by an encoder using entropy coding algorithms, such as Huffman's algorithm and arithmetic encoding.

2. PCTB Algorithm

The MBI compression algorithm, named PCTB, based on the PPM context simulation technique [1], was developed and implemented. The distinctive feature of this algorithm, as compared to the standard PPM algorithm, is the use of two-dimensional binary context of a specific shape and size, instead of standard one-dimensional context. The use of the property of the two-dimensionality of data in the process of image compression makes it possible to substantially improve the compression ratio, by taking into account the relationships between the neighboring pixels both in horizontal and vertical directions.

The algorithm uses a specific context shape and models of 31th, 11th, 7th, 4th and zeroth order. The B-tree storage structure is used for the efficient storage of context models in the memory.

The following techniques and methods are used in the algorithm: the proprietary algorithm of the estimation of the probability of escape, the method of exclusion of the last encountered symbol, the method of scaling the last encountered symbol using the chosen scaling factor, the modified method of information inheritance.

An arithmetic encoder is used as the entropy encoder. The decoding process is symmetric to the coding process.

The implementation of the algorithm has shown high efficiency in terms of the compression ratio (R_c), but the time expenditures were too high. The time, required for encoding, was reduced by 1.5-2 times by optimization of the computation [3]. However, even after the optimization, the process of encoding remains time-consuming. Moreover, for large images with low redundancy the context model tree grows too quickly, resulting in a high level of RAM consumption and frequent calls for the context model tree purge procedure. The context model tree purge procedure is required for the limitation of RAM consumption. At the same time, frequent calls for this procedure slow down the operating speed of the algorithm and reduce the R_c.

This paper deals with the problem of the reduction of the algorithm run-time and RAM consumption. To further enhance the operating speed of the algorithm, the methods of the parallelization and reduction of the size of maximal order context (MOC) are proposed.

3. Optimization of the time complexity of the algorithm

One way to enhance the speed of the algorithm is the optimization of existing computations. As part of the work [2], various approaches to the enhancement of the operating speed of the algorithm by means of the optimization of computations are proposed. The brief description of the proposed techniques is given below.

In order to optimize the computation function of a new MOC, the sequence order of the context pixels was changed from fixed, predetermined by a programmer, to serial row-major, from the leftmost pixel of the context, which is in line with the pixel to be encoded, to the rightmost, located as far as possible in vertical direction from the pixel to be encoded. This conversion made it possible to reduce the number of access operations to the image when calculating a new MOC from the value of the entire context to the value of the context in height. In the transition to a new line of the image, the fast computation was realized only for those context pixels that did not go beyond the boundaries of the image. A method of calculating lower order contexts, using the MOC calculated with the help of the predetermined mask, is proposed. A mask is a fixed list of serial numbers of MOC pixels, specified by a programmer.

In order to limit RAM consumption, the upper limit for the number of context models, stored in a context model tree, is introduced. When the predefined limit is exceeded, the tree of maximal order context models is purged completely.

A link to a parent context model was added to each context model, in order to accelerate the operation of identification of the parent context model, used in information inheritance and in the escape to the lower order contexts.

Initially, the algorithm used the AVL-tree structure to store the context models. Various options of the storage of the model tree were considered. As a result of the experimental testing, the B-tree turned out to be more efficient with respect to the operating speed, and it was taken as the basic structure for the storage of the context model tree.

The transition of the program to the Q_t 4.8.3 programming environment made it possible to improve efficiency and make the program structure more flexible.

The procedure of the updating of the percentage indicator of the encoding/decoding process was also optimized to improve efficiency.

The above approaches made it possible to reduce significantly, by 1.5-2 times, the time, required for encoding/decoding. The RAM consumption was also limited. Despite the fact that the acceleration proved to be essential, the algorithm still was significantly slower than its counterparts. Due to that, the work on the improvement of the algorithm performance was continued.

4. The dependence of the main compression parameters on the value of the context

One way to enhance the operating speed of programs is to identify the "hot spots of a program" or, in other words, the parts of the program code, consuming the most amount of computer resources. As a result of the research conducted by using a profiler, it was found, that most of the CPU time and RAM are consumed by the functions of calculating a new MOC and searching for a new CM in the CM tree. As noted above, these functions have already been optimized with respect to computation. However, it was noted, that MOC size has a strong influence on the operation of the said functions. Consequently, MOC size affects not only the compression ratio, but also the operating speed of the algorithm and the amount of memory consumed. It has been suggested, that in case of decrease in the MOC size to a certain level, R_c will be reduced slightly, but the compression rate will increase significantly. In view of this, the experiments were conducted to identify the dependence of R_c, compression time and RAM consumption on the MOC size. The results of this experiment are presented in Table 1. According to the experiments conducted, in case of the reduction of the size of the context to 15, the compression ratio is reduced slightly, while a significant decrease in the time required for compression and reduction of the amount of the RAM consumed are noted. The reduction of MOC size to less than 15 leads to a significant reduction in the compression ratio complete with less significant reduction in the RAM consumed and the time required for compression. In Tables 1-3, the rows correspond to the number of the compressed file, and the columns correspond to the size of the maximal order context. At the same time, the reduction of MOC size to below 20 almost does not affect the level of RAM consumption.

Table 1. The dependence of the compression time (sec) on the context size

File\context	31	25	20	15	10	5	1
1	0.98	0.94	0.92	0.77	0.70	0.74	0.66
2	5.47	4.31	3.50	2.67	2.11	2.05	1.56
3	4.30	4.11	4.08	3.56	3.16	3.58	2.87
4	712.63	544.81	358.42	214.62	139.06	127.47	97.89
5	180.71	175.30	173.63	155.85	137.90	188.71	123.95

Table 2. The dependence of the compression ratio on the context size

File\context	31	25	20	15	10	5	1
1	113.46	111.58	107.17	99.73	94.28	78.17	36.94
2	5.26	5.15	5.04	4.83	4.57	4.11	3.78
3	62.34	61.20	59.98	58.07	55.32	49.03	17.92
4	2.69	2.66	2.66	2.54	2.28	1.72	0.99
5	55.57	53.29	51.90	50.34	45.93	37.23	15.30

Table 3. The dependence of RAM consumption (Mb) on the context size

File\context	31	25	20	15	10	5	1
1	16	15.4	14.3	14.2	15	14.2	14.3
2	38.6	26.5	19	15.2	15.2	15.2	14.7
3	18.2	17.3	17.1	17.3	17.3	17.2	17
4	621	259.7	103.5	103	104	103	103
5	189	188.7	188.4	188.4	189	188	188.7

5. Parallelization of the encoding algorithm

Modern computers are often equipped with processors, containing multiple cores, which enables parallel computing. This feature is currently often used to speed up the algorithms. In order to further enhance the coding/decoding speed, the possibility of encoding and decoding BBI through the parallelization of processing into multiple threads was implemented in the compression algorithm. The parallelization algorithm for the compression of indexed images [3] is used as the basis of the parallelization algorithm. The ability to split the image into *n* parts is implemented. Assume the vertical dimension (height) of the image for *H*, the horizontal dimension (width) for *W*. The image is divided as follows: if the number of parts is equal to 4, the image is divided along the lines, connecting the midpoints of the opposite sides. If $n \neq 4$, the vertical sides of the image are divided into *n-1* equal parts with the height equal to H/n, and one part with the height (*H-H/n*(n-1)*). After splitting the image into several parts, each part of the image is compressed as an individual image in a separate thread. This approach makes it possible to use the capabilities of modern computers, evenly distributing the load on processor cores, while slightly reducing the compression efficiency. It is assumed that with this approach, the increase in the number of threads by *n* times will increase the compression rate to *n* times, provided that the number *n* is less than or equal to the *nc* number of the processor cores. At the same time, the rate will increase in proportion to the number of threads. If the *n* number is greater than the *nc*, the increase in the compression rate, approximately equal to the use of the *nc* threads, is expected. During the computation on a dual core PC1 (CPU – Core 2 Duo E7400 2.8 GHz) and a quad-PC2 (CPU – Core i5-3230M 2.6 GHz) the following results were obtained:

Table 4. Comparison of encoding time for the file PGS2 (32294x25003, colour depth – 1 bit, number of colours – 1) depending on the number of threads

Number of threads	Time PC1 (sec.)	Time PC2 (sec.)	R_c
1	90.496	69.405	71.99
2	53.228	50.701	71.90
4	51.012	43.992	71.76
8	50.451	39.094	71.53
16	50.029	37.565	71.14
32	49.764	36.769	70.52
64	51.106	37.564	69.54

Table 5. Comparison of encoding time for the file floor_plan (5000x3336, colour depth – 1 bit, number of colours – 1) depending on the number of threads

Number of threads	Time PC1 (sec.)	Time PC2 (sec.)	R_c
1	1.86	1.389	62.35
2	0.984	0.827	61.98
4	1.062	0.78	61.25
8	0.99	0.796	60.46
16	0.983	0.78	59.11
32	0.996	0.78	57.18
64	0.949	0.827	53.99

Where, in the first column of Tables 4 and 5 the number of threads used for image encoding is indicated. In the second and the third columns the corresponding encoding time in seconds is indicated for PC1 and PC2. The last column shows the Rc for this file, depending on the number of the threads used. This ratio depends on the number of threads, but doesn't depend on the computer used for encoding, so the two computers possess the same factors. It is seen from the experiments, that, as expected, the optimal number of threads is equal to the number of cores in the system. Thus, the compression ratio reduces linearly, but very slightly, and at 64 threads, the losses less than 3% are observed, while at *n*-16 threads, the losses are less than 1%.

6. Conclusion

The problem of the enhancement of the operating speed of the algorithm of adaptive compression of binary bitmap images was considered. The experiments were conducted for the enhancement of the operating speed of the proposed algorithm by parallelizing the algorithm into several threads and reducing MOC size. Both approaches proved to be effective. The MOC size, the reduction to which greatly reduces the file encoding time and significantly reduces RAM consumption, while only slightly reducing the Rc, was identified. The experiments were conducted for parallelizing the algorithm into several threads. Using these two approaches together, it is possible to enhance the rate of compression to an average of 8 times on a computer with a 4-core processor, to reduce RAM consumption by 2 times, with the average Rc losses being less than 10%.

Acknowledgments

This work was supported by RFBR (Projects No. 13-07-00521-A and No. 13-07-12211 OFI_M.)

References

1. **Borusyak AV, Vasin YuG, Zherzdev SV.** Compression of Binary Graphics Using Context Simulation. Pattern Recognition and Image Analysis, 2013; 23(2): 207-210.
2. **Borusyak AV, Vasin YuG, Zherzdev SV.** Optimizing the computational complexity of the algorithm for adaptive compression of binary raster images. Proceedings of The 11-th International Conference “Pattern Recognition and Image Analysis: new information technologies”, 2013; 1: 170-172.
3. **Borusyak AV, Vasin YuG.** Compression of indexed graphic images using context modeling. Vestnik of the Lobachevsky State University of Nizhni Novgorod, 2014; 4(1): 486-492.
4. **Vatolin D, Ratushnyak A, Smirnov M, Yurkin V.** Methods of data compression. Construction of archivers, image and video compression. Moscow: Dialog – MEPHI, 2003.
5. **Vasin YuG. Zherzdev SV.** Information Techniques for Hierarchical Image Coding. Pattern Recognition and Image Analysis, 2003, 13(3): 539-548.
6. Source <http://www.imagecompression.info/test_images>.