PERGAMON

Computers & Graphics 24 (2000) 91–98

COMPUTERS & GRAPHICS

www.elsevier.com/locate/cag

Computer graphics in Russia

Lossless compression of large binary images in digital spatial libraries☆

Eugene Ageenko*, Pasi Fränti

Department of Computer Science, University of Joensuu, Box 111, FIN-80101 Joensuu, Finland

Abstract

A method for lossless compression of large binary images is proposed for applications where spatial access to the image is needed. The method utilizes the advantages of (1) variable-size context modeling in a form of context trees, and (2) forward-adaptive statistical compression. New strategies for constructing the context tree are introduced, including a fast two-stage bottom-up approach. The proposed technique achieves higher compression rates and allows dense tiling of images down to 50×50 pixels without sacrificing the compression performance. It enables partial decompression of large images far more efficiently than if the standard JBIG was applied. © 2000 Elsevier Science Ltd. All rights reserved.

Keywords: Image compression; Spatial access; Context tree; Context modeling; Binary images; Digital spatial libraries

1. Introduction

We consider digital maps generated within several international projects headed by the *National Land Survey of Finland* (NLS) [1]. These include: *MapBSR* project, covering the entire Baltic Sea region, *Barents* geographic information technology (GIT), a project between Finland, Sweden, Norway and Russia, and *GIS-Sever project*, spanning the border between Finland and Russia. The objective of the projects is to produce uniform geographic information that can be used in planning and decision making of communication, infrastructure, technical, economical and cultural cooperation, tourism and security interests.

Raster images are generated from map database for digital publishing on CD-ROM, or in the web. The images consist of several binary layers, which together form the computer-generated color image, as shown in Fig. 1. The number of layers is limited but the size of a single image is typically very large. As a test case, we use images from the Topographic Map Series 1 : 20,000 of Finland.

The main problem in digital spatial libraries is the huge storage size of the images. For example, a single map sheet of 5000×5000 pixels representing a single map sheet of $10 \times 10\ \text{km}^2$ requires about 12 Mb. The necessity of compression for saving storage space is therefore obvious. Another highly desired feature of digital spatial libraries is spatial access to the image. Spatial access enables the user to operate directly on the compressed data without retrieving the entire image. It makes possible an efficient retrieval and decompression of the desired parts of the image with high precision. Spatial access would therefore be a significant feature in imaging databases [2], as in engineering document management systems (EDM) [3], and geographic information systems (GIS), where large format images are used [4].

Spatial access can be supported by *tiling*, i.e. dividing the image into fixed size rectangular blocks [4]. The blocks (denoted here as *clusters*) are compressed independently by any compression method. JBIG is the latest standard for binary image compression and therefore applicable for this purpose [5]. Straightforward combination of tiling and the backward adaptive JBIG, however, suffers from high learning cost because of coding much smaller size of data in comparison to the whole

☆ The short version of this paper will appear in Proc. IS&T/SPIE Symposium on Electronic Imaging 2000, San Jose, CA, Jan. 2000.

* Corresponding author. Tel.: + 385-13-251-3103; fax: + 358-13-251-3290.

E-mail addresses: ageenko@cs.joensuu.fi (E. Ageenko), franti@cs.joensuu.fi (P. Fränti)

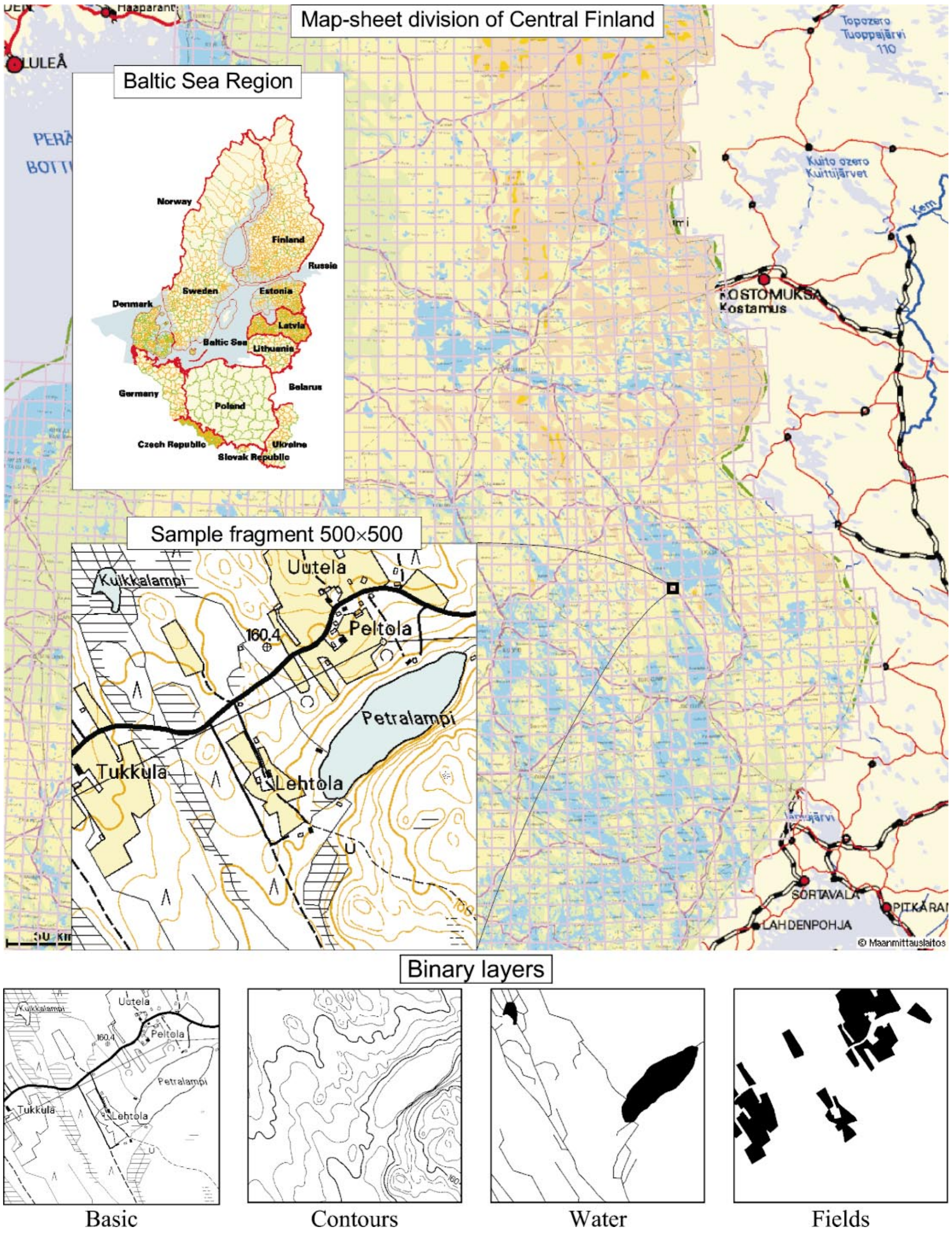

Fig. 1. Illustration of NLS Digital Spatial Library (with the permission of National Land Survey of Finland).

image. Using JBIG, it is possible to partition the image into clusters of 350×350 pixels without sacrificing the compression performance. A better solution is to use forward-adaptive statistical modeling as proposed in [6]. Using this technique, it is possible to achieve cluster size of 100×100 pixels.

We propose an improved compression method, in which variable-size context modeling and forward-adaptive statistical modeling are combined. This scheme improves the compression performance of JBIG by over 20%, which is sufficient to partition the image into clusters of as small as 50×50 pixels. We consider various tree construction approaches, including a new fast bottom-up approach. We consider semi-adaptive approach where the tree is optimized on-line for the image to be compressed, and static approach where the tree is estimated off-line using a training image. We give comparative results for the proposed and existing techniques by compressing a set of topographic images.

The rest of the paper is organized as follows. The components of the existing methodology are briefly summarized in Section 2. The variable size context modeling is described in Section 3. The principles of the context tree are given in the Sections 3.1 and 3.2. Existing top-down approaches for constructing the tree are discussed in Section 3.3, and new bottom-up approach is introduced in Section 3.4. The parameters of the context tree model and forward-adaptive statistical modeling are considered in Sections 3.5 and 3.6. The decompression phase is discussed in Section 3.7. Experiments of the proposed compression system are reported in Section 4, and conclusions are drawn in Section 5.

2. Compression system for digital spatial libraries

2.1. JBIG-based compression

In sequential JBIG, the image is coded pixelwise in raster scan order using *backward adaptive context-based statistical modeling* and *arithmetic coding* [6,7]. The combination of already coded neighboring pixels defines the context (we will assume a three-line 10-pixel default template). JBIG uses an arithmetic coder, namely the QM-coder. The probability estimation in the QM-coder is derived from arithmetic coder renormalization [8]. Instead of keeping the pixel counts, the probability estimation is implemented as a state machine. It consists of 226 states organized in a Markov-chain. The adaptation process starts from the *zero state*, having approximately equal probability distribution.

The emerging standard JBIG2 [9] improves the compression of text images using pattern matching technique for extracting symbols from the image. The compressed file consists of bitmaps of the library symbols coded by a JBIG-style compressor, location of the extracted marks as offsets, and a pixelwise coding of the matched symbols using two-layer context template. The standard includes also an optional lossy mode for filtering the matched symbols. These two enhancements, however, are of limited usage in the case of cartographic images, as they do not include large number of text elements. Moreover, the images are usually computer generated and therefore not affected by digitization noise.

2.2. Image tiling

To provide spatial access, the image is divided into clusters of $C \times C$ pixels and each cluster is compressed independently from each other. Cluster index table is constructed from the pointers indicating the location of the cluster data in the compressed file and stored in the file header. To restore any part of the image, only the clusters consisting of the desired pixels need to be decompressed.

The cluster size is a trade-off between compression efficiency and decoding delay. If very small cluster size is used the desired part of the image can be reconstructed more accurately and faster. The compression, however, would be less efficient because of an increased learning cost and a less accurate probability model because of the cluster boundaries. The index table itself requires space and the overhead is relative to the number of clusters.

2.3. Forward-adaptive statistical modeling

In forward-adaptive modeling (FAM), the image is compressed by a two-stage procedure. In the first stage, the image is analyzed and the frequencies of the white and black pixels (f_W^C and f_B^C) are calculated for each context C. The resulting probabilities are mapped to the respective states in the state automaton of the QM-coder using a look-up table. In the mapping, we use only the 28 "*fast-attacks*" states of the automaton because they represent the full probability range with sufficient accuracy and allow faster adaptation. The derived model table is stored in the header of the compressed file.

In the second stage, the clusters are compressed separately using QM-coder. The compression is essentially the same as in sequential JBIG. The only differences are that the QM-coder is reinitialized and the model is restored each time the compression of a new cluster starts.

The forward-adaptive modeling compensates the coding inefficiency from the tiling by reducing the learning cost. This is achieved at the cost of increased overhead. The overhead is two bits per context, which is usually negligible in the case of very large images. For details of the forward-adaptive modeling, see [6].

3. Variable-size context modeling

In forward-adaptive statistical modeling, the context size is a trade-off between accuracy and overhead of the model. A larger context template results in a more accurate probability model. The overhead, however, grows exponentially with the size of the context template. It is therefore better to use variable-size context model, in which the context selection is made using a context tree instead of a fixed size template [10,11].

3.1. Context tree

Variable-size context model can be implemented using a *context tree* (CT) where the number of context pixels

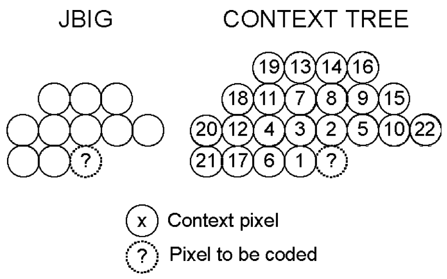

Fig. 2. The default three-line context template of the sequential JBIG (left), and the 22-pixel *ordered neighborhood* used for the context tree (right). The first 10 pixels in the neighborhood constitute the JBIG template.

depends on the combination of the neighboring pixel values. The context selection is made using a context tree instead of a fixed size template. Each node in the tree represents a single context, and the two children of a context correspond to the parent context augmented by one more pixel. The position of this pixel can be fixed in a predefined order estimated for a typical topographic image, see Fig. 2, or is optimized for every context within a limited search area relative to the compressed pixel position (we refer to the last case as *free tree* [11]). Only the leaves of the tree are used in the compression. The context tree example is shown in Fig. 3.

3.2. Construction of the tree

During the tree construction, the image is processed and the statistics f_W^C and f_B^C are calculated for every context in the full tree, including the internal nodes. The tree is then pruned by comparing the children and parents nodes at each level. If compression gain from using the children nodes instead of their parent node is not achieved, the children are removed from the tree and their parent will become a leaf node. The compression gain is calculated as

$$Gain(C, C_W, C_B) = l(C) - l(C_W) - l(C_B) - SplitCost. \quad (1)$$

where C is the parent context and C_W and C_B are the two children nodes. The code length l denotes the total number of output bits from the pixels coded using the context. The cost of storing the tree and the statistical model is integrated in the *SplitCost*. With forward-adaptive compression in mind, we can estimate the code length as

$$l(C) = f_W^C \log_2\left(\frac{f_W^C}{f_W^C + f_B^C}\right) + f_B^C \log_2\left(\frac{f_B^C}{f_W^C + f_B^C}\right). \quad (2)$$

According to the direction of the pruning, the tree construction is classified either as *top-down* or *bottom-up*.

3.3. Top-down construction

In top-down approach, the tree is constructed stepwise by expanding the tree one level at a time starting from a predefined minimum level k_{MIN}. The process starts by constructing the models for all contexts at the level k_{MIN}. The next level contexts are tentatively constructed, compared to their parent contexts, and pruned. The process continues until a predefined maximum level k_{MAX} is reached, or when no new nodes were created during the process of a single level.

A drawback of this approach is that a context, which delivers negative gain at some step of the iteration, will not be expanded further, even if the expansion could deliver positive gain later. The on-line construction of the tree makes also the compression an order of magnitude slower than JBIG.

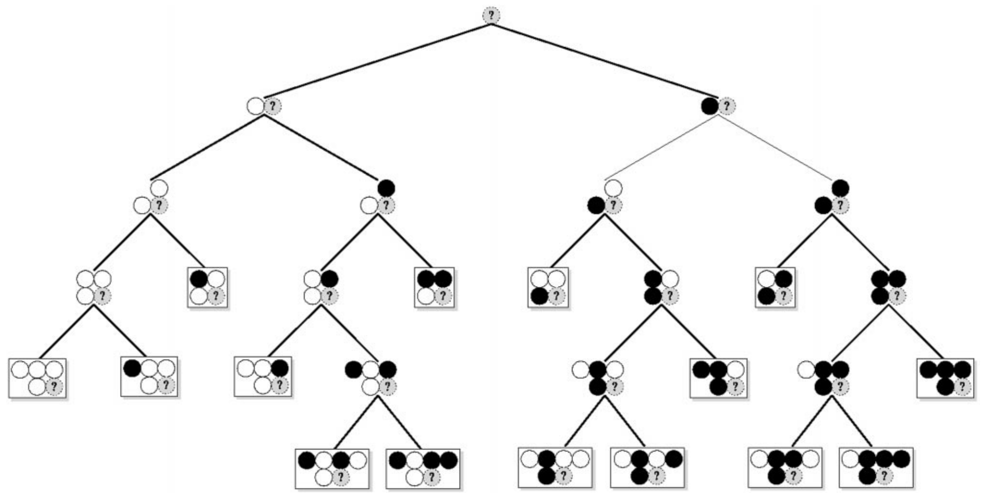

Fig. 3. Example of a context tree.

Another approach using top-down tree construction is the free tree [11], in which the position of the context pixel is adaptively determined for each context. When a new level is constructed, all possible positions for the next context pixel are analyzed within a predefined search area. The position resulting in maximal compression gain is chosen for each context separately. The drawback of this approach is that the position of the new context pixel must also be stored in the compressed file. The computational complexity of the free tree algorithm is also much greater and it grows with a factor of the search area size.

3.4. Bottom-up construction

In bottom-up approach, the tree is analyzed from the leaves to the root. A full tree of k_{MAX} levels is first constructed by calculating statistics for all contexts in the tree. The tree is then recursively pruned up to the level k_{MIN}, using the same criterion as in the top-down approach. The code lengths from children contexts $l(C_W)$ and $l(C_B)$ are derived from the previous level of recursion. Children sub-trees of the nodes that do not deliver positive compression gain are removed from the tree. The sketch of the implementation is shown in Fig. 4 and the algorithm is illustrated in Fig. 5.

The advantage of the bottom-up approach is that only one pass over the image is required. Unfortunately, high k_{MAX}, values will result in huge memory consumption. For this reason, we propose a two-stage bottom-up pruning procedure. In the first stage, the tree is constructed to the level k_{START} and then recursively pruned up to the level k_{MIN}. In the second stage, the remaining leaf nodes at the level k_{START} are expanded to the level k_{MAX} and then pruned up to the level k_{START}. In his way, the memory consumption depends mainly on the choice of the k_{START} because only a small proportion of the nodes at that level remains after the first pruning stage. k_{START} is selected to have as great value as memory resources permit to cause minimal influence on the structure of the context tree.

```
PruneTree (CONTEXTTREE CT, int level)
    if (level = k_MAX) then  // we reached the end of tree
        return (CodeLength (CT));
    else  // process the subtrees recursively
        CLw = PruneTree (CT→WhiteChild, level+1);
        CLb = PruneTree (CT→BlackChild, level+1);
        if (level ≤ k_MIN) then  // out of pruning range
            return (0);
        else  // check the node for pruning
            CL = CodeLength (CT);
            Gain = CL – CLw – CLb – SplitCost ;
            if (Gain > 0) then // split node
                return (CLw + CLb + SplitCost);
            else  // prune node
                RemoveTree (CT→WhiteChild);
                RemoveTree (CT→BlackChild);
                return (CL);
```

Fig. 4. Recursive bottom-up tree pruning algorithm.

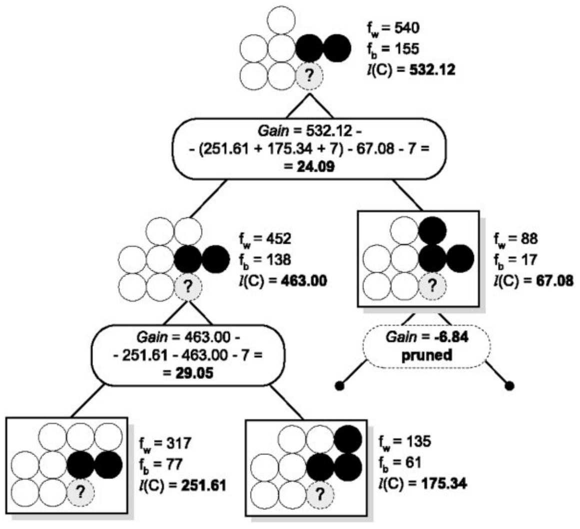

Fig. 5. Illustration of bottom-up tree pruning.

3.5. Semi-adaptive and static variants

There are two alternative approaches for generating the context tree. In *semi-adaptive* approach, the tree is optimized directly for the image to be compressed. An additional pass (or passes) over the image will be required. The cost of storing the tree structure is approximately two bits per context, i.e. one bit per node ("1" for indicating a divided node, and "0" for indicating a leaf node). For a free tree, the position of the next context pixel must also be stored. It can be represented as an index within the search area, and stored with [log(*window_size*)] bits.

Another approach is to use a *static* tree optimized using a training image. This is possible because of the similarity of the trees for images of similar type. The main problem of the static approach is to control the growth of the tree. There is no overhead from storing the tree and therefore we must add a progressively weighted constant to the *SplitCost* in order to prevent the tree from greedy growing. If the forward-adaptive statistical model is applied, two passes will be required anyway, and therefore the semi-adaptive approach is preferred.

3.6. Combination with forward-adaptive modeling

The context tree, as described here, can be integrated with the forward-adaptive statistical modeling as follows.

All compression procedures remain the same, only the context selection is derived by traversing the context tree. In the construction of the tree, we use (2) for estimating the code length of a context, and add the cost of storing the statistical model to the *SplitCost*. Incorporating the model cost (5 bits per context) and the tree store cost (2 bits per context) in the *SplitCost* makes it possible to achieve optimal tradeoff between compression improvement and overhead. The combination of context tree and forward-adaptive statistical modeling is denoted here as CT-FAM.

3.7. Decompression

To decompress the image or its fragment, the compressed file header is first processed, and cluster indices, model table, and context tree are read and re-constructed in memory. The respective clusters are then located in the compressed file and decompressed. The same procedures of the QM-coder are utilized for decoding, only the context selection is derived by the tree, and the coder is reinitialized before processing each cluster.

4. Empirical results

The performance of the proposed method is demonstrated by compressing a set of images from the NLS topographic database (Basic Map series 1 : 20,000) using Pentium-200/MS-Windows 95 system. The images are formed by several binary layers, each has size of 5000×5000 pixels:

- *basic* — topographic image, supplemented with communications networks, buildings, protected sites, benchmarks and administrative boundaries;
- *contours* — elevation lines;
- *water* — lakes, rivers, swamps, water streams;
- *fields*.

In our experiments, we use five images randomly chosen from the database images corresponding to the map sheets *No/No* 431306, 124101, 201401, 263112, and 431204. For each image, we have four binary components referred as $Layer_X$, where *Layer* is the binary layer name, and X is the image number from 0 to 4 in a given order.

We evaluate the seven methods shown in Table 1. Sequential JBIG [6] and the CCITT Groups 3 and 4 [12] are the points of comparison. CT is the combination of context tree and JBIG. In this method, the tree is constructed using bottom-up strategy and the Bayesian sequential estimator is applied for the code length as in [11]. These four methods do not support spatial access whereas the rest of the methods do. T-JBIG is the combination of tiling and sequential JBIG. FAM stands for the forward-adaptive JBIG-based compression [6], CT-FAM for the proposed technique, and CT-FAM^S is its static variant.

First, we evaluate the different approaches for building the context tree, see Table 2. For the free tree approach, the size of the search template is 40. The split cost is composed from the cost of storing the tree and the model (7 bits per context for context tree, and 12 bits for the free tree). Compression ratios are given for CT-FAM method when applied to the image Basic0. The respective compression ratio of JBIG is 8.74.The bottom-up tree construction is faster than the top-down approach but it requires more memory. The memory load grows exponentially with the size of the initial tree and is trade-off between compression performance and memory consumption.

The top-down construction of the tree can be performed with a small memory load (50 K), but it is time consuming and therefore inapplicable for on-line compression. Another problem is that the expansion of some branches may stop too early because of the locality of the splitting criterion. The bottom-up approach does not suffer from this problem.

The free tree does not give significant improvement over the top-down approach with fixed split pixel. The reasons are the high split cost, early termination of the tree expansion, and a limited search template (40 pixels).

Table 1
Compression methods and their properties

Method	Tiling	Statistical model	Context tree	Passes
Group $\frac{3}{4}$	–	n/a	–	1
JBIG	–	Backward-adaptive	–	1
CT	–	Backward-adaptive	Semi-adaptive	2[a]
T-JBIG	+	Backward-adaptive	–	1
FAM	+	Forward-adaptive	–	2
CT-FAM	+	Forward-adaptive	Semi-adaptive	2[a]
CT-FAM^S	+	Trained	Trained	1

[a]One stage is assumed in the bottom-up context tree construction. Add one more pass for two-stage bottom-up method.

Table 2
Properties of the discussed tree-building strategies. Numbers in parenthesis: (k_{MIN}, k_{MAX}), for two-stage bottom-up pruning: (k_{MIN}, k_{START}, k_{MAX})

	Top-down		Free tree	Bottom-up		
	(6,22)	(10,22)	(2,22)	(2,18)	(2,22)	(2,18,22)
Contexts in the tree	1366	2373	2041	5596	8209	6527
Tree file size (bytes)	341	591	1786	1400	2053	1632
Passes over image	16	12	20	1	1	2
Creation time	30 min 20 s	26 min 58 s	1 h 58 min 33 s	3 min 8 s	4 min 56 s	6 min 31 s
Memory load (bytes)	26 K	51 K	1 M	8.5 M	136 M	8.5 M
Compression ratio	10.04	10.40	11.35	11.14	11.65	11.44

Table 3
Bit rates per image type in the test set, and overall compression ratios for the discussed methods

	Methods not supporting tiling				Methods supporting tiling (256 × 256 cluster size)			
Test images	Group 3	Group 4	JBIG	CT	T-JBIG	FAM	CT-FAM	CT-FAM^S
Basic$_{1\text{-}4}$	2,834,589	2,881,614	1,197,983	884,435	1,263,311	1,211,338	903,597	944,107
Contours$_{1\text{-}4}$	1,968,901	1,230,480	643,998	514,353	683,314	632,882	536,788	549,571
Water$_{1\text{-}4}$	1,122,591	548,124	270,703	206,282	280,031	249,697	207,829	210,636
Fields$_{1\text{-}4}$	233,415	64,530	29,127	25,030	35,914	33,412	28,558	33,412
TOTAL (16)	6,159,496	4,724,748	2,141,811	1,630,100	2,262,570	2,127,329	1,676,772	1,737,726
Compression ratio	**8.12**	**10.58**	**23.34**	**30.67**	**22.10**	**23.50**	**29.82**	**28.77**

A delayed pruning technique [10] and a significantly larger search template (about 500 pixels) could be applied for improving the result. However, it would result in significant increase in memory consumption and running time, and is therefore not worth doing here.

Bottom-up pruning requires only one or two passes over the image and gives better compression performance. The one-stage variant with $k_{MAX} = 22$ has the highest compression performance but the two-stage variant requires much less memory (8.5 Mbytes vs. 136 Mbytes). In the first stage, the tree is pruned from level 18 to 2. During this stage, 525,252 nodes are analyzed in total. In case of Basic$_0$, the number of leaf nodes was reduced from 256,548 to 5596. Only 1305 of these belong to the 18th level. In the second stage, these nodes are expanded down to 22th level. In total, 20,880 nodes were analyzed and 2236 new leaf nodes were created. Thus, most of the nodes are analyzed and pruned during the first stage. The two-stage bottom-up pruning is used in the following.

The overall effect of tiling, and the variable-size context modeling is summarized in Fig. 6. The advantage of using a better initial model outweighs the overhead in all cases, except if a very small cluster size is selected. The sequential JBIG could be applied with the tiling using cluster size of about 300 × 300 without sacrificing the compression performance. The corresponding number is 100 × 100 for the FAM, and 50 × 50 for the CT-FAM.

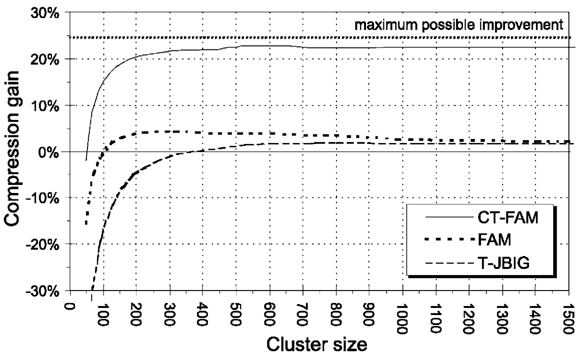

Fig. 6. Compression performance of T-JBIG, FAM, CT-FAM as a function of the cluster size compared to the sequential JBIG.

Moreover, the CT-FAM improves the compression performance of about 20% if the cluster size is 200 × 200 or greater. The maximum potential improvement is estimated as the compression rate improvement that would be achieved if the tiling has not been applied (CT). It amounts to circa 25%.

Comparative compression results for the discussed methods are summarized in Table 3. For this test, the image 0 is used as training image (for the static variant), and the images 1–4 are used for the compression. Tiling the image to clusters of size 256 × 256 is assumed for the methods that support spatial access. The two-stage bottom-up pruning is used for context tree construction. For evaluating the static variant of the CT-FAM method, four context trees were trained off-line (using Image$_0$) for each binary component separately. Experiments show that the proposed method improves JBIG by over 20% for this cluster size. The static variant is also applicable because of similar type images. In our example, the static variant was only 3.5% worse than the semi-adaptive.

Unlike JBIG, the image decompression time depends on the image complexity and the size of the context tree. For the whole image, it varies from 1 s to approximately 2 min, when sequential JBIG requires constantly 65 s. On average for our test set, decompression of the entire image using CT-FAM method is about 1.8 times slower than using JBIG. At the same time, partial images (e.g. 512×512 pixels, used by default on NLS GIS WEB-server) can be retrieved practically instantly.

5. Conclusion

We propose a compression method that uses a combination of variable-size context modeling and forward-adaptive statistical modeling. The use of context tree makes it possible to utilize larger context templates without overwhelming the learning cost. Selective context expansion results in improved prediction and better compression performance. The method alleviates the deterioration of the coding efficiency caused by tiling and achieves higher compression rates because of the improved pixel prediction caused by utilizing the larger contexts.

The method is applied for the compression of binary layers of maps for four different domains. The simulation results show that the proposed method allows dense tiling of large images down to 50×50 pixels vs. 350×350 pixels that of JBIG without sacrificing the compression performance. The improved compression method allows partial decompression of large images far more efficiently than with standard JBIG. For clusters larger than 200×200 pixels, the method improves the compression performance of JBIG by about 20%.

We have also considered a static variant of the method, in which the model is generated using a training image. It results in a faster one-pass compression and enables image tiling down to 100×100 pixels. Static variant can be applied if the images are known to be of similar type. Otherwise, the two-pass method should be used at the cost of higher compression times. The decompression times are similar in both cases.

Acknowledgements

We acknowledge the National Land Survey of Finland for providing the image data. The work of Pasi Fränti was supported by a grant from the Academy of Finland. Eugene Ageenko acknowledges the Moscow State University, Dept. of Applied Mathematics, where during the graduate and post-graduate studies he started this research.

References

[1] National Land Survey of Finland. Opastinsilta 12 C, P.O. Box 84, 00521 Helsinki, Finland. http://www.nls.fi/index_e.html

[2] Pajarola R, Widmayer P. Spatial indexing into compressed raster images: how to answer range queries without decompression. In: Proceedings of the International Workshop on Multimedia DBMS. NY: Blue Mountain Lake, 1996. p. 94–100.

[3] Ageenko EI, Fränti P. Enhanced JBIG-based compression for satisfying objectives of engineering document management system. Optical Engineering 1998;37(5):1530–8.

[4] Samet H. Applications of spatial data structures: computer graphics, image processing and GIS. Reading, MA: Addison-Wesley, 1989.

[5] JBIG: Progressive Bi-level Image Compression. ISO/IEC International Standard 11544, ISO/IEC/JTC/SC29/WG9, 1993.

[6] Ageenko EI, Fränti P. Forward-adaptive method for compressing large binary images. Software Practice & Experience 1999;29(11):943–52.

[7] Pennebaker WB, Mitchell JL. JPEG Still Image Data Compression Standard. New York: Van Nostrand Reinhold, 1993.

[8] Pennebaker WB, Mitchell JL. Probability estimation for the Q-coder. IBM Journal of Research, Development 1998;32(6):737–59.

[9] Howard PG, Kossentini F, Martins B, Forchammer S, Rucklidge WJ. The emerging JBIG2 standard. IEEE Transactions on Circuits and Systems for Video Technology 1998;8(7):838–48.

[10] Fränti P, Ageenko EI. On the use of context tree for binary image compression. IEEE Proceedings of the International Conference on Image Processing (ICIP '99). Japan: Kobe, 1999.

[11] Martins B, Forchhammer S. Bi-level image compression with tree coding. IEEE Transactions on Image Processing 1998;7(4):517–28.

[12] Arps RB, Truong TK. Comparison of international standards for lossless still image compression. Proceedings of the IEEE 1994;82:889–99.