Context tree compression of multi-component map images

Pavel Kopylov and Pasi Fränti

Department of Computer Science, University of Joensuu, FINLAND

Abstract: We consider compression of multi-component map images by context modeling and arithmetic coding. We apply optimized multi-level context tree for modeling the individual binary layers. The context pixels can be located within a search area in the current layer, or in a reference layer that has already been compressed. The binary layers are compressed using an optimized processing sequence that makes maximal utilization of the inter-layer dependencies. The structure of the context tree is static variable depth binary tree, and the context information is stored only in the leaves of the tree. The proposed technique achieves improvement of about 25 % over static 16 pixel context template, and 15 % over similar single-level context tree.

Index terms: binary images, context optimization, context tree, multi-layer images.

1. Introduction

We consider large images from the topographic map series 1:20 000 of *National Land Survey of Finland* (NLS) [1]. The size of each image is 5000×5000 pixels, and represents a 10×10 km^2 area. The map image consists of four binary layers with different semantic content, corresponding to the topographic data, fields, elevation lines and water area. The layers are combined and displayed to the user as color image, as shown in Figure 1.

The binary layers can be compressed using context-based statistical modeling and arithmetic coding as in JBIG [2] and JBIG2 [3][4]. The pixels form geometrical structures with appropriate spatial dependencies that can be localized to a limited neighborhood. Context-based image compression utilizes the dependencies by conditioning the pixel probabilities on the combination of neighboring pixel values. The compression consists of two distinct phases: statistical modeling and arithmetic coding [5].

In the modeling phase, the probability distribution is estimated on the basis of the pixel configuration in the template. For example, the pixel configuration in Figure 2 is defined by the binary number 1110010010_2, which gives the context index 914 out of 1024 possible within this template. Each context is assigned with its own statistical model that is adaptively updated during the compression/decompression process. Arithmetic coding assigns optimal code for the pixels in regards to the given statistical model [6].

In principle, a better probability estimation can be achieved using a larger context template. The use of large template, however, does not always result in compression improvement. The number of contexts grows exponentially with the size of template; adding one more pixel to the template doubles the size of the model. This leads to the *context dilution* problem, in which the statistics are distributed over too many contexts,

Proceedings of the DATA COMPRESSION CONFERENCE (DCC'02)
1068-0314/02 $17.00 © 2002 IEEE

and thus, affects the accuracy of the probability estimates. The size and location of the template pixels can be optimized by exhaustive search as has been done for the case of dithered images in [7], and for multi-level map images in [8].

Context tree provides a more flexible approach for modeling the contexts so that larger number of neighbor pixels can be taken into account without the context dilution problem [9]. The contexts are represented as *binary tree*, and the context is constructed pixel by pixel. The context selection is deterministic and only the leaves of the tree are used. The location of the next neighbor pixels, and the depth of the individual branches of the tree depend on the combination of the already coded neighbor pixel values. Once the tree has been created, it is fully static and can be used in the compression as any other fixed-size template.

In this paper, we extend the idea of the context tree for modeling multi-component map images. We propose a method for optimizing the context tree for a given image so that inter-layer dependencies are taken into account. The tree is constructed so that the template pixels can be located also in an already compressed reference layer. The method also optimizes the processing order of the layers so that inter-layer dependencies are maximally utilized. We apply the method in a static manner for each layer separately using a training image (static approach), or optimize and store the context tree for each image separately (semi-adaptive approach).

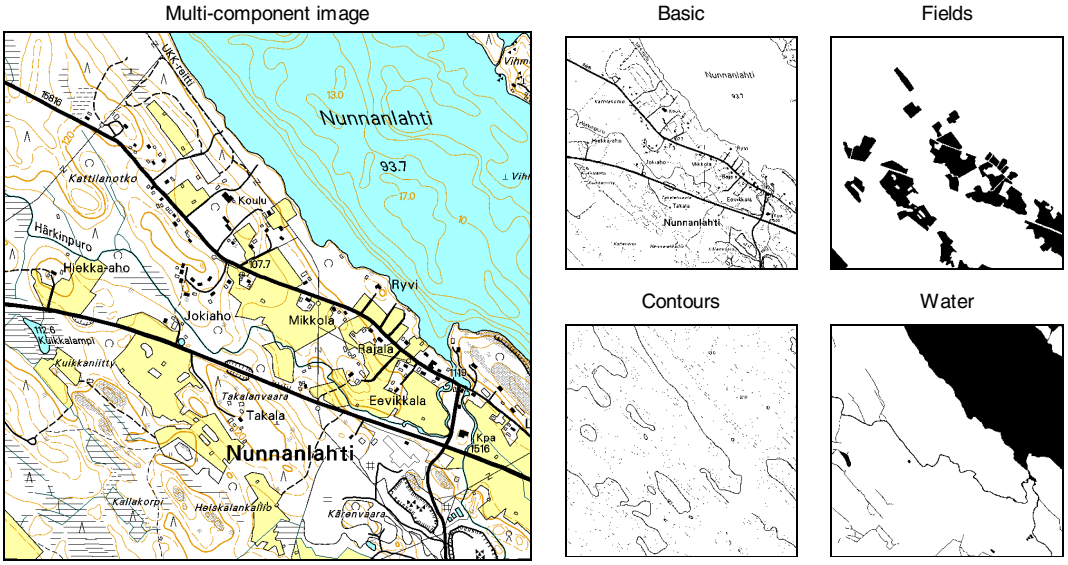

Figure 1: Illustration of multi-component map image. The shown fragment has the dimensions of 1000 × 1000 pixels.

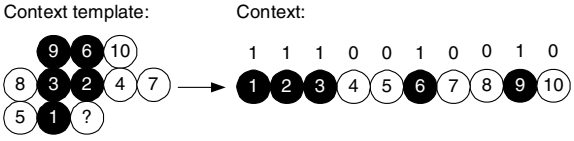

Figure 2: An example of a 10-pixel context.

2. Context tree modeling

Context tree is applied in the compression in a similar manner as fixed-size context templates; only the context selection is different. The context selection is made by traversing the context tree from the root to leaf, each time selecting the branch according to the corresponding neighboring pixel value. The leaf has a pointer (index) to the statistical model that is to be used. Each node in the tree represents a single context, as illustrated in Figure 3. 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, or optimized within a limited search area, relative to the compressed pixel position.

The tree can be trained beforehand (*static approach*), or optimized directly for the image to be compressed (*semi-adaptive approach*). In the latter case, an additional pass over the image will be required and the tree must also be stored in the compressed file. The cost of storing the tree structure is one bit per node, which is practically the same as two bits per context. If the position of the context pixel is optimized for each context, then this information must also be stored in the compressed file.

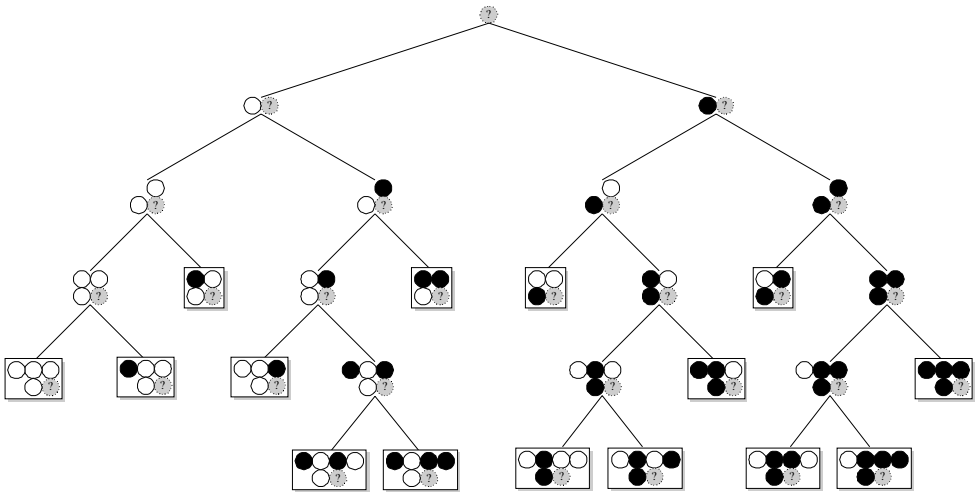

Figure 3: Illustration of a Context tree.

To construct a context tree, the image is processed and statistics 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 is not achieved from using the children nodes instead of their parent node, 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 is integrated into the *SplitCost* parameter. The code length can be calculated by summing up the self-entropies of the pixels as they occur in the image:

$$l(C) = \sum_t \log p^t(C) \tag{2}$$

The probability of the pixel is calculated on the basis of the observed frequencies using a Bayesian sequential estimator:

$$p^t(C) = \begin{cases} p_W^t(C) = \dfrac{n_W^t(C) + \delta}{n_W^t(C) + n_B^t(C) + 2\delta}, & \text{if } t^{\text{th}} \text{ pixel is } \textit{white} \\ p_B^t(C) = 1 - p_W^t(C), & \text{if } t^{\text{th}} \text{ pixel is } \textit{black} \end{cases} \tag{3}$$

where n_W^t, n_B^t are the time-dependent frequencies, and p_W^t, p_B^t are the probabilities for white and black colors respectively, and $\delta = 0.45$, as in [2].

According to the direction of the pruning, the tree construction is classified either as *top-down* or *bottom-up*. 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}. A drawback is that the expansion of the tree can terminate too early in situations as shown in Figure 4 [10].

In *free tree* [9], the position of the context pixel is also determined at each step. When a new children node is constructed, all possible positions for the next context pixel are analyzed within a predefined search area, and the position resulting in maximal compression gain is chosen. A drawback of the free tree approach is that the position of the new context pixel must also be stored in the compressed file.

In bottom-up approach, a full tree of k_{MAX} levels is first constructed. The tree is then pruned one level at a time up to the level k_{MIN} using the same criterion as in the top-down approach. The bottom-up approach provides better optimization of the tree [11], but the position of the context pixels must be fixed.

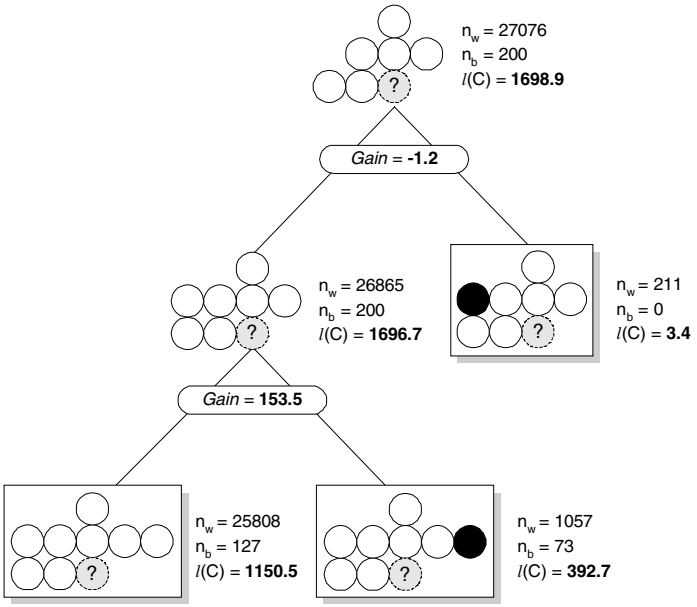

Figure 4: Example of tree construction.

3. Multi-level context tree

The multi-level context tree is constructed as follows. The tree is pruned using the top-down approach with delayed pruning technique as presented in [10]. The tree starts from scratch and the branches are expanded one pixel at a time. The location of the template pixels are optimized and fixed beforehand and then applied for every branch. Another approach is to optimize the location separately for every branch (Free Tree approach).

Pixels can be included from two layers: the current layer and another *reference layers*. The pixels in the current layer can be located in the neighborhood area including already coded pixels, but the pixels in the reference layer can be located anywhere in the image. The only limitation is that the reference layer must have been coded before the current layer. We use the joint 77 pixels neighborhood as shown in Figure 5.

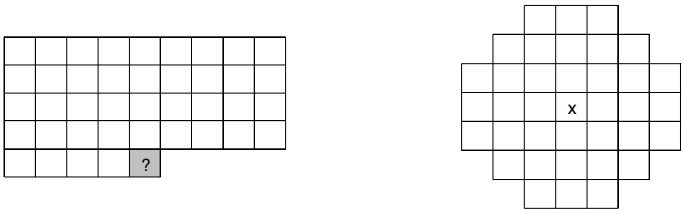

Figure 5: The neighborhood area for optimizing the location of the template pixels. The area in the current layer is shown left, and in the reference layer right.

Additional information from the reference layer can improve the probability estimation. For example, consider the example in Figure 6, where the compression of the two images via separate single-level context trees would result in 4854+1330=6184 bytes. If we use the information from the first layer when compressing the second layer, the tree structure of the second layer would be simpler. All information would be concentrated only in the first branch of the tree, as shown in Figure 7. The compression of the second layer would be only 146 bytes, and the final size of the compressed file 4854+146=5000 bytes. This kind of situation can happen when the layers are generated by color separation from the original color image.

The map images have also other inter-layer dependency. For example, the same pixel is usually not set in the water layer and in the field layers at the same time, although still possible as the layers are generated from map database. Another observation is that the basic and the water layers have some redundant information along the rivers and lake boundaries. In general, anything is possible, and it is not easy to observe the existing dependencies by the eye. The dependencies, however, can be automatically captured by the statistical modeling.

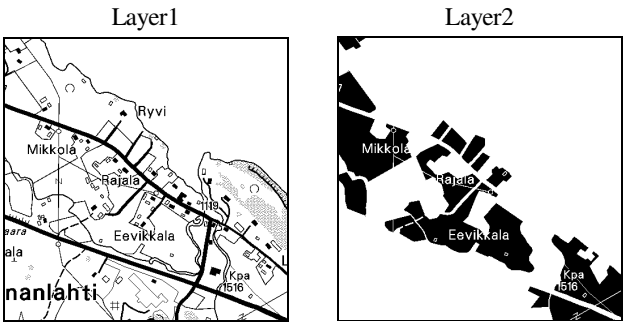

Figure 6: Example of two layers obtained by color separation.

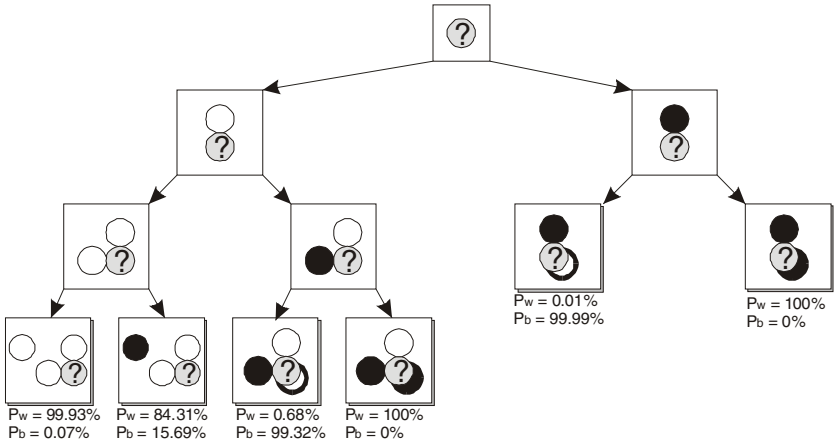

Figure 7: Example of a two-level context tree, in which two context pixels are from the current layer and one from the reference layer (shown below the current pixel '?').

The existing dependencies are demonstrated in Figure 8 for the case of the NLS map images. We can see that there are significant inter-layer dependencies, especially between the *basic* layer and the two other layers (*water*, *field*). The *contour* layer, on the other hand, is independent from all other layers. The main observation is that we cannot utilize all the existing dependencies as the order of processing restricts which layers we can use as the reference layer.

For example, if we compress the basic layer first, we can then improve the compression of the water layer by 52% (118705 bytes). The opposite order would improve the compression of the basic layer by 35% (345061 bytes). It is easy to see that the best order for these layers would be to compress the water layer first, the basic layer second, and then fields layer last. The contours layer should be processed either the first or the last so that it would not affect the compression of other layers.

In general, we can select any predefined order on the basis of known (or assumed) dependencies. If we do not know the image source beforehand, we should optimize the order of the layers for maximal utilization of the inter-layer dependencies. The selected processing order can be stored in the compressed file by few bits. The best ordering can be achieved as follows.

Suppose that we have k layers. We first try out all pairwise dependencies by tentatively compressing every layer using each other as reference. The result would be a $k \times k$ matrix consisting of the absolute bit rates for every *layer-reference layer* pairing. Using the information of this matrix, we can calculate the result of all $k!$ possible permutations for the processing order. Assuming that the number of layers is small (with the NLS images k=4), this is not a problem. With larger values of k, the problem would reduce to an interesting problem that is closely (but not exactly) related to the *minimum spanning tree* problem.

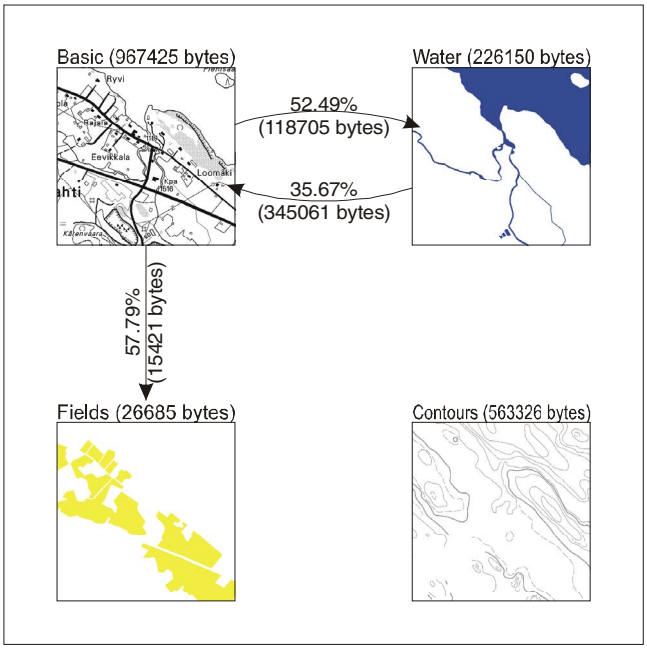

Figure 8: The arrow links show the inter-layer dependencies as a number of saved bits the first image is used as reference image when compressing the next one.

4. Experiments

We compress a set of topographic map images originating from the NLS topographic database [1]. We use five randomly chosen images from the database corresponding to the map sheets *No/No* 431306, 124101, 201401, 263112, and 431204. The images are of the size 5000×5000 pixels, and have the original scale 1:20 000. This corresponds to a resolution of two meters per pixel. The topographic map images are composed of four semantic layers:

- *basic* – topographic image, supplemented with communications networks, buildings, protected sites, benchmarks and administrative boundaries;
- *fields* – solid polygonal regions;
- *contours* – thin lines representing the elevations levels;
- *water* – solid regions, and various width lines representing lakes, rivers, swamps, water streams.

Two variants are implemented and tested: *Context tree* refers to the fixed order of the context pixels, and *Free tree* to the variable ordering. We consider also both *semi-adaptive* and *static* approaches. The static tree is trained for the image 431306, which contains most common geometrical structures. The semi-adaptive tree is optimized for the input image to be compressed. In the semi-adaptive approach, the cost of storing the tree is included into the splitting criterion. The additional cost is 2 bits per node for *Context tree*, and $2+\lceil \log(77) \rceil = 9$ bits per node for *Free tree*. Thus, we evaluate the following methods:

- static Context tree (*Static CT*)
- semi-adaptive Context tree (*Semi-adaptive CT*)
- static Free tree (*Static FT*)
- semi-adaptive Free tree (*Semi-adaptive FT*)

Table 1 gives the results when compressing each layer separately, and Table 2 when using additional information from the reference layer. Sequential JBIG using a context template of 16 pixels is used as point of comparison. The results and corresponding processing sequences are summarized in Figure 9. The results show that the best compression is obtained with *Static Free Tree* when compressing each layers separately (Table 1), and by *Semi-adaptive Free tree* when multi-level context tree is applied (Table 2). The most important contexts and their statistics are shown in Figure 10.

Let us study the found dependencies. There is a static dependency *Fields-Basic*, that can be used when *Basic* and *Fields* layers are present. We just have to ensure that *Basic* layer is compressed first as the additional information is used during the compression of *Fields* layer. The *Contours* layer can be placed anywhere in the processing sequence because no layer depends on it, and this layer does not depend on other layers. Two other cases of the dependencies were found: *Basic-dependent* and *Water-dependent*.

To sum up, the multi-level context tree improves the compression performance by 15% on average compared to *Context-tree* without using multi-level ability, and 25% compared to the sequential JBIG with context template of size 16.

Table 1: Summarized results of compressing each layer separately. The numbers are given as bytes, and as the relative improvement in comparison to JBIG 16.

	Static CT		Semi-adaptive CT		Static FT		Semi-adaptive FT		JBIG 16
Basic	846220	12,53%	961977	0,56%	847667	12,38%	900706	6,90%	967425
Fields	24453	8,36%	27270	-2,19%	23349	12,50%	25346	5,02%	26685
Contours	502119	10,87%	538435	4,42%	501279	11,01%	529324	6,04%	563326
Water	197592	12,63%	219807	2,80%	195297	13,64%	207136	8,41%	226150

Table 2: Layer-by-layer compression results using multi-level context tree. The numbers are given as bytes, and as the relative improvement in comparison to JBIG 16. The dependent layers are shown in the brackets.

Sum	Static CT		Semi-adaptive CT		Static FT		Semi-adaptive FT		JBIG 16
Basic (Water)	791657	18,17%	783046	19,06%	672812	30,45%	622364	35,67%	967425
Fields (Basic)	12054	54,83%	11963	55,17%	11173	58,13%	11264	57,79%	26685
Water (Basic)	120676	46,64%	126004	44,28%	96792	57,20%	107445	52,49%	226150

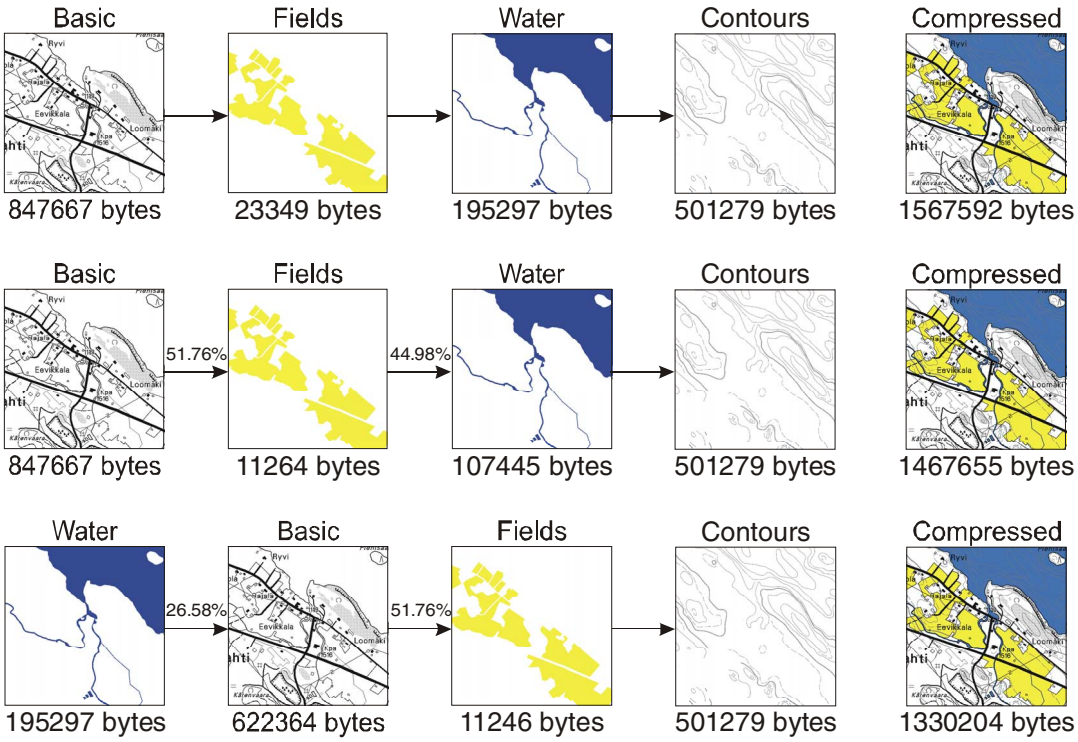

Figure 9: Three different processing sequences. The first one compresses each layer separately, and the other two using two different processing order indicated by the arrows. Fixed-size template of 16 pixels would sum up to 1783586 bytes (JBIG-16).

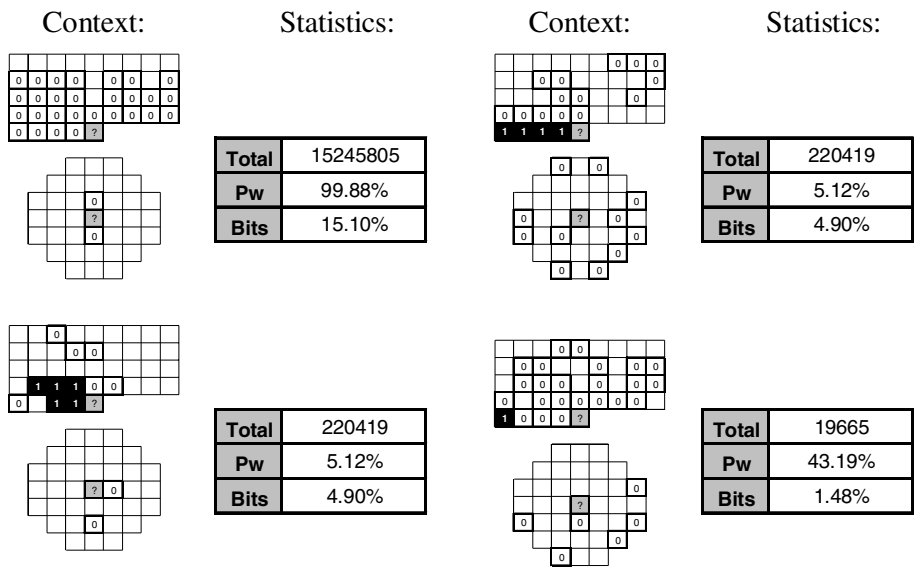

Figure 10: Four most used contexts for the compression of *Basic* when the *Water* is used as the reference layer.

5. Conclusions

Multi-level context tree was proposed for compressing multi-component map images. Both the tree and the processing order of the layers are optimized. The proposed technique achieves of about 25 % compression improvement over static 16-pixel context template, and 15 % over similar single-level context tree. A drawback of the method is that it requires heavy optimization in the compression end.

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] JBIG: ISO/IEC International Standard 11544, *ISO/IEC/JTC1/SC29/WG9;* also ITU-T Recommendation T.82. Progressive Bi-level Image Compression, 1993.

[3] JBIG2: Final committee draft for ISO/IEC International Standard 14492, 1999. (http://www.jpeg.org/public/jbigpt2.htm)

[4] Howard P.G., Kossentini F., Martins B., Forchammer S., Rucklidge W.J., Ono F., "The emerging JBIG2 standard", *IEEE Trans. Circuits and Systems for Video Technology* 8 (7): 838-848, 1998.

[5] Rissanen J.J., Langdon G.G., "Universal modeling and coding", *IEEE Trans. Information Theory* 27: 12-23, 1981.

[6] Rissanen J.J., Langdon G.G., "Arithmetic coding", *IBM Journal of Research, Development* 23: 146-162, 1979.

[7] Lin Y., Wang Y.J., Fan T.H., "Compaction of ordered dithered images with arithmetic coding", *IEEE Trans. Image Processing* 10 (5): 797-802, May 2001.

[8] Ageenko E.I., Kopylov P., Fränti P., "On the size and shape of multi-level context templates for compression of map images", *IEEE Int. Conf. on Image Processing (ICIP'01)*, Thessaloniki, Greece, vol. 3: 458-461, 2001.

[9] Martins B., Forchhammer S., "Bi-level image compression with tree coding", *IEEE Trans. Image Processing* 7 (4): 517-528, 1998.

[10] Fränti P., Ageenko E.I., "On the use of Context tree for binary image compression", *Proc. IEEE International Conference on Image Processing (ICIP'99)*, Kobe, Japan, vol. 3, 752-756, 1999.

[11] Ageenko E.I., Fränti P., "Compression of large binary images in digital spatial libraries", *Computers & Graphics* 24 (1): 91-98, February 2000.