Optimal layer ordering in the compression of map images

Pavel Kopylov and Pasi Fränti

Department of Computer Science, University of Joensuu, FINLAND.
E-mail: justas,franti@cs.joensuu.fi

Abstract:

We study the compression of color map images by context tree modeling and arithmetic coding. The key issue of this method is the attempt of utilizing the correlations between the color layers of the image, and to solve the optimal order of the layers as optimum branching problem. The acquiring of the inter-layer dependencies is done by optimization of the context tree for every pair of image layers. The cost matrix of the inter-layer dependencies is then solved by Edmond's algorithm for optimum branching.

1. Introduction

Usually, map images contain much less color information than photographic images. In fact, the number of colors in a typical map image is less than 256. Among different lossless methods tailored for the compression of such images, the most known are dictionary based methods: GIF and PNG. The CompuServe Graphics Interchange Format (GIF) is based on LZW dictionary compressor [1]. PNG (Portable Network Graphics) encodes the image using the *deflate* [2] algorithm, which is a combination of LZ77 dictionary compression [3] and Huffman coding.

Statistical methods such as JPEG-LS [4] and JBIG [5] does not perform well enough for the map images. JPEG-LS uses linear predictive model, which works well on natural images where spatially adjacent pixels tend to have similar values but not so well on map images. JBIG uses local context modeling, which works well on binary images where the limited number of colors allows the use of a reasonably sized context model.

One way to handle map images is to separate image color information into a set of binary layers based on color or other predefined information, and compress them separately.

In Embedded Image-Domain Adaptive Compression of Simple Images (EIDAC) [6] three-dimensional context model was proposed. The method is tailored for compression of grayscale images, but also have bit-plane nature. The algorithm processes the image bit planes one by one, but the context pixels are selected not only from the current bit plane, but also from already processed layers.

In another approach called SKIP pixel coding [7], the binary layers of the image are acquired by color decomposition. The coding sequence proceeds layer by layer. In a particular layer, if a given pixel has already been coded in a layer of higher priority it does not need to be coded in the current layer or any of the lower layers. Thus the coding of large amount of not relevant information about blank areas could be "skipped".

It was also shown that the location of the context components, and the shape of the context are crucial for obtaining better compression rates [8][9]. Moreover, use of the *context tree* [10] provides a more flexible approach for modeling, and thus allow us to achieve more accurate probability estimates without leading to sparse context problem.

Also the discovery and the use of inter-layer dependencies allow us to model the correlation between image layers. It was shown in [11] that the use of multi-level context tree for multi-component map images allows us to achieve about 25% of compression improvement over JBIG method.

In this paper we propose to exploit inter-layer dependencies by using the additional information from different layers during the compression of the current one. This can be done by solving the dependency correlation as a directed minimum spanning tree problem (also known as optimum branching), and to apply multi-layer context tree compression of the layers.

2. Context based compression

Statistical image compression consists of two distinct phases: *statistical modeling* and *coding* [12]. In the modeling phase, we construct the probability distribution for the occurrence of the symbols to be compressed. The coding can be efficiently performed using *arithmetic coding*, which is an optimal coding for a given probability model [13].

A binary image can be considered as a message, generated by an information source. The idea of statistical modeling is to describe the message symbols (pixels) according to the probability distribution of the source alphabet (binary alphabet, in our case). Shannon has shown in [14] that the information content of a single symbol (pixel) in the message (image) can be measured by its *entropy*.

The dependencies can be localized to a limited neighborhood, and described by a *context-based statistical model* [15]. In this model, the pixel probability is conditioned on the *context* C, which is defined as distinct black-white configuration of neighboring pixels within the local template.

In principle, better probability estimation can be achieved using a larger context template but this leads to the *context dilution* problem, in which the statistics are distributed over too many contexts.

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 [10]. The contexts are represented as *binary tree*, and the context is constructed pixel by pixel. 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.

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 parent 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 code length can be calculated by summing up the self-entropies of the pixels as they occur in the image.

According to the direction of the pruning, the tree construction may be classified either as *top-down* or *bottom-up*. In the 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}. *Bottom-up* approach constructs a full tree of k_{MAX} levels, which 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 [16] but the position of the context pixels must be fixed.

In *free tree* [10], 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.

3. Map images representation

A map image could be provided us as a result of rasterization of vector map format files like Simple Vector Format (SVF), Scalable Vector Graphics (SVG) or ERSI ArcShape [17], as a set of binary images with different semantic meaning. In case that we will consider topographic map series 1:20 000 of *National Land Survey of Finland* (NLS) [18], each binary image represents different kind of logical information on the map like: Basic for buildings, communication networks, Fields, Water, and Elevation lines.

The size of each image is 5000×5000 pixels, and represents a 10×10 km^2 area. The map image can then be easily reconstructed by combining the binary layers, and displayed to the user as color image, as shown in Fig. 1.

On the other hand, the map image could be provided not as a set of semantically separated binary layers, but as a color image without any additional information about semantics of the image. Then, in order to proceed with compress of this map image by context based compression technique we have to perform the *color separation* as preliminary step to produce binary layers. The algorithm for the color separation starts by analyzing the amount of the presenting colors in the image.

This approach has one side effect: it will produce the set of binary layers, which consist of the same number of distinct pictures, as the number of colors in original image. In other words, it will separate an image into all possible image colors. So, if we consider the map image in Fig. 1 as an example, the resulting number of layers will be 5, not 4, because the background color (white) will also be a separate layer.

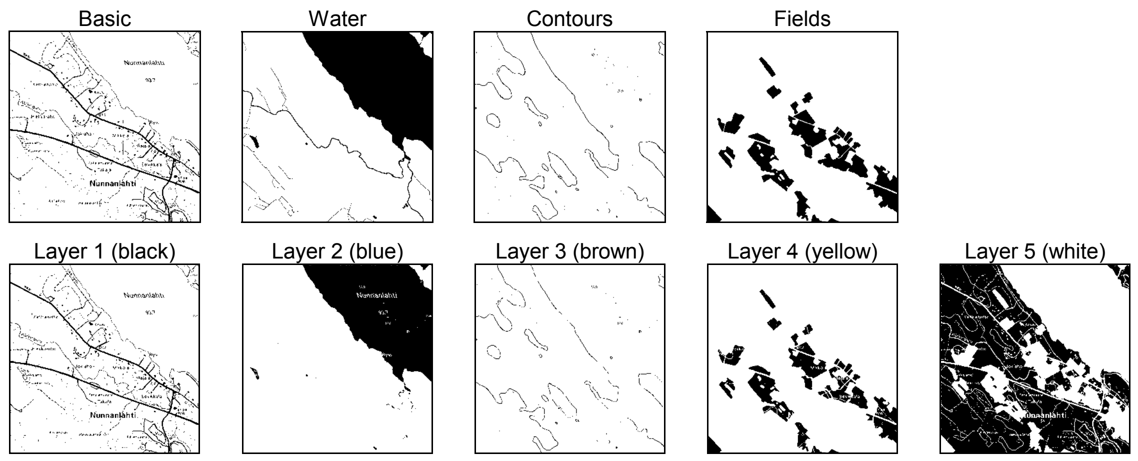

Figure 1: Illustration of a multi-component map image. The shown fragment has the dimensions of 1000×1000 pixels.

4. Compression of binary layers

The straightforward approach is to compress the image layers separately. For this we can use fixed-size context template defined by standard 1-norm and 2-norm distance functions, see Fig. 2. The size of the context template is usually a parameter of the compression algorithm, and it mainly depends on the size of the image.

It is also possible to optimize the size and shape of the template for the given image layer to be compressed at the cost of longer compression time [8]. The optimal context template can be solved by compressing the image using all possible templates and selecting the one with achieved the best compression result. However, there are an exponential number of different template configurations as a function of the template size. A more practical approach is to optimize the locations for one pixel at a time.

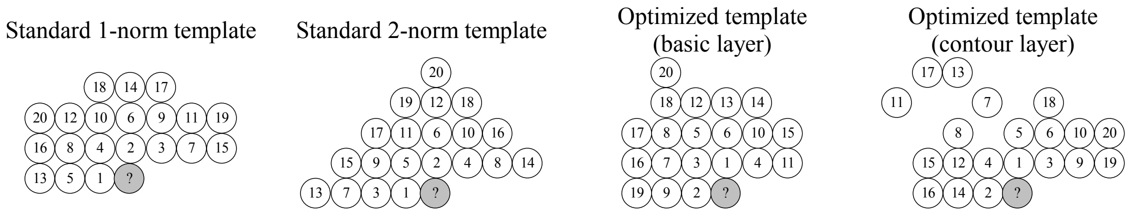

Figure 2: Alternative orderings for the context templates.

The idea of *multi-layer* context template is to utilize the information from additional image layer, referred here as the *reference image*. The restriction on the use of the reference image is that it must already have be coded so that both encoder and decoder have the same information. The main difference in the construction of *single-layer* and *multi-layer* context templates is in additional neighborhood mask used for selection of the pixels from the reference image. The pixels in the current layer must be already coded pixels, but in the reference the pixels can be anywhere in the image.

The idea of utilizing multi-layer dependencies can be extended also to the context tree modeling. The multi-level context tree is constructed as follows. 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).

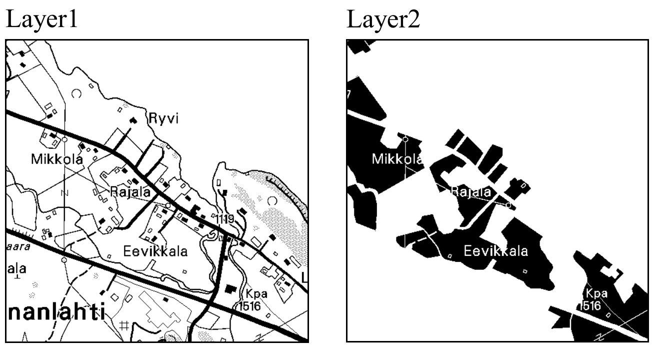

Figure 3: Example of two layers obtained by color separation.

The use of the information from the reference layer will allow us in some cases to increase the compression ratio of the single layer up to 50% according to [11]. In fact, if we will consider the compression of sample images in Fig. 3, the compression of these two images separately using single-level context trees would result in 4854+1330=6184 bytes. On the other hand, 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 Fig. 4. Thus the compression of the second layer would be only 146 bytes, and the final size of the compressed file 4854+146=5000 bytes.

The map images usually have inter-layer dependencies. For example, the same pixel is usually not set in the *water* layer and in the *field* layers at the same time although it is possible as the layers are generated from map database independently from each other. Another observation is that the *basic* and the *water* layers have 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.

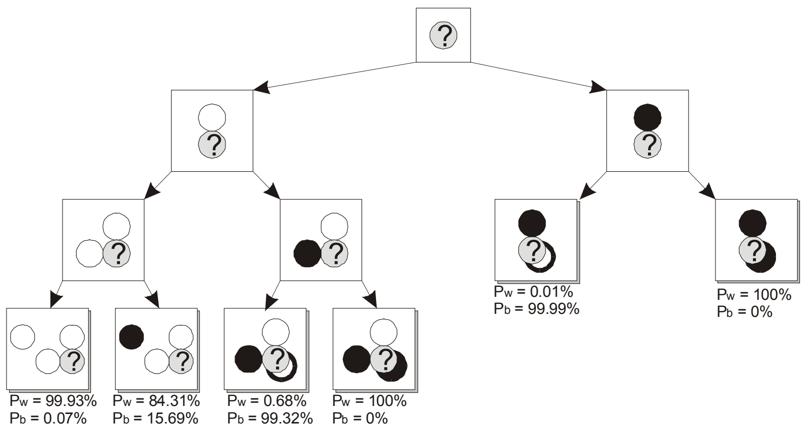

Fig. 4: Example of a two-level context tree, in which two context pixels are taken from the current layer and one from the reference layer (shown below the current pixel '?').

5. Optimization of the order

The existing dependencies are demonstrated in Fig. 5, in the case of the NLS map images. There are significant inter-layer dependencies 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.

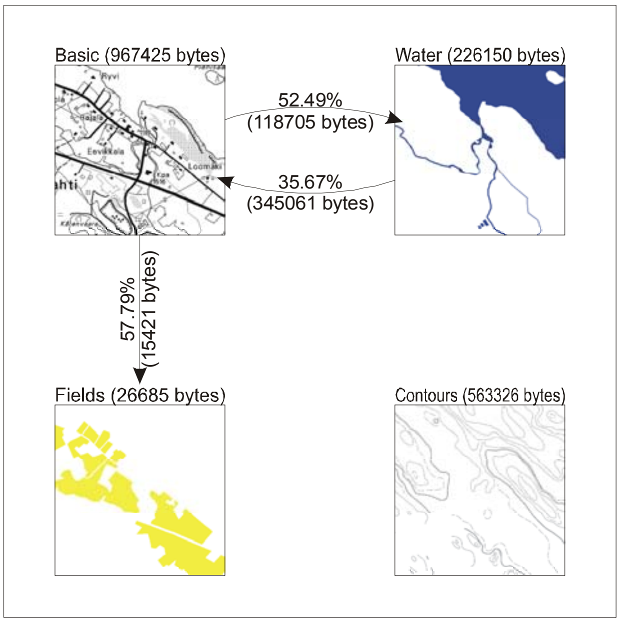

Figure 5: The arrow links show the inter-layer dependencies as a number of saved bits the first image is used as reference image when compressing the next one.

In general, we can select any predefined order because 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 only. The best ordering can be achieved as explained in the following subsections.

5.1. Construction of the cost matrix

Suppose that we have *k* layers. To obtain the best possible order we have to study out all pairwise dependencies by tentatively compressing every layer using each other as a reference layer. The result would be a *k*×*k* *cost 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. If the number of layers is small enough (with the NLS images *k*=4), this is not a problem. With larger values of *k*, when the image was generated by color separation, this will result in a longer computational time.

On the other hand, not all information in the matrix is relevant to us. In the case when there are no dependencies between the layers, the corresponding compression result would be the same (or worse) with or without the use of inter-layer context model. We can therefore reduce the amount of information in the cost matrix by subtracting the original values by the values obtained by layer-independent compression, and then eliminate all values less than or equal to zero. The resulting matrix is shown in Table 1.

The reduced cost matrix is considered as a directed graph with *k* nodes. The task of obtaining the optimal order is closely related (but not exactly) related to the *minimum spanning tree* problem. We follow the approach taken by Tate for optimal band ordering in compression of multi-spectral images [19].

Table 1: Example of the cost matrix.

	0	1	2	3	4	5	6	7	8	9	10	11
0		4283	7110	0	0	0	0	729	226	0	0	0
1	6293		4082	0	3	259	0	1218	288	334	118	1049
2	9963	3253		0	0	0	0	1888	0	0	0	0
3	61	0	0		98	325	3008	1113	0	0	0	860
4	0	0	0	0		0	0	0	0	0	0	0
5	158	0	0	119	0		0	2474	11	0	0	501
6	0	0	0	1570	53	122		1801	0	0	0	2570
7	3254	1698	3006	615	88	3249	2895		108	1253	883	9475
8	641	0	0	0	0	0	0	0		0	0	0
9	0	0	110	0	0	0	0	0	0		2094	285
10	415	230	818	0	12	102	0	1391	0	4984		1011
11	0	0	0	0	0	0	635	5945	0	97	0	

5.2. Solving optimum branching

A spanning tree of a graph is a subset of the edges that contains all the nodes of the graph. The *minimum spanning tree* (MST) is a spanning tree with the minimum sum of the weights of the edges included in the given graph. The minimum weighted tree can be solved in polynomial time using *Prim's algorithm* [20], for example. However, there are few differences that separate our problem of obtaining the optimal order from the minimum spanning tree problem:

- We have a directed graph whereas the MST is defined with undirected graph.
- We can have only one incoming edge for any node.
- We can have several separate spanning trees instead of only one.
- We have maximization problem.

The first two differences make the problem as a *directed spanning tree problem*. The directed spanning tree is defined as a spanning tree where all nodes (except the root) have exactly one incoming edge. This is also known as the *optimum branching* problem [21], and it can be solved in $O(n^2)$ time [22].

In the optimal ordering, it is not necessary to have a single spanning tree but we can have separate sub graphs, see Fig. 5. This means that we should actually find spanning forest instead of a single tree. The problem was considered as the *maximum spanning forest* problem in [19]. However, we have eliminated all negative weights in the cost matrix (Table 3), and the inclusion of a zero-edge can be considered as independent compression of the corresponding layers. Thus, we can still consider the optimal ordering as maximum directed spanning tree problem. For simplicity, we apply the Edmond's algorithm as proposed in [21], see Figure 6.

The optimal branching for the data in Table 1 is shown in Figure 7, and the corresponding directed minimum spanning tree in Figure 8. This ordering of the layers sums up to 124 977 bytes, which corresponds to the improvement of 24.79 % in comparison to the original result.

```
Given:
  Connected graph G=[V,E]
  Solution set S=[V,E]

MST_For_Directed_Graph(G,S)

  FOR (each Vi) DO
    Ei = FindMinEnteringEdge(Vi,G);
    AddEdgeAndItsEndpoints(S,Ei);

  C=LocateCycles(S);
  IF (C != empty) THEN
    FOR (each Ci)
      Ee=FindEnteringEdges(G,Ci);
      CalculateModifiedCost(Ee);
      Eem=FindMinEdge(Ee);
      ReplaceEdge(Ec,Eem);
```

Figure 6: Edmond's algorithm.

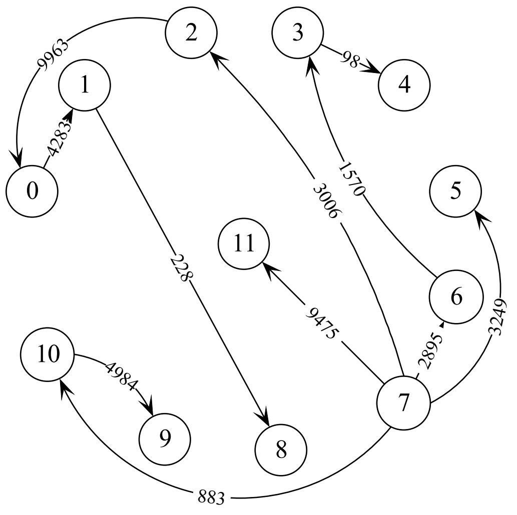

Figure 7: Optimum branching for Table 1.

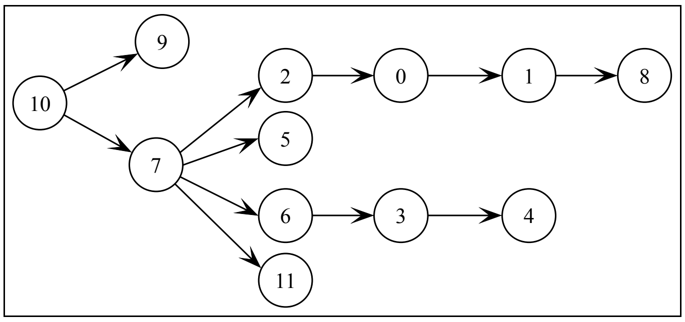

Figure 8: The corresponding directed MST of Figure 7.

5.3. Selection of the background color

In the case of color separation, we can also eliminate one layer completely and consider it as the *background color*. Usually the background color is white but this is not necessarily the case always. In fact, we can set any of the layer as the background color. The consequence is that the chosen layer is not compressed. There are two obvious choices for selecting the background color:

- *Greedy*: The layer with the maximal compressed size.
- *Optimal*: The layer of whose removal gives most improvement in compression.

The greedy choice is not necessary the best because of the inter-layer dependencies. In other words, the background layer cannot be used as a reference layer, and therefore the removal of dependent layer can increase the compressed size of other layers.

The optimal choice can be obtained by considering the removal of all possible layers. This is computationally feasible because the problem of finding the optimal ordering takes $O(n^2)$ where n is the number of nodes in the initial graph, and is typically small (e.g. n=4). Thus, we can find the optimal background color at most in $O(n^3)$ time. The bottleneck of the optimization is the calculation of the numbers in the matrix at the first place, not solving the graph problem.

6. Experiments

We evaluate the proposed technique by compressing five sets of map images as shown in Fig. 9. In the beginning, all images were passed trough color separation. The obtained binary layers were then compressed using context tree modeling and arithmetic coding.

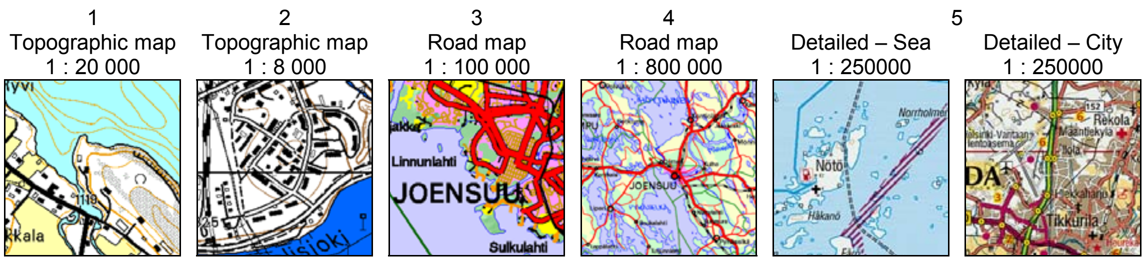

Figure 9: Sample 256x256 pixel fragments of the test images.

The results are summarized in Table 2 with the context tree modeling for each layer separately (CT), and with multi-level context tree (MCT). Both variants are considered with and without the optimal removal of the background color. The results show that the file size of the CT is about 40% more than that of the MCT. The results of the MCT are then compared with other compression methods in Table 3.

Table 2: The average results for each set of test images in kilobytes, and percent of improvement in comparison with the CT.

	CT	CT w/o background		Multi Level CT		Multi Level CT w/o background	
1:20000	960	616	35,79 %	577	39,94 %	401	58,23 %
1:8000	64	34	46,28 %	36	44,06 %	30	53,38 %
1:100000	243	185	23,74 %	194	20,10 %	155	35,97 %
1:800000	268	240	10,45 %	219	18,33 %	198	26,22 %
sea	181	125	30,71 %	124	31,28 %	106	41,12 %
vantaa	215	190	11,76 %	186	13,53 %	164	24,02 %
Sum	**1930**	**1391**		**1335**		**1054**	

Table 3: The comparison of compression methods in bytes.

	GIF	**PNG**	**JBIG 16**	**PPM**	**PWC**	**SKIP**	**MCT**
1:20000	1801	1854	1018	1449	777	532	401
1:8000	86	90	62	81	35	34	30
1:100000	288	278	283	203	198	202	155
1:800000	303	287	274	211	197	198	198
sea	150	155	181	113	124	117	106
vantaa	225	212	238	172	155	172	164
Sum	**2853**	**2876**	**2056**	**2229**	**1485**	**1256**	**1054**

7. Conclusions

We have proposed a method for compressing multi-level context tree modeling and optimizing the order of processing the layers. The optimal order of processing the layers was solved by Edmond's algorithm as for the directed minimum spanning tree problem.

References

[1] T.A Welch., "A Technique for High-performance Data Compression," *Computer* 17(6) pp. 8-19, June 1984.

[2] Peter Deutsch, "DEFLATE Compressed Data Format Specification," *rfc1951*, http://www.cis.ohio-state.edu/htbin/rfc/rfc1951.html, May 1996.

[3] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Transactions on Information Theory*, 23(3) pp. 337-343, May 1977.

[4] M. Weinberger, G. Seroussi, G. Sapiro, and M. W. Marcellin, "The LOCO-I Lossless Image Compression Algorithm: Principles and Standardization into JPEG-LS," *HPL98-193*, HP Labs, 1998.

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

[6] Y. Yoo, Y. Kwon and Ortega A. "Embedded Image-Domain Adaptive Compression of Simple Images," *32 Asilomar Conf. on Signals, Systems and Computers, Nov. 1998.*

[7] S. Forchhammer and O. Riis, "Content Layer Progressive Coding of Digital Maps", *IEEE Proceedings Data Compression Conference*, Snowbird, Utah, USA, pp. 233-242, March 2000.

[8] E.I. Ageenko, P. Kopylov and P. Fränti "Optimizing context template for compression of multi-component map images", *GraphiCon'00*, Moscow, Russia, pp. 151-156, 2000.

[9] E.I. Ageenko, P. Kopylov and P. Fränti, "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, pp. 458-461, vol.3, October 2001.

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

[11] P. Kopylov and P. Fränti, "Context tree compression of multi-component map images", *IEEE Proc. Data Compression Conference*, Snowbird, Utah, USA, pp. 212-221, 2002.

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

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

[14] C.E. Shanon, A mathematical theory of communication. *Bell Syst. Tech Journal* 27: 398-403, 1948.

[15] G.G. Langdon, J. Rissanen, Compression of black-white images with arithmetic coding. *IEEE Trans. Communications* 29 (6): 858-867, 1981.

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

[17] ESRI, "ESRI Shapefile Technical Description", An ESRI White Paper, 1998. (http://www.esri.com/library/whitepages/pdfs/shapefile.pdf)

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

[19] S.R. Tate, "Band ordering in lossless compression of multispectral images", *IEEE Trans. on Computers*, 46(4): 477-483, April 1997.

[20] R.C. Prim. Shortest connection networks and some generalizations. *Bell Systems Technology Journal,* 36:1389-1401, 1957.

[21] J. Edmonds, "Optimum branchings", *J. Research of the National Bureau of Standards*, 71B, 133-240, 1967.

[22] R.E. Tarjan, "Finding Optimum Branchings", *Networks*, 7, 25-35, 1977.