Symbol representation in map image compression

Akimov Alexander and Pasi Fränti
Department of Computer Science
University of Joensuu, Finland
+358-13-2517931
akimov@cs.joensuu.fi, franti@cs.joensuu.fi

ABSTRACT
We propose map image compression system, in which we separate text and symbol information from the rest of the data. The text and other symbols are stored as one bitmap for each symbol into a dictionary. The technical challenge of the work is to convert the symbol data directly to output format similar to that of the JBIG2 standard. In this way, the text elements and special symbols are compressed more efficiently but we still have the maps in compatible raster image format.

Categories and Subject Descriptors
I.4.2, I.4.9: Image compression, Applications.

General Terms
Algorithms.

Keywords
Map images, compression, symbol representation, navigation.

1. INTRODUCTION
Maps are widely available in raster, as well as vector format. Vector maps may be displayed in any resolution and are thus convenient for zooming. Panning can be performed by retrieving neighboring map blocks. However, mobile devices have limited computing and storage capabilities and are thus unable to handle a complete Database Management System (DBMS). The use of vector data is thus impractical. In addition, maps are sometimes unavailable in vector format, while the various formats and incompatibility between different systems may restrict their use.

On the other hand, current compression technologies allow efficient storage and transmission of raster maps via wireless networks. This means that when maps need only to be viewed at the mobile device, storage and map sheet generation in varying scales at the server side becomes a feasible solution. The DBMS may be located at the server side, whereas the client stores maps only when needed [1]. The advantages of this approach are: (1) it does not depend on any database or vector format, as raster maps can be easily generated and reproduced from any source format, including paper maps. (2) Only modest memory and computing capabilities are required, allowing operation on a mobile device.

SAC'04, March 14-17, 2004, Nicosia, Cyprus

Map imaging system, as proposed in [1], consists of two steps at the server side: rasterization and compression. In the first step, the map data is converted into a set of binary raster layers, and then compresses the binary layers separately [2]. At the client side, the map images are decompressed and reconstructed for viewing. On the basis of this approach, dynamic map imaging application could be constructed using low cost devices as outlined in [3].

In this paper, we improve the map image compression system by separating the text and symbol information from the rest of the data. Instead of rasterizing the symbols into the final images, we store only one bitmap for each symbol into a special symbol dictionary. The appearances of the symbols are coded as pointers to the dictionary. Note that we do not need to perform any character recognition procedure in the raster images, because of the ability to recognize original symbols in text format.

The proposed approach is supported by the JBIG2 file format [4], which has been applied for storing map images in [5] although without utilizing the dictionary structure before. In the proposed approach, we store the text elements and special symbols more efficiently. The difference to JBIG2 is that we can skip the refinement step, i.e. we do not compress the difference of the real and the matched symbol as, in our case, they are identical.

To sum up, we exploit the redundancy of repeating the same symbols in the image, and yet have the images in a compatible raster format. The rasterization is still done at the server side in the compression phase, and at the client side, the maps are available in compatible raster format independent from the used map database system.

2. OVERALL SYSTEM DESCRIPTION
The main steps of the proposed map image compression system are shown in Figure 1. We exploit the vector information in the compression of text and map symbols, and use the special features of JBIG2 tailored for text information. Specifically, we rasterize text information separately from the rest of the vector data.

Map images can be a result of rasterization of vector map format such as Simple Vector Format (SVF) [6], Scalable Vector Graphics (SVG) [7] or ERSI ArcShape [8]. The map server can provide the maps as a set of layers with different semantic meaning. For example, the topographic map series 1 : 20 000 of National Land Survey of Finland (NLS) [9] divides the information into four logical layers: basic (topographic data and contours), elevation lines, fields and water (see Figure 2). The size of each layer is 10000×10000 pixels representing a 10×10 km^2 area. The map image can then be reconstructed by combining the binary layers, and displayed as a color image. In this work, we consider the two input vector formats: ArcShape and SVF.

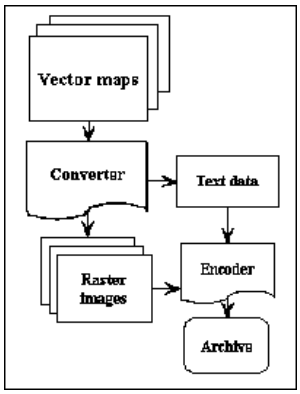

Figure 1. The principle scheme of the vector map rasterization and encoding process.

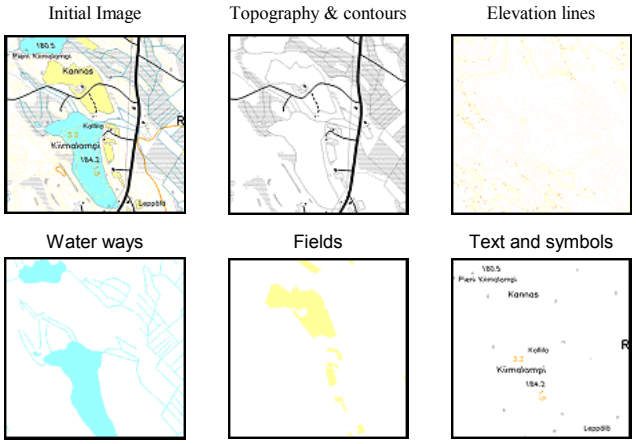

Figure 2. The layer division of the NLS map image.

2.1 Text and symbols in Vector Format

The ArcShape format [8] was developed by *ESRI* [10] for advanced map processing. It supports three types of primitives: *Point*, *Polyline* and *Polygon*. These primitives are sufficient for representing any map. The primitives could have additional parameters such as Z-coordinate (altitude) and a measure. The measure is a value associated with the primitive (for example, the number of road). We will not consider these parameters because they are used only in some special cases. Full technical description of ArcShape format is in [8]. We just mention that the ArcShape format uses ordered vertices for the text primitives. In this work, we developed the ArcShape for maps acquired from the NLS maps library, and therefore, we use some of these features.

ArcShape image consists of a set of *shapefiles*. Each shapefile stores primitives of a specified type (points, polylines or polygons). A map consists of a set of logical layers. Each logical layer is a set of four shapefiles (points, polylines, polygons and text). Shapefile belongs to some logical layer and has some primitive type depending on the file name. The main file in NLS map has the pattern name *?xxxxxx??.shp*. The sequence *xxxxxx* specifies a number of the map and is a constant for all files from one map, for example 431204. The first character of the name defines a logical layer that the shapefile belongs to. The following list shows all possible values of the first character:

- j — administration information,
- l — communication objects,
- m — areas (lakes, swamps, fields),
- n — water,
- r — buildings,
- k — elevation lines,
- s — conservations.

The character after the map number means the one of the quarter part of the ArcShape image, and the last character the type of the shape primitives included. For example, the shapefile with the name *m431204Bp* contains polygon primitives from top left part (part B) of the areas layer.

The database has information about the direction of the text primitives and arrow heads but no information about colour, line style and other important primitive attributes. The reason is that the database file locates in itself all textual information but not the graphical layout about how the map should be displayed. This feature of the NLS library makes the text extraction very simple. On the other hand, the graphical layout and the rasterization parts have had to be implemented ourselves.

SVF (Simple Vector Format) has been designed to be a simple format for describing vector images [6]. The basic drawing objects include points, lines, circles, arcs, Bezier curves and text. Features of the format include layers (for controlling the visibility of objects), hyperlinks (for allowing the user to click on a portion of the drawing to perform an action), notifications (for sending messages when the user has passed a certain zoom level), fills, and the ability to override the default colours. Text in SVF files can be displayed using a default system dependent courier-like font. Text is drawn using the current text height (default is 10). A width can also be specified with the text. If this value is 0, the text will be drawn using the font's default width. In the case, when the width is specified, the text will be scaled to fit. If the width of the text is important, because of the font metrics, which can differ between systems, then the text scaling procedure is suggested [6].

The proposed map rasterization scheme is shown in the Figure 3. During the rasterization we separate text information from the main image, which contains only generic info. The rasterization procedure generates two output streams. The first one contains the generic information of the map, and the second one the *symbol collection*. By the word *symbol* we mean here a small bitmap, which represents a character or other graphical symbol in the map. The bitmap in the collection is stored as a binary array, so it is necessary to associate it with a pair of numbers: the width and the height of the symbol image. Besides this, we have additional information for each text appearance, such as its location on the map, its color and its content. This data is unique for every text or symbol appearing on the map.

2.2 JBIG and JBIG2 compression standards

JBIG (*Joint Bilevel Image Experts Group*) is a lossless binary image compression standard [11], which uses pixel by pixel *context-based* compression. The combination of the neighboring pixels (given by a template) defines the context, and in each context the probability distribution of the pixels is adaptively determined on the basis of the already coded pixel. The pixels are coded by arithmetic coding according to their probabilities. The arithmetic coding component in JBIG is the QM-coder.

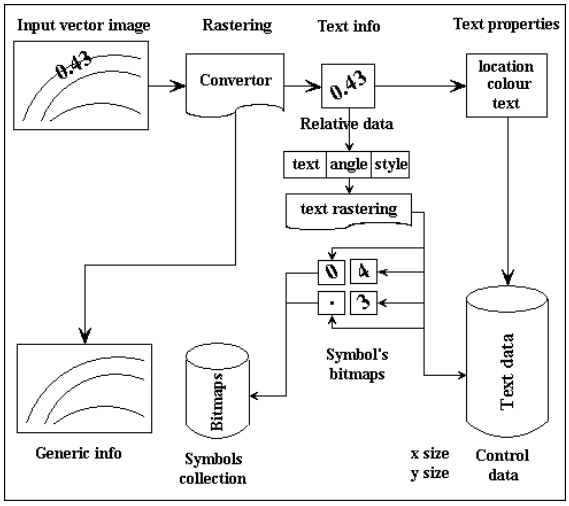

Figure 3. The rasterization scheme.

The newer standard JBIG2 [4, 12] enhances the compression of text images using pattern matching technique. The standard will have two encoding methods: *pattern matching & substitution* (PM&S), and *soft pattern matching* (SPM). The image is segmented into pixel blocks containing connected black pixels using any segmentation technique. The content of the blocks are matched to the library symbols. If an acceptable match (within a given error marginal) is found, the index of the matching symbol is encoded. In case of unacceptable match, the original bitmap is coded by a JBIG-style compressor. The compressed file consists of bitmaps of the library symbols, location of the extracted blocks as offsets, and the content of the pixel blocks.

The PM&S method works as follows [13]. The image is segmented into pixel blocks containing connected black pixels. These blocks are sequentially matched against representative symbol bitmaps from the adaptively constructed dictionary. If an acceptable match is found, the pointer to the corresponding bitmap in the dictionary and the position of the character on the page are encoded. If there is no acceptable match, the bitmap of the current pixel block is encoded using standard bitmap encoding techniques such as MMR or JBIG1, and added to the dictionary. The method allows high lossy compression levels, but results in infrequent but inevitable substitution errors. For cases where such errors are unacceptable, the *residue coding*, that is the refinement coding of lossy image back to the lossless original, or the SPM technique can be used.

3. TEXT PROCESSING

The text rasterization procedure uses Hershey vector font for drawing [14]. The vector font is useful for scaling, styling and rotation opposite to a raster font. A character is represented as a set of coordinate pairs. The coordinate pair (–1,–1) means pen up operation, for example. Thus, we can use an algorithm of drawing lines for output a text. Figure 4 demonstrates the character A with the coordinate pairs marked with dark grey color. The character is represented via six coordinate pairs: (0,0), (4,10), (8,0), (–1, –1), (2,5), (6,5), which is enough for drawing the character.

In the compressed file, the text elements are stored as bitmaps. From this point of view, a simple character is nothing more than a small binary image. For example, the same characters of the same size and style, but with different angles are represented as different symbols. On the Figure 5 shows the moment of rotation, when the same text in two cases produces different bitmaps.

Text rotation means rotation of the text around the point of text output: the pole of rotation (see Figure 6). The rotation is done directly before drawing a line between two coordinate pairs. Next routine makes rotation of a point around another point with some angle in radians using the standard matrix of rotation. The data about the received bitmap: width of rotated character, its height and its coordinates on the map are stored in the compressed file. This related data is stored in the special structure: *control data.*

The data in the *control data* structure is represented in Figure 7. It consists of four blocks: text numerical information, the array of *strip* properties, the array of the strip elements properties and the array character bitmap sizes. The text numerical information is the information about the number of different text appearing in the map file and information about the text colors, which were used in the map.

Let us define now the each different text as a *strip*. The strip can consist of one character or of several pages of text (the name of the country "*Great Britain*" on a map would be one strip). But each strip element has the same properties such as text size, text color and close location of each symbol to the previous one. The array contains information about the coordinates of the beginning of the strip and the strip's color.

The next information block has the data about the strip properties: the array with the lengths of the strips, the coordinates of each strip elements, and the indexes of these elements. To decrease the amount of data, we use Huffman coding for compressing the coordinate differences. To decrease the data redundancy, the array of coordinates is divided into two parts: the first one contains the information about the x coordinates and the second one contains the information about the y coordinates. The last array is built from the indexes of the strip elements.

The last block of the information is not related with the strips and contains the horizontal and vertical sizes of the character bitmaps. The arrays in the last two blocks are encoded by the Huffman algorithm, where each array is encoded separately.

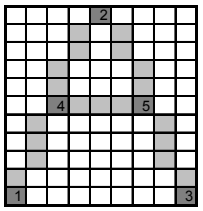

Figure 4. An example of the Hershey font.

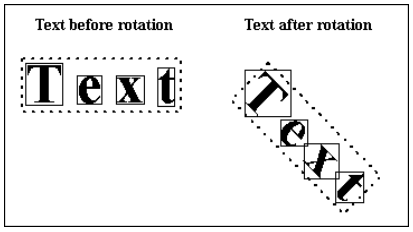

Figure 5. Text rotation example.

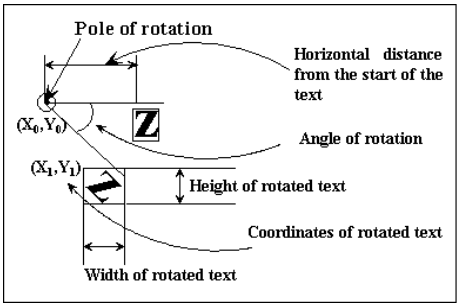

Figure 6. Text rotation process.

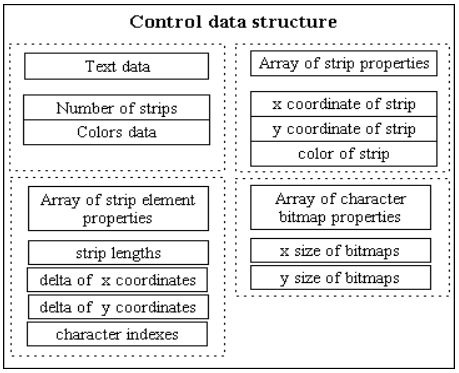

Figure 7. Control data structure.

4. PROPOSED COMPRESSION METHOD

4.1 Encoding procedure

A new map image format called MISS (Map Image Storing System) was recently developed as a part of the DYNAMAP project [1, 2]. The maps in the MISS format are separated into several layers, which were further divided into blocks for supporting map browsing in mobile devises. Because of expensive data transferring, the map server does not send the entire map, where user is traveling, but a little piece around the current location of the user. This concept is referred as *dynamic map handling*. It allows map browsing without having the entire map in the memory. New image blocks are added dynamically during the browsing. This approach allows saving a lot of memory and makes the map structure flexible. New blocks can be requested via network communications. Transferring is effective, when blocks are sent in the compress format. Therefore, MISS format should be able to extract a block from MISS file and import it without decompression.

The file structure of MISS is represented by the following substructures: *page, layer and block* [8]. The MISS page structure is the general structure; it stores geographic information of the map, a page resolution, a background color and an array of pointers to its layers. One MISS file can contain several pages with different maps. The next structure after the page is the layer. The layer represents encoded binary layer, which corresponds to some color from the image. The layer contains the filename of the input image, color, shifting in the map (usual equals zero), separation to blocks, additional data for decompression and an array of blocks. Block is the lowest level structure of the MISS format. It is an encoded bitmap, with the predefined size of the layer. The blocks are encoded independently from each other. The principal structure of the MISS file is shown below in the Figure 8. The advantage of this format is direct access to the blocks. So if we know the index of the block we can then extract it from the MISS archive without decompression of the entire file.

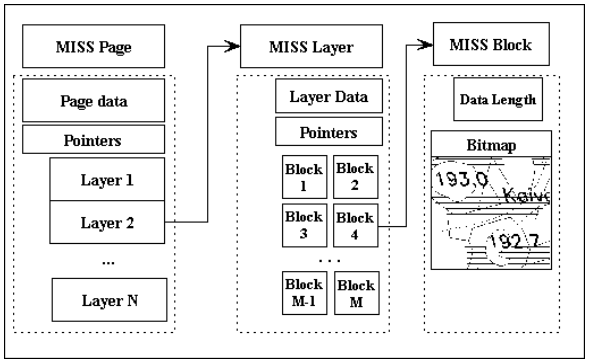

Figure 8. MISS files structure.

We will not consider all the features of the given file format, but let's say some words according to our task. When MISS encoder starts to operate, it has side by side with the set of binary images, the text data and the collection of symbols. The original MISS file format was changed to allow the compression of these two additional structures. The basic principles of this modification are shown in Figure 9. In specific, we add one additional layer to the main structure called *text layer*. It contains all information from the control data structure, all non-raster information from the symbols collection and, finally, all symbols. This structure was chosen because of the ability of MISS to the direct access to the blocks in the layers, so the process of symbol decompression will become much easier (details in the next chapter).

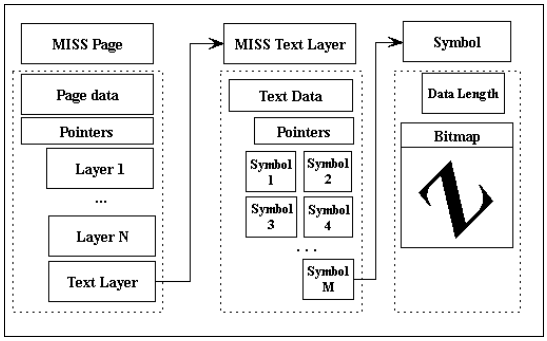

Figure 9. Modified MISS file format

The scheme of compression one layer is represented in Figure 10. As it is shown the blocks are formed before compression. So the coder defines the bounds of compression, if it deals with the generic info layer, or the bounds of compression are the sizes of the symbols, if the compressor deals with the text layer. The encoder after this encodes the blocks separately and independently from each other. The encoded bitmaps are placed in the compressed file. To reconstruct the whole layer we need to decode each block and unite the resulting bitmaps into one according to the block's indexes.

But in the case of the text layer we have situation when the encoder needs to store a lot of relative data. That is the reason why this data is also compressed. The compression is very simple:

due the flexibility of the MISS standard we are able to insert into layer any kind of blocks. So the data is represented as the binary image and the encoder compresses it as well as other symbols.

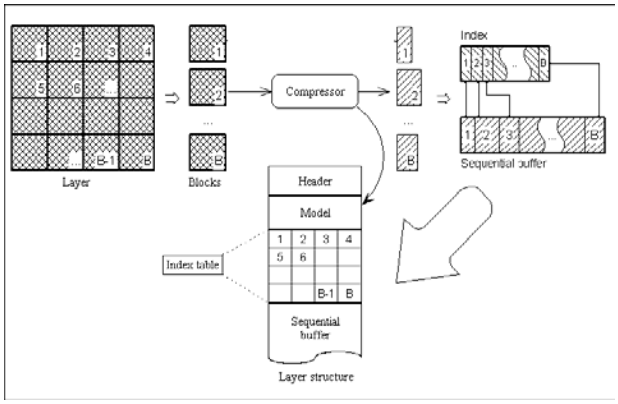

Figure 10. The layer encoding scheme.

4.2 Decoding procedure

The JBIG2 compression standard distinguishes the following four procedures:

1. The generic region decoding procedure, which decodes a bitmap treating it simply as an array of binary pixels.
2. The generic region refinement decoding procedure, which decodes a bitmap by treating it as an array of binary pixels, but coding each pixel with respect to some reference bitmap.
3. The text region decoding procedure, which decodes a bitmap by drawing a collection of symbols into it, possibly applying the generic refinement region decoding procedure to each one.
4. The halftone region decoding procedure, which decodes a bitmap by placing a collection of patterns into it, at locations specified by a halftone grid.

The given algorithm uses the generic region and text decoding procedures.

The generic decoding procedure is used to decode a rectangular array of 0 or 1 values, which are coded one pixel at a time (i.e., it is used to decode a bitmap using simple, generic, coding). The decoding procedure also modifies the array of probabilities which may be used by other invocations of this generic region decoding procedure. The generic region decoding procedure may be based on sequential coding of the image pixels using JBIG1 coding. This procedure is used to decode the binary layers.

The text region decoding procedure is used to decode a symbol-coded bitmap by placing a set of symbols on it. If the symbols bitmaps and the control data about the image have been obtained, then the symbol-coded image decoding procedure support parallel decoding. The symbols decoding process includes the decoding of the symbol's bitmaps data (its horizontal and vertical sizes) and the bitmaps as themselves. The bitmaps decoding algorithm fully repeats the decoding algorithm for generic regions. The control data decoding process decodes the array of text elements properties. The result of the decoding is a combination of the binary layers and the symbols, which are placed on the map according to the information from the control data.

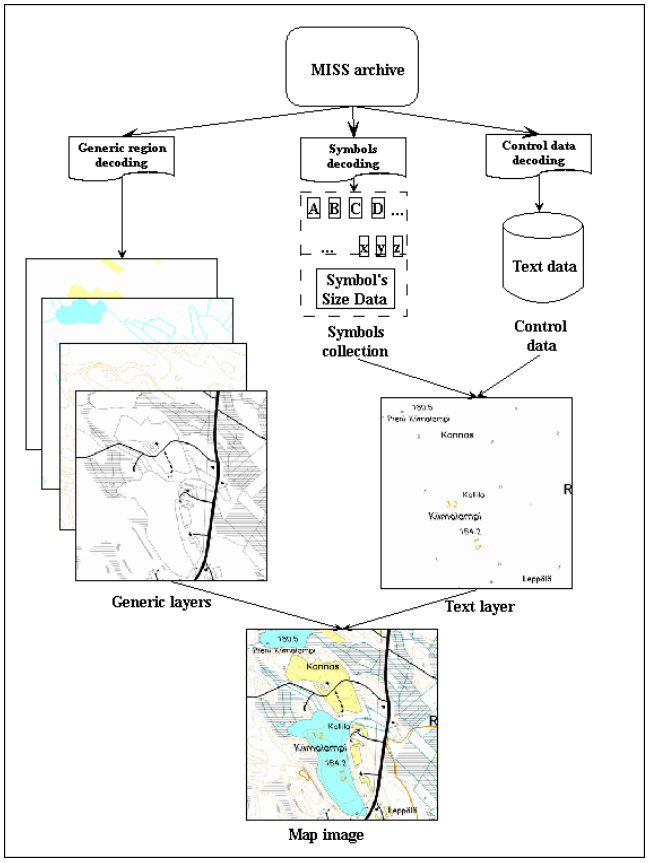

Figure 12. The decoding scheme.

5. EXPERIMENTS

We used a set of maps from the NLS topographic database [9]. The maps are the 431204, 124101, 201401, 431306 region maps. To compare the efficiency of the proposed algorithm we compressed the maps by the old version of the MISS encoder, which is the usual JBIG compression and processed compression by the previously described method. To estimate the compression results more precisely, we considered the compression of the whole image with independent compression of the layers. The results of the encoding are shown in Table 1. The results of the compression of the first two layers are almost the same, as the layers do not have text information on them. The difference in the compressed file sizes is given in bytes. There are also statistical data that have been received during the compression. Table 2 describes statistics of the text rasterization process as the number of different characters, the number of text appearances on the map, and the size of the compressed text layer. The elements of Table 2 are:

- The size of the dictionary measured as the number of different bitmaps in the symbol collection
- The number of symbols measured as the overall number of the character bitmaps appearing on the map.
- Strip data measured as the number of bytes needed to store all side information about the strips (blocks 1 and 2 in the chapter 3).

- X sizes measures as the number of bytes of the compressed array of horizontal sizes of the bitmaps.
- Y sizes measured as the number of bytes of the compressed array of vertical sizes of the bitmaps.
- Strip lengths measured as the number of bytes of the compressed array of the strip lengths.
- ΔX coordinates measured as the number of bytes of the compressed array of the ΔX coordinates of the strip elements.
- ΔY coordinates measured as the number of bytes of the compressed array of the ΔY coordinates of the strip elements.
- Bitmap's indexes measured as the number of bytes of the compressed array with indexes of the strip elements.

Table 1. Compression results (bytes).

	MISS			
	124101	**201401**	**431204**	**431306**
Layer 1	213	44475	7966	38716
Layer 2	36817	129258	452048	192257
Layer 3	26447	531510	461415	589455
Layer 4	136559	678780	607846	576388
Whole map	199757	1383766	1528996	1396537
	Proposed method			
	124101	**201401**	**431204**	**431306**
Layer 1	239	44501	7989	38742
Layer 2	36843	129291	452074	192280
Layer 3	27123	539624	466143	594323
Layer 4	116806	622925	578238	508052
Whole map	186514	1336733	1503048	1331644

Table 2. Statistical data of the text rasterization process.

	124101	**201401**	**431204**	**431306**
Size of dictionary	344	804	718	822
Number of symbols	2791	5440	9222	17726
Size of compressed blocks				
Strip data	9930	13238	9222	17726
X sizes	351	733	609	736
Y sizes	453	706	633	731
Strip lengths	165	368	245	387
ΔX coordinates	1068	2173	1529	2707
ΔY coordinates	672	1270	914	1530
Bitmap's indexes	3332	5574	4249	7045

6. CONCLUSIONS

We have proposed method for compressing vector maps by rastering them and excluding textual information into a special symbol library. The system architecture is based on the MISS [1,2] storage system, which adds some elements to the JBIG2 standard. The main goal of the method was to improve the compression by exploiting vector information. The experiments showed that the separation of the text and symbols from the generic info improved compression within the limits that the maps are stored in compatible raster formats. Thus, the method retains independent from the used vector database system.

7. REFERENCES

[1] P. Fränti, P. Kopylov and V. Veis, "Dynamic use of map images in mobile environment", *IEEE Int. Conf. on Image Processing (ICIP'02)*, Rochester, New York, USA, vol. 3, 917-920, September 2002.

[2] P. Fränti, E. Ageenko, P. Kopylov, S. Gröhn and F. Berger, "Map image compression for real-time applications", *Joint Int. Symposium on Geospatial Theory, Processing and Applications (Geomatics2002)*, Ottawa, Canada, July 2002.

[3] P. Fränti, "Dynamic Map Handling", *GIM International*, 17 (1), 28-31, January 2003.

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

[5] P. Fränti, E. Ageenko, P. Kopylov and S. Gröhn, "Compressing multi-component digital maps using JBIG2", *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing, (ICASSP'02)*, Orlando, Florida, USA, vol. 3, 2677- 2680, May 2002.

[6] Soft Source, "*Information on SVF (Simple Vector Format)*" http://www.softsource.com/svf/

[7] J.D. Eisenberg, *SVG Essentials*, O'Reilly, 2002.

[8] ESRI, "*ESRI Shapefile Technical Description*", An ESRI White Paper, 1998. http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf

[9] NLS, "*Technical description of NLS map format*" http://www.nls.fi/kartta/selosteet/ts/maastotietokanta.html

[10] M. Zeiler, "*Modeling Our World*", ESRI, 1999.

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

[12] JBIG Committee, *Coding of Still Pictures*, ISO/IEC/ JTC1/SC29/ WG1 N1359, 1999.

[13] P.G. Howard, "Text image compression using soft pattern matching", *The Computer Journal* 40 (2/3): 146-156, 1997.

[14] P. Bourke, "*Hershey Vector Font*", 1997. http://astronomy.swin.edu.au/pbourke/other/hershey/