UNIVERSITY OF JOENSUU
DEPARTMENT OF COMPUTER SCIENCE
Report Series A

Storage of multi-component digital maps using JBIG2 image compression standard

Pasi Fränti, Eugene Ageenko and Sami Gröhn

Report A-2001-2

ACM I.4.2, I.5.3
UDK 519.712
ISSN 0789-7316
ISBN 951-708-976-7

Storage of multi-component digital maps using JBIG2 image compression standard

Pasi Fränti, Eugene Ageenko and Sami Gröhn

Department of Computer Science, University of Joensuu
Box 111, FIN-80101 Joensuu, FINLAND
Email: franti,ageenko,sgrohn@cs.joensuu.fi

Abstract: Digital maps can be stored and distributed electronically using compressed raster image formats. Spatial access must be implemented to enable the user to operate directly on the compressed data without retrieving the entire image. In this work, we study how the latest JBIG2 standard can be used for storing the map images. The image tiling can be supported quite easily by storing each image block as its own segment. A slightly more difficult problem is how to store the multiple image components because the standard does not have any direct support for it. There are alternative ways to overcome this problem by utilizing the features of JBIG2 in creative ways. We show by experimental results that the block division of 128×128 can be implemented with about 15-30 % overhead.

Keywords: Map images, real-time applications, personal navigation, image compression, spatial access, JBIG2.

1. Introduction

Digital maps can be stored and distributed electronically using compressed raster image formats to be used on CD-ROM, web, or in mobile navigation devices. We use images from the Topographic Map Series 1:20 000 of Finland [7]. The images consist of several logical binary layers, which together form the computer-generated color image. The main problem for the use of the images is their huge size. For example, a single map sheet of 5000×5000 pixels representing a single map sheet of 10×10 km^2 requires about 12 Mb. This is usually too much for compact mobile devices that are used in navigation.

Another highly desired feature 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 be a significant feature in personal navigation using compact devices with limited size of display. Spatial access can be supported by tiling, i.e. dividing the image into fixed size rectangular blocks [1]. The blocks are then compressed independently by any compression method. The main problem is to combine the spatial access with efficient compression.

JBIG [6] and JBIG2 [5] are the latest standards for binary image compression. They use context-based statistical modeling with arithmetic coding, and achieve compression ratio of about 20:1 for typical map images. In addition to that, JBIG2 includes new

useful features such as image segmentation, symbol dictionaries, and multiple pages. These new features are important when compressing multi-component images, which are further divided into smaller blocks in order to allow spatial access. Especially the possibility of compose the image from multiple segments is highly useful here.

In this work, we study how JBIG2 standard can be used for storing multi-layer map images. The image tiling can be supported quite easily by storing each image block as its own segment. A slightly more difficult problem is how to store the multiple image components because the standard does not have any direct support for it. There are alternative ways to overcome this problem by utilizing the features of JBIG2 in creative ways.

We show by experimental results that the block division of 128×128 can be implemented with about 6 % overhead. Surprisingly, the biggest problem turns out to be the way the direct access is implemented in JBIG2. It includes offsets of the data segments in the file header and, in this way, allows direct access to the compressed segments. Unfortunately, the spatial location of the segments are left out from the file header. This means that even more creative solutions should be used if we are going to use JBIG2 for the purpose.

The rest of the paper is organized as follows. In Section 2, we discuss digital map images and the application specific requirements. The JBIG and JBIG2 standards are summarized in Section 3. In Section 4, we study different ways how the JBIG2 standard can be used for storing the multi-layer map images integrated with spatial access. Compression results are given in Section 5, and conclusions are drawn in Section 6.

2. Digital map images for real-time navigation

Digital maps provide visual view on a given geographic location that can be used for applications dealing with spatial data, *geographic information systems* (GIS), or in *personal navigation*. We can select user-specific views of the map server for different applications. The main goal is to have the maps available in real-time and independent of the location of the user, and without excessive computing resources. Typical navigation devices have limited memory resources and very narrow wireless communication channel (or no communication channels at all).

Because of limited resources, the end-user cannot store large map databases with specific software for the database management. A much simpler but still useful approach is to provide the user with digital map images in compressed raster format [2]. In this way, the user will get almost all the same benefits as he would get with the access to traditional geographic information systems with map databases. In addition to that, the user would not need to have huge hardware and software resources.

We consider topographic maps provided by the *National Land Survey of Finland* (NLS) [7]. The maps are represented here as raster images of size 5000×5000, each map divided into several binary layers:

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

Application can show the map on screen by plotting each component with its specific color as shown in Fig. 1. The color of a pixel in the image is determined by checking the pixels in each layer. If the pixel is set as "foreground color" in any layer, the pixel is colored by the appropriate color assigned to that particular layer. Otherwise the pixel is plotted by the default background color. The colors and the components to be shown can be specified by the user or it can be predefined in the application. Next, we define important properties that must be taken into account in the storage of the map images.

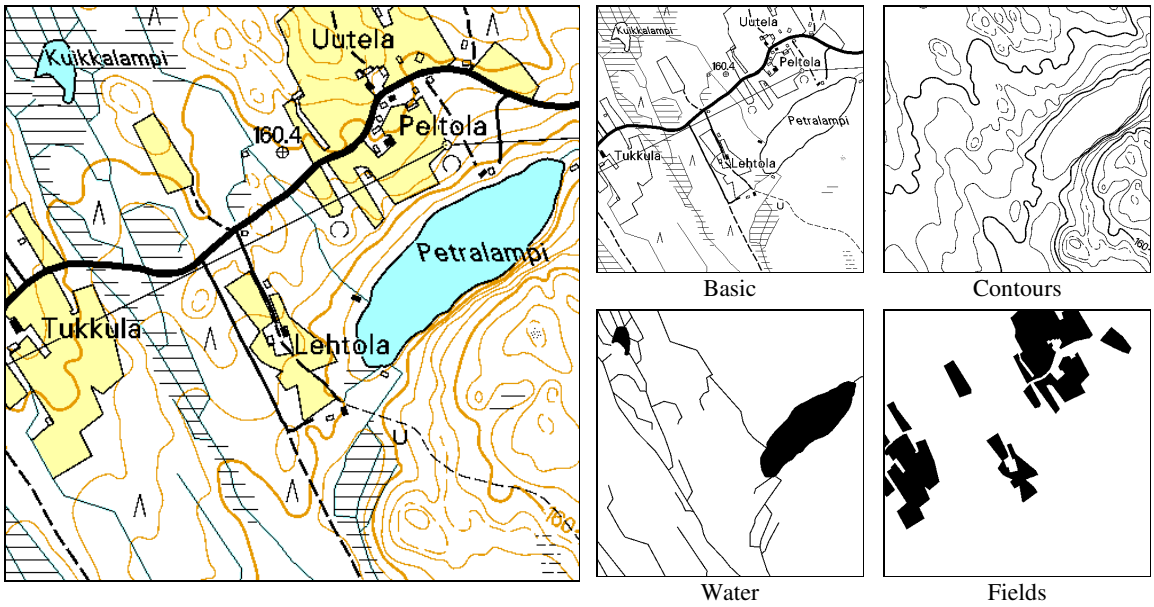

Figure 1: A sample multi-layer digital map of size 500×500.

2.1 Compression

The images can be generated from the database beforehand and stored in the navigation device, or the maps can be requested on-line using any suitable communication channel. Because of the limited memory and channel resources, the images must be stored compactly and restored efficiently. Latest binary image compression standards JBIG and JBIG2 are good candidates for the compression task.

Another aspect in the compression is to have fast decompression. One might tolerate longer compression times if it can be performed off-line. The decompression, on the other hand, is performed real-time and must be fast so that the system does not loose its interactivity because of the decompression delays. The decompression of JBIG is reasonably quick and there are additional techniques to speed the method if necessary.

2.2 Multi-scale maps

The user must have the capability to browse the map with variable resolution. This can be achieved using (1) progressive compression method as presented in JBIG, or (2) by storing multiple maps representing the same area with various scale. The first approach have the same image re-scaled for several different resolutions. The compressed file contains the lowest resolution image followed by the higher resolution images in an increasing order. The lower resolution images can be utilized for improving the compression of the following higher resolution images. Typically, the additional lower resolution versions add about 10% to the file size.

The second approach has the advantage that the maps can be optimized for each resolution separately. For example, maps with different scale can have different colors and layout, see Fig. 2. The different scales can be stored independently from each other, which means that the approach adds 33% to the storage size compared to a single image of the highest resolution (assuming that the scales decreases by a factor of 2). Another problem is that the transition between the maps of two different scales is not necessarily smooth enough if the difference in scale is more than 2-3 times.

The principle difference between these two approaches is that the maps with different scales have usually different level of details. In other words, some objects may be present in one scale but absent in another scale. On the other hand, if we would apply the resolution reduction for generating lower resolution image, the level of details remains the same as in the original image. Therefore, the maps with significantly reduced resolution would look messy.

We combine the two approaches by having independent images of different scale, and generate additional intermediate scales using the resolution reduction algorithm of JBIG, see Fig. 3. With this method, we can generate images of satisfactory quality, which is usually enough to fill the resolution gap between the existing scales.

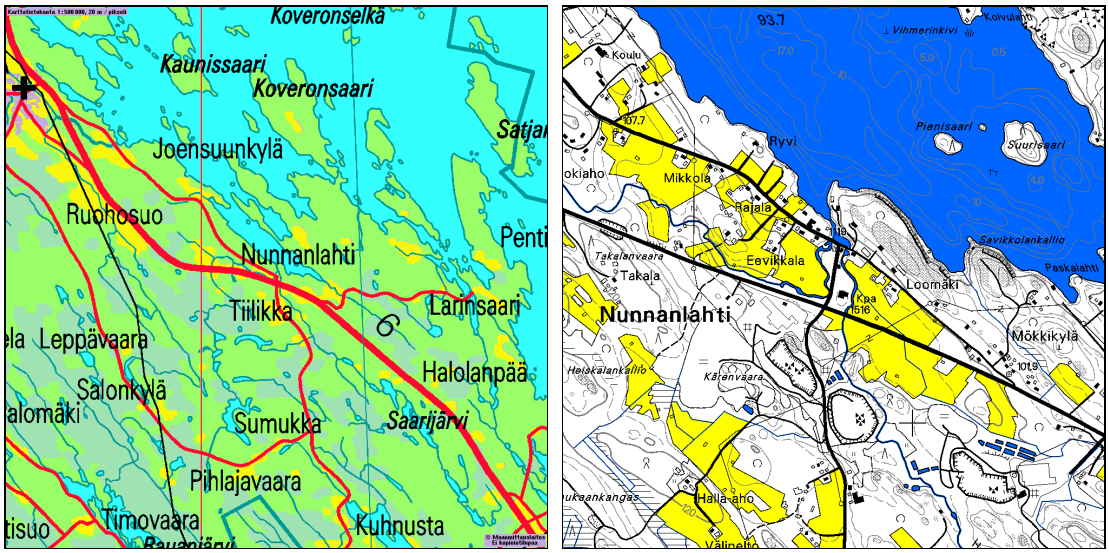

Figure 2: Images with two different scales. The left one is from the library with the scale 1:80 000 and the right one from the library with the scale 1:20 000.

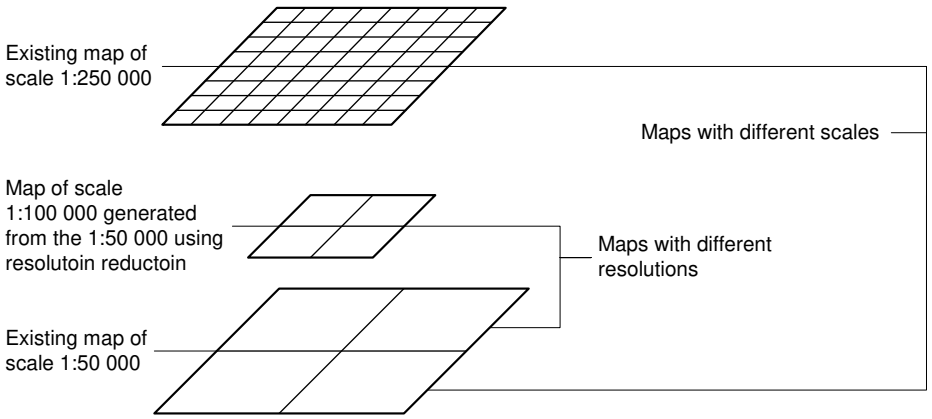

Figure 3: Illustration of the method to implement multiple scale images.

2.3 Spatial access

Spatial access enables the user to operate directly on the compressed data. The actual viewing device can be very small, e.g. 200×200 pixels, and it is therefore important to have direct access to an image fragment in the compressed file without having to retrieve and decompress the entire image. If spatial access is supported, an image may be interactively browsed on the viewing device. When the image is scrolled, a new part of data is retrieved and decompressed on the fly. In this way, spatial access eliminates time delays caused by image decompression and transfer.

Spatial access can be supported by tiling, i.e. partitioning the image into fixed size rectangular blocks and compressing them independently of each other. Pointers (*block indices*) are stored in the beginning of the compressed file to indicate the starting location of each block in the file, see Fig. 4. The blocks can then be decompressed in random order. The use of this kind of file organization is illustrated in Fig. 5.

The block size is a trade-off between compression efficiency and decoding delay. If very small block 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 block boundaries. The index table itself requires space and the overhead is relative to the number of blocks.

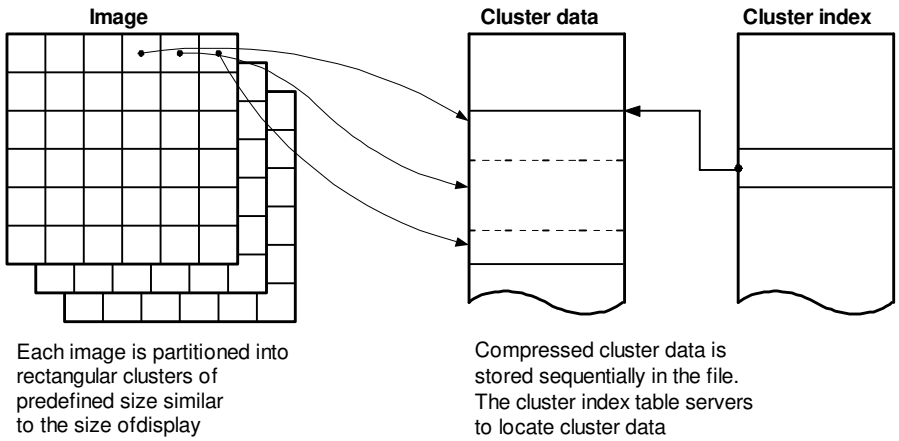

Figure 4: Diagram of the block decomposition.

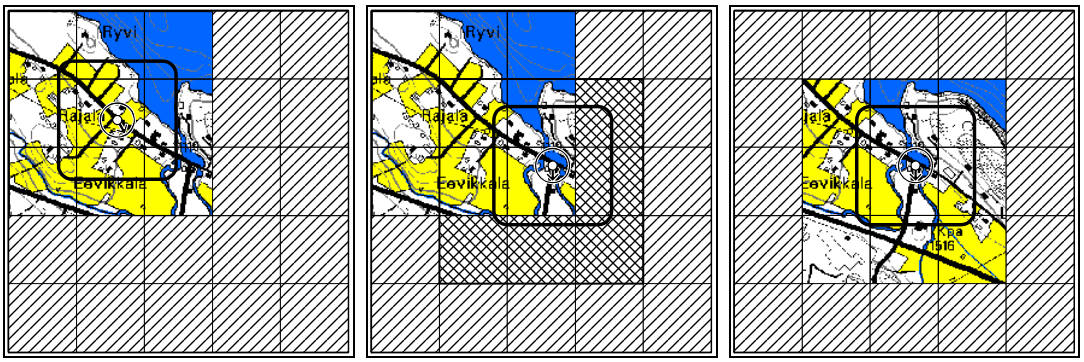

Figure 5: Use of method in clients device: Nine blocks are first decompressed and stored in the cache (left). Change of location is then registered (middle), and new image blocks are then decompressed and the view updated (right).

3. JBIG and JBIG2 standards

JBIG (*Joint Bilevel Image Experts Group*) is an international standard for compression of bi-level images in communications [6]. The standard defines two methods for bi-level compression, *progressive* and *sequential*. In sequential coding, the image is coded in raster scan order using a *context-based probability model* and *adaptive arithmetic coder* (QM-coder) [9]. The probability distribution of the black and white pixels is conditioned on the context, which is defined by the combination of already coded neighboring pixels. A three-line ten-pixel template is used by default, see Fig. 6. Both encoder and decoder estimate the model dynamically during the compression. The estimation starts from scratch and adapts the model to the input data. The probability estimation in the

QM-coder is derived from the arithmetic coder renormalization and is based on the Bayesian estimation concept [8].

The emerging JBIG2 standard enhances the compression of text images as they contain many repeated symbols. Instead of coding all the pixels of every symbol occurrence, it is possible to code only the bitmap of one representative instance of the symbol. There are two encoding methods used in JBIG2: *pattern matching and substitution* (PM&S) and *soft pattern matching* (SPM).

The PM&S method works as follows [3, 10]. 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.

The soft pattern matching differs from PM&S in that, in addition to a pointer to the dictionary and position information, it includes *refinement* data that can be used to recreate the original symbol, providing for lossless compression [4]. The SPM method is illustrated in Fig. 6. The only difference to PM&S (shown in *italics* in the figure) is that lossy direct substitution of the matched symbol is replaced by a lossless encoding that uses the matched character in the coding context. The refinement coding process is similar to the bitmap coding, with the difference that the two-layer context template is used. The template is shown in Fig. 7. It consists of a combination of four neighboring pixels from the input block, and seven from the bitmap of the matching dictionary symbol.

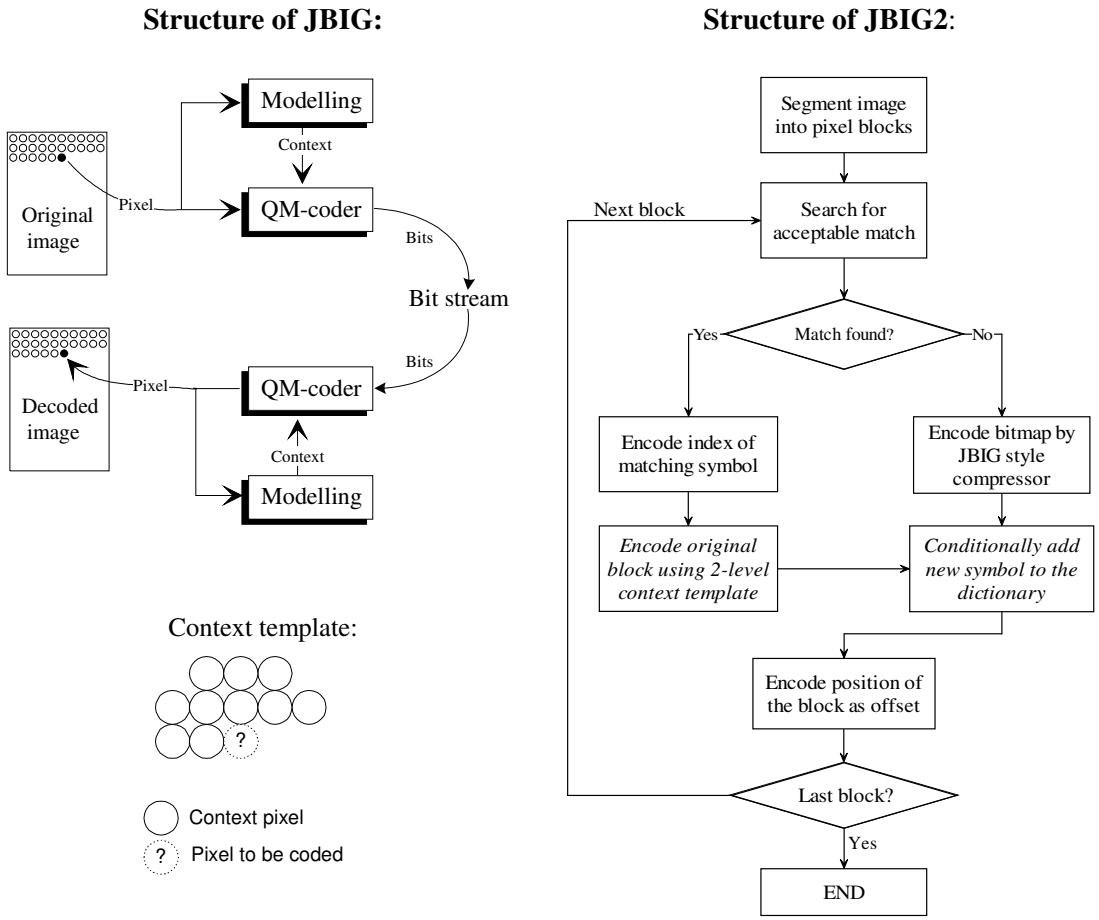

Figure 6: Structure of the JBIG and JBIG2 image compression standards.

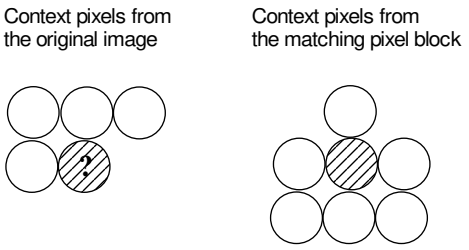

Figure 7: Two-layer context template of JBIG2 [5].

4. Alternatives ways to implement spatial access

The JBIG2 standard mainly defines the general file structure and the decoding procedure, but leaves some freedom in the design of the encoder. Here we are not interested of the pattern matching and the text coding options introduced in JBIG2. Instead, the generic coding method of JBIG2 (similar to that of JBIG) is sufficient to us, and we are more interested in the file organization itself. Specifically, we want to find

out how we can implement multi-layer image storage and spatial access using the JBIG2 standard.

For us, the two most interesting features in JBIG2 standard are the multi-page option, and the image segmentation. The first one gives us an obvious but not necessarily the best way to store the multiple layers of the map images. The segmentation, on the other hand, gives us a tool to tile the image. We divide the image into $C \times C$ pixel blocks, and store each block as its own segment.

JBIG2 defines two alternative file organizations as shown in Fig. 8. The *segment* is a basic element of a JBIG2 file. It usually contains a header, which serves for identifying segment type, segment size as also page- and cross-segment dependencies. In *sequential organization*, segment data is located right after corresponding segment header. In *random-access organization* all the segment headers are in the beginning of the file. We use the random-access mode, and define the segments so that they cover the entire image.

We have three alternative ways to organize the multi-layers images into a single file:

- *Variant 1*: All layers of a single block in one segment
- *Variant 2*: Overlapping segments
- *Variant 3*: One layer per page

In the following subsections, we study the implementation issues of these three variants. We also calculate the overhead of the file, page and segment headers. The headers take about 13 to 43 bytes each. Usually this is not a problem but in our case the situation is different because the file will consists of a large amount of segments including also the empty areas of the image. We compare the overhead of the headers relative to the compressed file size without segmentation. The overhead consists of three parts:

- Ω_{FH}: overhead of the file header,
- Ω_{P}: overhead of the page headers,
- Ω_{S}: overhead of the segment headers.

The total overhead in case of image tiling is calculated as:

$$\Omega_1 = \Omega_{FH} + \Omega_P + \Omega_S \cdot N_S$$

where N_{S} is the number of segments. In the *variant 1*, we have only a single page, and one segment per block.

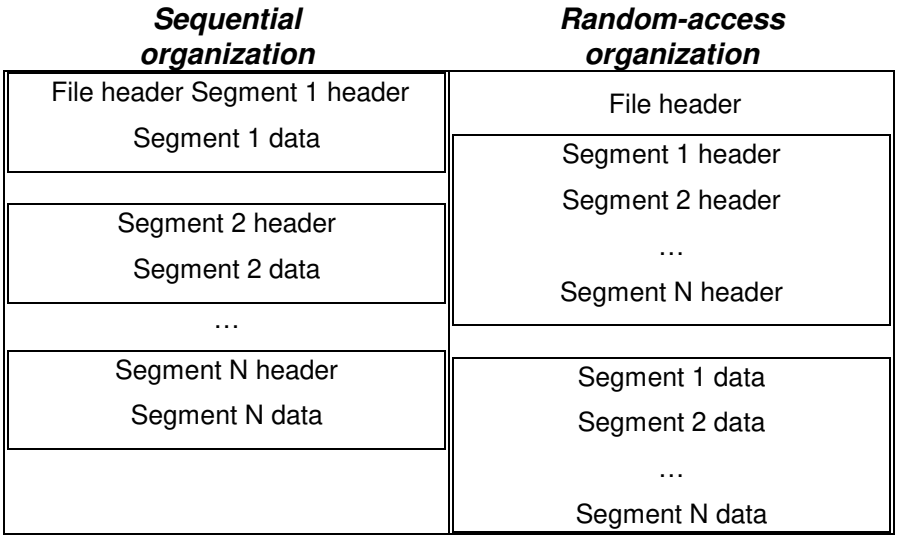

Figure 8: Two alternative file structures of JBIG2.

4.1 All layers in one segment

The *variant 1* stores all layers of a single image block into one segment, see Fig. 9. The main advantage is that the overhead due to unnecessary segment headers are avoided. The compressed data is organized so that the code from different layers follow each other subsequently. The layers are separated from each other by an extra escape sequence. Random access can be performed only to the first layer. This variant is not fully compatible with the JBIG2-standard, because a standard decoder will decode only the first layer.

The total overhead of *variant 1* is thus:

$$\Omega_1 = 13 + 43 + 31 \cdot \frac{X \cdot Y}{C^2} \tag{1}$$

where $X \cdot Y$ is the image size, and C^2 is the block size. If image segmentation is not used, the overhead would be 13+43+31 = 87 bytes.

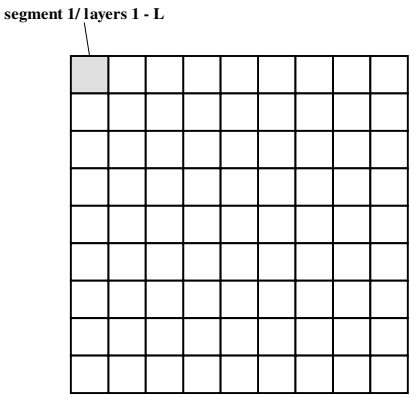

Figure 9: File organization of the variant 1.

4.2 Overlapping segments

The *variant 2* stores every layer into its own segment, see Fig. 10. Thus, there will be L segments per every block and several segments therefore can represent the same image block. This alternative is compatible with the JBIG2 standard as any standard viewer is capable of viewing all layers; although the image would most likely to be shown black-and-white. The disadvantage of this variant is that the overhead of the segment headers is L times as much as in the *variant 1*.

The total overhead of *variant 2* is:

$$\Omega_2 = 13 + 43 + L \cdot 31 \cdot \frac{X \cdot Y}{C^2} \tag{2}$$

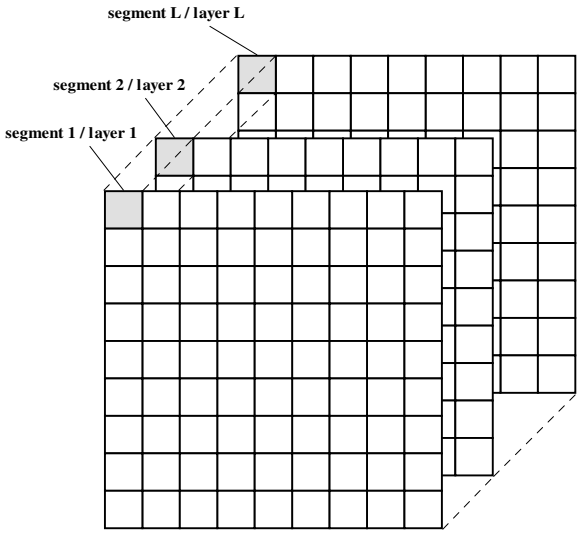

Figure 10: File organization of the variant 2.

4.3 One layer per page

In *variant 3*, each layer is stored on a separate page, see Fig. 11. The pages will be combined into a single image by the viewer. This structure is not fully compatible with the standard because a standard viewer will consider the layers as separate pages. The number of segments is the same as in the variant 2. In addition to that, the page overhead is multiplied by L, consisting of the page header and the end of page sequence.

The total overhead of variant 3 is:

$$\Omega_3 = 13 + L \cdot 43 + L \cdot 31 \cdot \frac{X \cdot Y}{C^2} \tag{3}$$

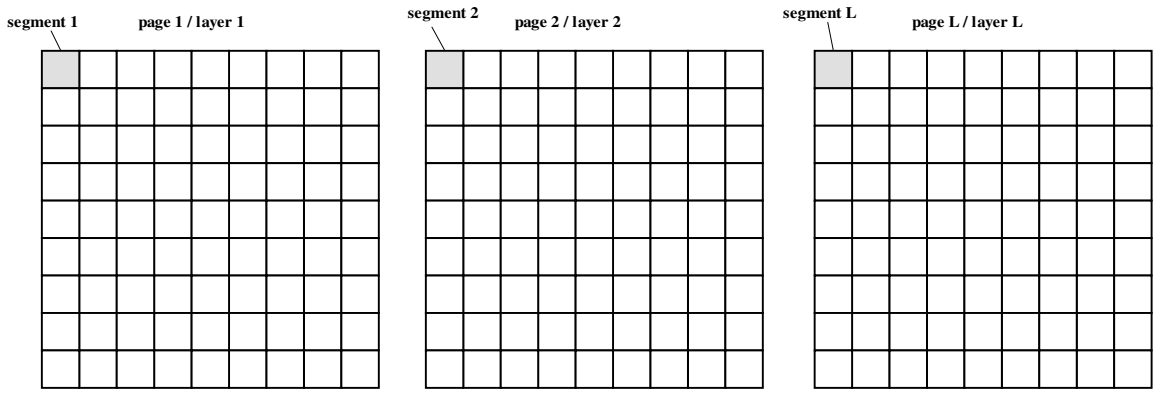

Figure 11: File organization of the variant 3.

4.4 Summary

It can be seen from equations (1)-(3) that the *variant 1* is the best choice if the file size is the only motive. There are, however, other factors that we must consider. For example, the user might want to have direct access to certain layers only. In this case, the *variant 1* would not be a feasible solution. Another factor might be the capability to interpret the images using standard JBIG2 decoder. In this case, the *variant 2* would be the only choice. These factors, however, are not likely to be important from the point of view of usability.

Much more serious problem is the way the direct access is implemented in JBIG2. It includes offsets of the data segments in the file header and, in this way, allows direct access to the compressed segments. Unfortunately the spatial location of the segments are left out from the file header. This means that we cannot leave out empty blocks, and that we must explicitly assume the order of the blocks. This can cause great inefficiency in the compression performance. Therefore, we should find more creative solutions if we are going to use JBIG2 for the purpose, or alternatively, create another format that is not compatible with the JBIG2 standard.

5. Compression results

Here we present test results for a set of test images. We use map images from the NLS *Basic Series* 1:20 000. We use five randomly chosen images 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 *LayerX*, where *Layer* is the binary layer name, and *X* is the image number from 0 to 4 in a given order. Each layer is further divided into blocks of sizes 128×128, 256×256 and 5000×5000 (no tiling). The blocks are compressed with the JBIG2 using its generic compression mode.

The header overheads are shown in Table 1 for each image separately. For *variant 1*, the overhead varies from 1% to 7% in the case of block size 256×256, and from 5% to 23% in the case of block size 128×128. On average, the overheads are 1.6% (256×256), and 5.8% (128×128). The overhead of the alternative variants is about three times that of the *variant 1*.

Table 2 shows the compressed file sizes, and the number of non-empty blocks. The number of non-empty blocks is 65% in the case of block size 128×128. This means that if we must store headers of also the empty blocks, the overhead would increase by about 35%, summing up to about 8%. This is not necessarily a problem unless the memory is crucial factor in the application.

6. Conclusions

We have studied how JBIG2 standard can be used for storing multi-layer map images. We studied different alternatives for organizing the multiple layers in the compressed file. We showed by experiments that image segmentation to blocks of 128×128 can be implemented with about 6 % overhead. The direct access, however, is not appropriately supported by JBIG2, which has two restrictions. This means that we must either define own file format similar to JBIG2, or search for more creative solutions in order to achieve direct access without excessive overhead.

Table 1: Overheads of the different variants for blocks sizes of 256×256, and 128×128. The number of bits are given relative (%) to the compressed file.

Absolute

Whole image

	Image 1	Image 2	Image 3	Image 4	Image 5	average
Ω_1	87	87	87	87	87	87
Ω_2	180	180	180	180	180	180
Ω_3	681	681	681	681	681	681

C = 256

	Image 1	Image 2	Image 3	Image 4	Image 5	average
Ω_1	12,456	12,456	12,456	12,456	12,456	12,456
Ω_2	23,926	41,565	32,947	38,651	43,022	36,022
Ω_3	24,055	41,694	33,076	38,780	43,151	36,151

C = 128

	Image 1	Image 2	Image 3	Image 4	Image 5	average
Ω_1	49,656	49,656	49,656	49,656	49,656	49,656
Ω_2	81,803	146,810	118,166	148,639	153,878	129,859
Ω_3	81,932	146,939	118,295	148,768	154,007	129,988

Relative

	Image 1	Image 2	Image 3	Image 4	Image 5	average
Ω_1	0.05%	0.01%	0.03%	0.01%	0.01%	0.01%
Ω_2	0.11%	0.02%	0.05%	0.02%	0.02%	0.02%
Ω_3	0.41%	0.08%	0.20%	0.09%	0.09%	0.09%

	Image 1	Image 2	Image 3	Image 4	Image 5	average
Ω_1	6.86%	1.27%	3.47%	1.61%	1.56%	1.61%
Ω_2	13.18%	4.24%	9.17%	4.99%	5.39%	4.66%
Ω_3	13.25%	4.25%	9.21%	5.00%	5.41%	4.67%

	Image 1	Image 2	Image 3	Image 4	Image 5	average
Ω_1	23.64%	4.64%	12.08%	5.73%	5.63%	5.78%
Ω_2	38.95%	13.73%	28.75%	17.16%	17.46%	15.11%
Ω_3	39.01%	13.74%	28.78%	17.18%	17.48%	15.12%

Summary

	Whole	C=256	C=128
Ω_1	87	12,456	49,656
Ω_2	180	36,022	129,859
Ω_3	681	36,151	129,988

	Whole	C=256	C=128
Ω_1	0.01%	1.61%	5.78%
Ω_2	0.02%	4.66%	15.11%
Ω_3	0.09%	4.67%	15.12%

Table 2: Compressed file sizes from the compressed blocks (left) and from the overhead of tiling (right) for *variant 3*.

Pure code stream sizes

C = 5000

	Basic	Field	Contour	Water	Total
Image 1	133468	4	15456	18866	167794
Image 2	532966	23031	282112	49937	888046
Image 3	194912	1678	95074	42473	334137
Image 4	321097	4039	253801	159623	738560
Image 5	356206	20006	316843	73396	766451

C = 256

	Basic	Field	Contour	Water	Total
Image 1	136238	0	17511	27783	181532
Image 2	596347	26969	301813	54959	980088
Image 3	208606	1991	105326	43216	359139
Image 4	337540	4933	270130	162445	775048
Image 5	358976	24562	335172	79030	797740

C = 128

	Basic	Field	Contour	Water	Total
Image 1	145954	0	19565	44508	210027
Image 2	633298	32393	337591	66065	1069347
Image 3	237832	2333	120367	50534	411066
Image 4	370784	5783	303168	186266	866001
Image 5	384989	28998	374085	93221	881293

Number of non-empty blocks

1	Basic	Field	Contour	Water	Total
Image 1	1	1	1	1	4
Image 2	1	1	1	1	4
Image 3	1	1	1	1	4
Image 4	1	1	1	1	4
Image 5	1	1	1	1	4

400	Basic	Field	Contour	Water	Total
Image 1	275	0	95	400	770
Image 2	400	177	381	381	1339
Image 3	400	24	391	246	1061
Image 4	400	47	400	398	1245
Image 5	400	200	400	386	1386

1600	Basic	Field	Contour	Water	Total
Image 1	824	0	240	1573	2637
Image 2	1576	502	1422	1234	4734
Image 3	1587	51	1389	783	3810
Image 4	1597	105	1573	1518	4793
Image 5	1599	468	1596	1299	4962

References

[1] Ageenko E.I. and Fränti P., "Enhanced JBIG-based compression for satisfying objectives of engineering document management system", *Optical Engineering*, **37** (5), 1530-1538, May 1998.

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

[3] Ascher R.N., Nagy G., "A means for achieving a high degree of compaction on scan-digitized printed text". *IEEE Trans. Computers*, **23** (11): 1174-1179, November 1979.

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

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

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

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

[8] Pennebaker W.B., Mitchell J.L. "Probability estimation for the Q-coder". *IBM Journal of Research, Development* **32** (6): 737-759, 1998.

[9] Pennebaker, W.B., Mitchell, J. L., *JPEG Still Image Data Compression Standard*. Van Nostrand Reinhold. 1993.

[10] Witten I.H., Moffat A. and Bell T.C., *Managing Gigabytes: Compressing and Indexing Documents and Images*. Van Nostrand Reinhold, New York, 1994.