UNIVERSITY OF JOENSUU
DEPARTMENT OF COMPUTER SCIENCE
Report Series A

Enhanced JBIG-based compression for satisfying objectives of engineering document management system

Eugene I. Ageenko and Pasi Fränti

Report A-1997-2
(revised 28.8.1997)

ACM I.4.2
UDK 681.3.06
ISSN 0789-7316
ISBN 951-708-499-4

Enhanced JBIG-based compression for satisfying objectives of engineering document management system[#]

(submitted to *Optical Engineering*)

Eugene I. Ageenko
Pasi Fränti[*]

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

Abstract: Main objectives of an engineering document management (EDM) system are outlined. Existing image compression algorithms (eg. G4 and JBIG) offer efficient solutions to the storage problem but do not sufficiently support other objectives such as spatial access and fast decoding. Here we propose a novel method based on JBIG so that the other objectives are also met. The compression performance of the proposed method is only 10 % worse than that of JBIG. At the same time, spatial access to the compressed file is achieved. The method is also 2.5 times faster in decompression than JBIG. This speed-up is comparable to the Group 4 standard (G4), but with a better compression performance. The proposed method is applicable, not only to engineering drawings, but to binary images in any document imaging system.

Subject terms: Image compression; document imaging; binary images; lossless compression.

[#] short version of the paper [1] was presented in *Picture Coding Symposium* 1997.
[*] author for correspondence

1. Introduction

Line-drawing images like *engineering drawings, cartographic maps, architectural and urban plans, schemes, circuits (radio electrical and topological)* are mainly designed by CAD systems. They are stored in digital form using vector representation such as *Computer Graphics Metafile* (CGM) or *AutoCAD drawings* (DVG). Nevertheless, there are (and will be) still a large number of drawings that are stored as paper documents.

It has been estimated by International Data Corporation (IDC) that there exist about eight billion drawings in the world [2]. Only about 13 % of them are designed and stored using CAD systems. Some paper documents may have been created prior to the CAD era, or transmitted through facsimile or other media not supporting the CAD formats.

In a typical *engineering document management system* (*EDM*) paper documents are digitized and archived in compressed digital form in order to reduce the costs of archiving, updating, and reproducing of the documents. It increases the productivity of designers because the images in EDM can be easily browsed, accessed and retrieved for viewing, printing and even further processing. A system diagram off an EDM system is shown in Fig. 1. The main objectives of the images in EDM are:

1. Small storage requirement
2. Lossless reconstruction
3. Fast decompression
4. Quick preview possibility
5. Spatial access to the image

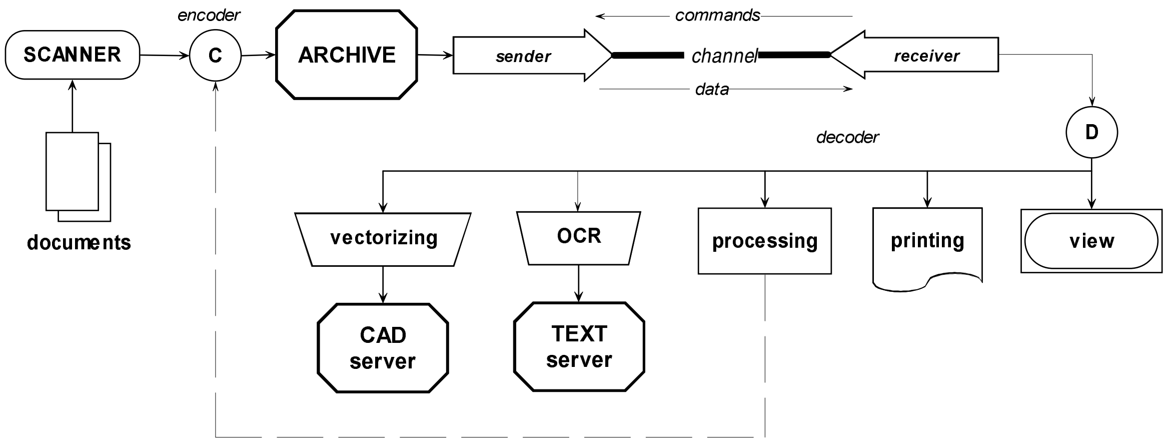

Fig. 1. Engineering document management system.

EDM images are first digitized by an optical scanner, then encoded into space efficient form and stored into the digital archive. The scanning process can be efficiently done using current, relatively inexpensive technology. The questions of a proper encoding algorithm and file format, however, are much more problematic.

A possible solution for engineering image compression is to perform a *raster-to-vector conversion*, where the bitmap image is segmented into CAD-primitives such as line segments, circles and circular arcs [3, 4, 5]. The vectorized representation is then stored with any CAD/CAM format. The storage size of an engineering drawing in CAD format takes about 2 % compared to a raster format with 300 dpi; this corresponds to a compression ratio of 50.

Raster-to-vector conversion, however, is problematic because the conversion systems have a high complexity and they are usually far from perfect. The conversion does not produce a faithful copy of the original but loss of data is apparent. Moreover, the process is often not automatic but requires human interaction, which makes the process expensive. Industrial projects have shown that the costs for such data acquisition exceed the hardware and software costs of operational information systems by a ratio of 100:1, according to [6].

Fortunately raster-to-vector conversion is not necessary but raster format with a suitable compression method is often sufficient. Using the latest compression technology [7, 8], raster images can be compressed approximately by the same amount as required by vectorized images stored in CAD format. No distortion is caused to the image (besides the digitization phase) because of lossless compression. Moreover, a methodology known as *hybrid raster-vector editing* allows easy processing of raster documents [1].

Here we propose a method to represent binary images in a compressed raster form so that the main objectives of EDM are met. The method is based on *JBIG* (*Joint Bilevel Image Experts Group*), the latest binary image compression standard [9]. The requirements 1 and 2 of EDM are met by JBIG already. To meet the other objectives also, the following modifications are proposed.

Spatial access is sufficiently supported by clustering the image into fixed size blocks. Pointers (indices) to the clusters are stored at the beginning of the image file to allow direct access to them. The content of the clusters are separately compressed using a semi-adaptive context modeling and two-stage coding process. The two stages include block level codes supporting quick preview property, and pixel-level codes for exact reconstruction of the clusters. The block level codes also enable faster decoding than the baseline JBIG [10]. The method is applicable to binary images in any document image management system, not merely in EDM.

The rest of the paper is organised as follows. EDM system and its objectives are described in Section 2. The standard JBIG compression method is recalled in Section 3. A compression system for the EDM is then outlined in Section 4. The compression algorithm and the data structure for the EDM file format are given in Section 4.1. It is followed by the discussion of the implementation details in the subsections 4.2, 4.3 and 4.4. The speed and compression efficiency of the proposed method is then studied in Section 5. Finally, conclusions are drawn in Section 6.

2. Requirements of engineering document management system

In a document imaging system like EDM, the documents are obtained and stored in electronic form. The images must be interactively browsed and efficiently retrieved for further processing, including viewing and printing on hard-copy terminals. The huge storage size of digitized images has been a major restriction in document imaging systems. Although efficient solutions already exist in the form of image compression, not enough attention has been paid for supporting the other objectives. The main EDM objectives will be outlined in this section.

Storage requirement:

The primary task in EDM is to reduce the cost of the image storage and transmission. The storage size impacts nearly every aspect of a document imaging system. Cost savings emerge from several areas: fewer storage resources are needed and less network bandwidth required. Faster transfer implies productivity gain because it makes Internet and LAN access more useful; less time is spent in waiting and fewer resources are required to retrieve the files.

The storage problem of EDM images is obvious: a raster image of size A4 scanned at relative low resolution of 200 dpi (1728×2376) takes about 0.5 Mb whereas a high quality engineering drawing of size A1 at 400 dpi (4752×6912) requires 16 Mb; and there is no upper limit. For example, typical images in *Geographic Information Systems* (GIS) take 100 Mb and even more [11, 12]. Similar huge volume imaging systems are penetrating into an increasing number of application domains including cartography, urban planning and transport management systems.

To solve the storage problem, images must be maintained in compressed (or vectorized) form. Vectorized images are suitable for editing and they can be scaled without a loss in the quality. They are greatly needed in parametric modeling and control system applications but they still represent less than 15 % of all applications where engineering documents are used. In most applications, the raster format is sufficient; especially if the hybrid editing is supported [1].

Typical hybrid editing systems support: (1) raster editing of the raster data, (2) vector editing of the vector objects, (3) semi-automatic vectorizing. The third feature is interesting. The user first picks up a raster object; the system then determines the object type, traces its shape, and replaces the raster object by the just recognized vector primitives. Once the object is pointed out, the vectorizing process is performed automatically. This feature enables the user to edit raster drawings as if they were vector images. Objects may also be scaled and rotated at any angle without distortion.

Lossless reconstruction:

The quality of the digitized image depends on the scanning resolution. Fig. 2 illustrates the dependence of the storage requirement on the image resolution. By doubling the resolution (e.g. from 200 to 400 dpi) the raw image size is multiplied by a factor of four.

The increase in the compressed image size, however, is smaller than that because higher compression ratios can be obtained for higher resolution images.

Besides the digitization process, no loss is apparent in the images. The EDM system could support semi-automatic vectorizing though, which would be performed only when requested. No resources would be wasted on converting every document into CAD format, but the conversion would be made only when so desired. Neither would there appear any loss of data without the control of the user.

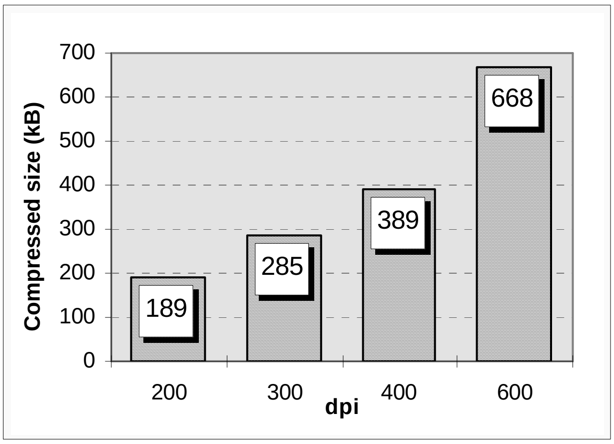

Fig. 2. The total size of the CCITT images when compressed by standard JBIG.

Fast decompression:

The purpose of the rest EDM objectives is to support real-time access to the image archive. The actual image database may not be physically present, but it might be located in different place and accessed through communication channels, which could be nothing more than a slow telephone connection. One might tolerate longer compression times if it can be done off-line, but fast decompression is always desired. The compression reduces the amount of data to be transferred, thus making the image retrieval faster. However, the decompression itself must also be fast, at least faster than the data transmission so that the system does not loose its interactivity because of decompression delays.

Quick preview:

The quick preview property allows the user to browse the archive without decompressing entire images. Preview represents a recognizable version of an image using only a small portion of the compressed image data. The quality of the preview must be high enough to reliably detect the correct image, but it must also be constructed fast enough to avoid inconvenient delays.

Spatial access:

When an image is accessed, the entire file is typically read and decompressed into memory. This is not possible if the uncompressed raster image size exceeds the available memory resources (cf. GIS images). Besides, high-speed channels are not always available. For example, most communications channels in Russia are 14.400-28.800 bps channels based on analog phone lines. 64-127 Kbit/s bridges are used only for connecting separate city networks together (mostly via satellite links). The actual transmission speed practically never exceeds 1 kilobytes per second.

The decompression of the entire image can be a major source of inefficiency. Only a small part of the image is often needed, or the image is processed and/or viewed fragment by fragment. Typical viewing devices, for example, have a smaller resolution than the original raster image and thus, only a small fragment of the entire image may be viewed at a time. When the image is scrolled, new portion of the data is retrieved and decompressed. Spatial access together with a fast "on-the-fly" decompression allow the user to operate directly on the compressed data without retrieving the entire image.

Unfortunately spatial access to compressed image file has received relatively little attention in the literature [11] (for solutions in text compression, see [13]). The current compression standards, for example, do not support spatial access at all but the entire image prior to the accessed part must be decompressed. Spatial data structures like *quadtree* allow both compact representation and spatial access to the image at the same time [12, 14, 15]. Our motivation, however, is to support spatial access directly via the compressed bit stream. This property is usually lost when an efficient representation is found for a quadtree structure. Besides, we must not forget the primary goal: to compress the image as much as possible. Quadtree does not offer competitive compression performance in comparison to JBIG.

3. JBIG compression algorithm

In JBIG the image is compressed pixel by pixel in scan raster order using *arithmetic coding* and context-based probability modeling, see Fig. 3. The combination of already coded neighboring pixels defines the context. In each context the probability distribution of the black and white pixels is adaptively determined. The pixel is then coded by arithmetic coding using the probability model of the context. Separate models are used for each context. After coding of the pixel, the statistics in the context are updated. The model thus dynamically adapts to the image statistics during the coding process.

Binary images are a favorable source for context-based compression since even a relative large number of neighboring pixels results to a reasonably small number of contexts. The larger the context, the more accurate prediction (probability model) is possible to obtain. However, with a large context the adaptation to the image statistics takes longer which increases the learning cost [16]. JBIG uses a context size of 10 pixels by default; thus, having 2^{10}=1024 different contexts in total. The context template for this, and for other context sizes of k=1..16 are shown in Fig. 4.

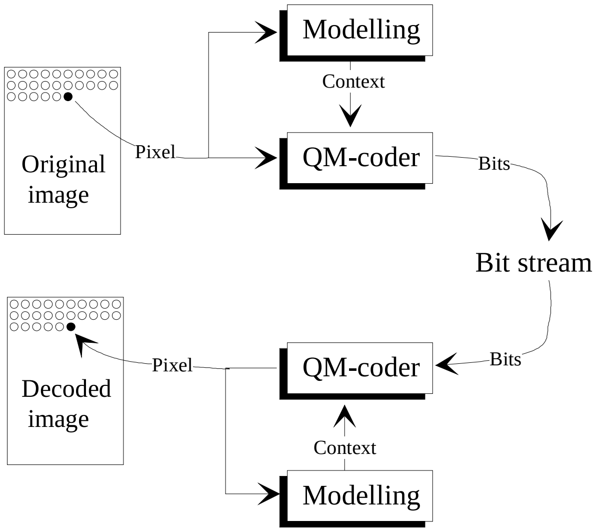

Fig. 3. Block diagram of JBIG.

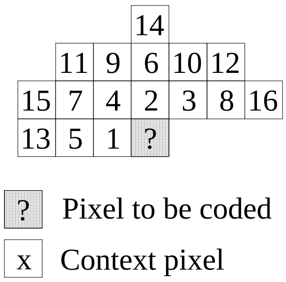

Fig. 4. Example of context templates up to 16 pixels. The pixels are included in the order given by the numbering.

Arithmetic coding is an optimal coding method with respect to the probability model. Each pixel can be compressed by $-\log_2(p)$ bits, where p is the probability estimation for the current pixel to be coded. This is denoted here as *dynamic entropy* because the probability changes from pixel to pixel. A binary arithmetic coder known as *QM-coder* is adopted in JBIG [17, 18]. It is an approximative implementation of arithmetic coding tailored for binary data. Its suboptimality is compensated by the sophisticated table-driven probability estimation, which has the property of fast adaptation and local adaptivity.

JBIG includes also a *progressive mode* where a reduced resolution version of the image is compressed first. It is followed by progressively increasing resolutions of the image so that the resolution is always doubled for the next layer. The drawback of the progressive mode is the redundancy that it adds to the code stream. The redundancy remains about 10 % according to [18].

4. Compression system for EDM

The components of the EDM files are illustrated in Fig. 5. The compression method is based on the baseline JBIG with the following modifications:

- Image is segmented into separate clusters of $C \times C$ pixels.
- Pointers (indices) to the clusters are stored.
- Separate block level codes (preview data) are included.
- Semi-adaptive context modeling is applied instead of dynamic modeling.

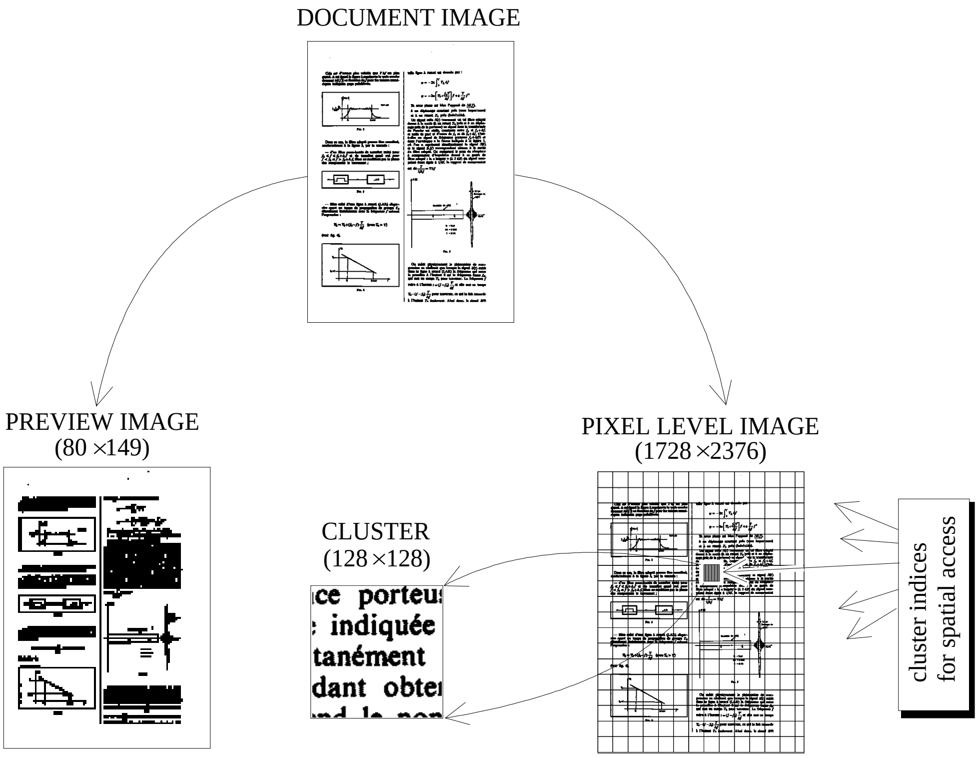

Fig. 5. Outline of the EDM compression system.

Spatial access is supported by dividing the image into fixed size clusters of $C \times C$ pixels. Each cluster is compressed separately. An *index table* is constructed from the pointers indicating where the data of each cluster is located in the compressed file. The index table is stored at the beginning of the compressed file. To restore any part of the image, only the clusters consisting of the desired pixels need to be decompressed. The cluster size is a compromise between compression efficiency and decoding delay. The smaller the cluster, the shorter is the decoding delay but the greater the overhead of the indices.

To permit the clustering, each cluster must be compressed/decompressed independently from the other clusters. There are two main restrictions in JBIG contradicting this: (1) the probability estimation uses history of all previously coded pixels, (2) the code stream produced by QM-coder is unbreakable. The first restriction is overcome using semi-adaptive probability model instead of the dynamic one. The probability models for each context are calculated globally and stored in the header of the compressed file. The same model is then applied for all clusters. The second restriction can be eliminated simply by treating the end of a cluster like the end of file situation; the data buffer is filled by dummy bits and flushed to the code stream. No end-of-code symbol is needed between the clusters because the indices identify the breaking points uniquely.

The compression algorithm must also enable a quick preview of the image. The progressive mode of JBIG could be used but we prefer a simpler block modeling scheme of [10, 19]. The image data consists of two separate levels: block level and pixel level codes. The block codes are obtained by dividing the clusters into smaller blocks of $B \times B$ pixels. Each block is classified either as an *all-white*, *all-black*, or *mixed* block. This classification is coded and stored in the compressed file.

The block coding method has several advantages. (1) It is much simpler to implement than the progressive mode of JBIG. (2) It does not increase the bit rate because the pixels in uniform (all-white and all-black) blocks can be omitted without compression. Only the pixels of the mixed blocks need to be compressed. This fact compensates the overhead due to the block codes completely. (3) The decrease in the pixel level data means also a speed-up in decompression times by a factor of 2.5, on an average.

The preview image can be constructed from the block codes using a simple resolution reduction technique known as the *Logical sum* method. Each pixel in the preview image represents a $B \times B$ block in the original image. The color of a pixel is white if the corresponding block type is all-white; otherwise it is black. This kind of preview is usually sufficient to identify the image, see Fig. 6. At the same time the overhead remains marginal. Slightly better preview quality can be obtained if the mixed blocks are represented as a color of gray. For resolution reduction methods aimed at better line preserving properties, see [20, 21].

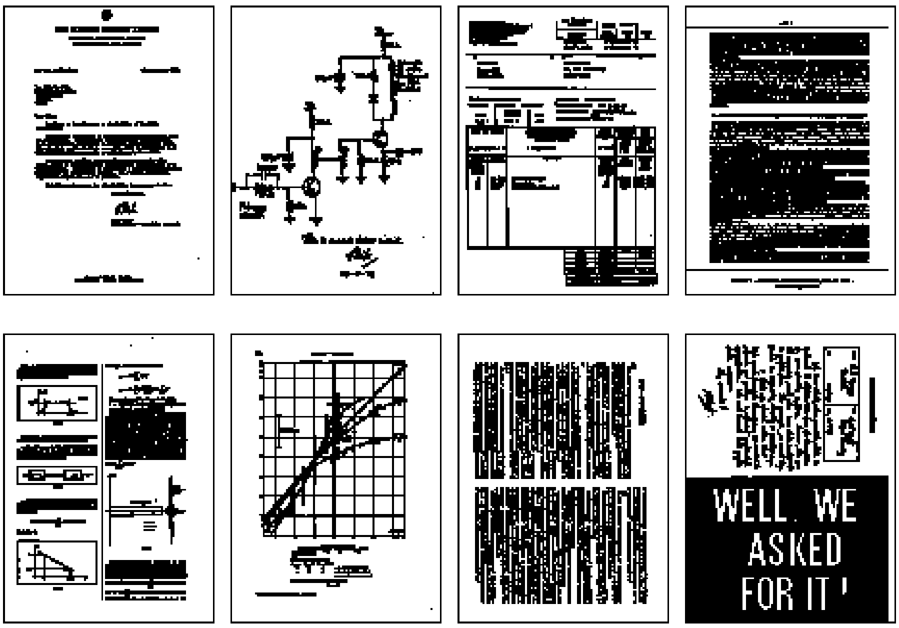

Fig. 6. Preview of the CCITT images using 16×16 block size.

4.1 Algorithm and data organisation

The file data structure is shown in Fig. 7. Unlike in [19] the codes are not mixed but the block level codes appear in the compressed file before the pixel level code. Thus, a quick preview can be constructed by reading the block level codes only. The pixel level data is stored cluster by cluster. Any cluster can then be reconstructed on basis of its block level codes, and by sequentially decompressing its pixel level code starting from the position given by the index of the cluster. The text header consists only of an identification string and image size.

The compression algorithm for the EDM is outlined in Fig. 8. In the analysis phase, the block types are determined for each cluster. The context models for the pixel level data are calculated during the same pass. The header data is then stored, block types compressed, and context models written into the compressed file. In the compression phase, the pixels of the mixed blocks are compressed by semi-adaptive JBIG for each cluster separately. The implementation details are discussed in the following subsections.

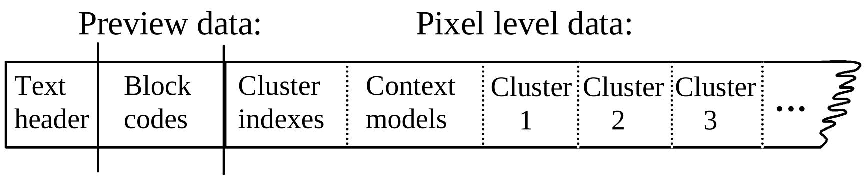

Fig. 7. Data organization of the EDM file.

1. Analyze the image (analysis phase)
    1.1. Analyze block types
    1.2. Construct the pixel level models
2. Write header and block level data
    2.1. Store text header
    2.2. Compress block codes by QM-coder
    2.3. Store dummy indices
    2.4. Store pixel level models
3. Process each cluster (compression phase)
    3.1. Compress the pixels by modified JBIG
    3.2. Record the starting positions of the next clusters
4. End up compression
    4.1. Replace the dummy indices by the real ones

Fig. 8. Main structure of the compression algorithm.

4.2 Cluster indices

The cluster indices are coded by calculating the starting point of the cluster relative to the previous cluster. This corresponds to the length of the compressed cluster data. Two bytes are used for each index. In the worst case (when no compression is achieved) this is enough for representing cluster sizes up to 724×724 (= $8 \cdot 2^{16}$ = 524.288 pixels). The space requirement of the indices is known before the compression and enough space can be allocated in the header. The actual indices, however, are not known until the entire image has been compressed and they are stored only at the last stage of the algorithm.

The overhead of the indices remains rather small. However, if very small cluster sizes are used (resulting relatively large number of clusters) compression of the cluster indices might be needed to reduce the overhead. An upper bound of $log\ N$ bits (where N is the size of the compressed file) for each index is known. The drawback of the compression would be that the space requirement of the indices cannot be known beforehand. Therefore a temporary storage space should be used for the compressed data. It could not be written into the final output file until all the data has been processed, and thus, all index values obtained. In the present method, we omit such compression schemes for simplicity.

4.3 Preview data

The preview data consists of *block codes*. It is not a binary version of the (reduced resolution) image but each block is classified either as *all-white*, *all-black*, or *mixed* block. The block classifications are coded by two binary decisions shown in Fig. 9. All-white blocks are represented by a single 0-bit, all-black blocks by a bit sequence of 10, and mixed blocks by 11. The actual coding is performed by the standard QM-coder using its dynamic probability estimation scheme. To enhance the compression performance we apply here a simple second order context model. The classification of the neighboring blocks to the left and to the above determines the context; there are $2 \cdot 3^2 = 18$ different contexts for the block codes in total. The block types of the entire image are constructed during the same pass.

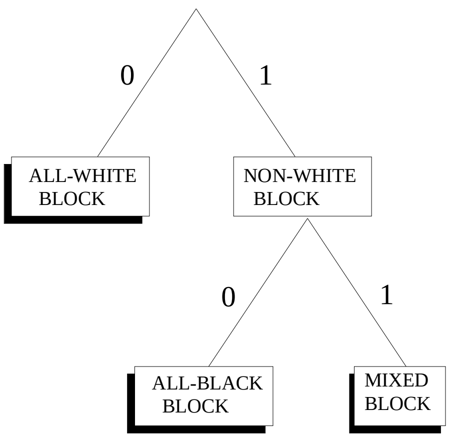

Fig. 9. Decision tree of block classification.

4.4 Pixel-level data

In principle, the clusters could be treated as independent images and apply JBIG to them separately. Dynamic modeling of JBIG has the advantages that only one pass over the data is needed and no overhead (models or code tables) need to be stored in the compressed file. At the same time the compression ratio remains virtually the same as that of a semi-adaptive model. The learning cost, however, becomes relatively high when coding smaller sizes of data. Semi-adaptive modeling is thus a better choice for our purpose.

In semi-adaptive modeling two-passes over the image are needed. In the first pass (analysis phase) the input data is analyzed and statistical models are constructed for each context. The pixel level data are then compressed in the second pass (compression phase). The same models are used for every cluster. The QM-coder is reinitialized and the models are restored each time when the compression of a new cluster starts. Note that the neighboring pixels in the context template can overlap the cluster boundaries, but the pixel values outside of the current cluster cannot be used. They are thus assumed to be white. After the cluster has been coded, the data buffer must be filled by dummy bits and flushed to the code stream. The position of the starting byte of the next cluster is also recorded for the index data.

The models are obtained in the analysis phase by calculating the frequencies of white and black pixels for each context separately. The data of the entire image is used. The resulting probabilities of each context are then mapped to the nearest state in the probability model automaton of QM-coder. Each state consists of the probability of the *least probable symbol* (LPS), and a bit indicating which color is the LPS. Each context can be stored by one byte only: 7 bits for the state and one bit for indicating the LPS symbol. The total overhead for storing the model is 2^k bytes for a k-pixel contexts (e.g. 1 kB for k=10).

In semi-adaptive modeling the compression phase is usually static; once the models have been initialized they do not change during the compression. The coding, however, is performed here using the original routines of QM-coder. The models can therefore adapt within the cluster and possibly exploit local differences in the statistics. It is not expected that this would have any major effect on the compression performance though. The only difference to the baseline JBIG is that the coder must be reinitialized in the beginning of each cluster and the models reset to the ones obtained in the analysis phase. For the details of the semi-adaptive modeling scheme, see [22].

5. Test results

The performance of the EDM system is tested next by compressing the standard CCITT test images at 200 dpi. All test runs were performed using a Pentium-90 computer (80 MIPS). The default values for the parameters were set to the following:

- Context size: 9
- Block size: 16×16
- Cluster size: 128×128

The optimal context size of EDM was found to be 9 for the CCITT images, see Fig. 10. It is independent of the selected cluster size and relatively robust to the image type. Larger contexts are impractical because the overhead of the models increases exponentially as the function of the context size. The optimal context size for JBIG is 14 for the CCITT images.

The block size is a trade-off between a compression ratio and running time, see Fig. 11. A block size of 16×16 is a safe choice in a sense that the compression ratio is hardly compromised at all. Faster decoding (and a higher quality preview) could be achieved using a block size of as low as 8×8 block at the cost of only 2 % decrease in the bit rate.

The effect of the cluster size is illustrated in Fig. 12. The performance of an EDM system using a dynamic modeling is also shown, even though the results of the semi-adaptive modeling are better. We propose the cluster size of 128×128 for images at 200-300 dpi, and 256×256 for images at 400-600 dpi. Very small cluster sizes cannot be used without compromising the compression performance too much.

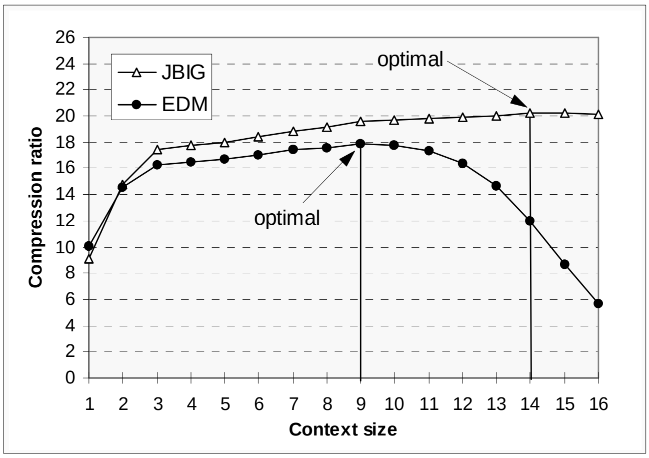

Fig. 10. Compression ratio as the function of context size.

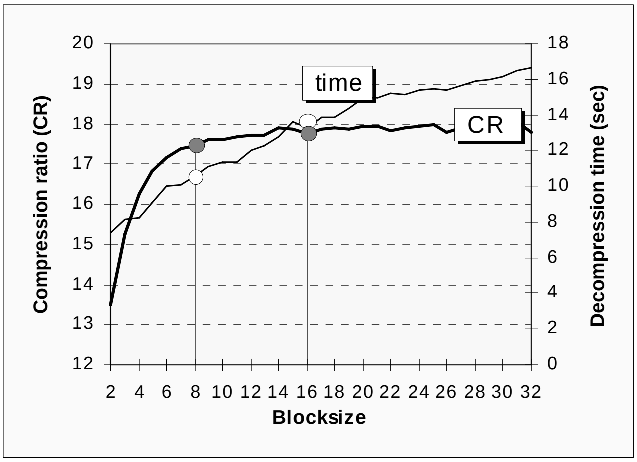

Fig. 11. Compression ratio and decompression time as the function of block size.

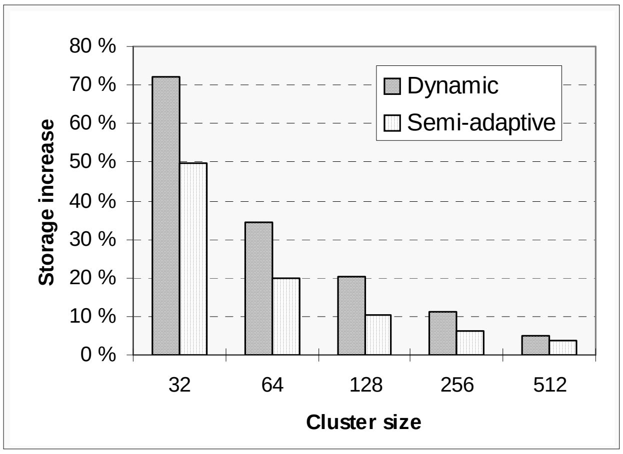

Fig. 12. Compression deficiency of EDM as the function of cluster size.
Both dynamic and semi-adaptive modeling are considered. Optimized context sizes were used: (3, 4, 7, 9, 9) for cluster size (32, 64, 128, 256, 512) in dynamic modeling. In case of semi-adaptive modeling the context size was always 9.

The compression performance of EDM with the default parameter setup is summarized in Table 1. The total size of the EDM compressed images is 10 % larger than that of JBIG, see Table 1. This is quite reasonable cost for supporting quick preview and spatial access for the compressed file. The corresponding values for the *G3* and *G4* standards are 60 % and 20 % [23, 24].

The origin of the storage size is summarized in Table 2. The index table and the context models take both a fixed amount of storage size. The EDM system thus favors larger images because the overhead of the context models is independent of the image size and type. The decrease in the compression ratio is also higher for well-compressible images. The compression ratio itself, however, is not interesting but it only matters how much is the absolute increase in the storage size. To sum up, 4 % redundancy is due to the index table and the context models. The remaining 6 % originates from the block and pixel level data.

The overhead of the block codes is compensated by the fact that the pixels in uniform blocks can be omitted without compression. The pixel level data is reduced approximately by the same amount than is the overhead of the block codes. Their net effect is thus ± 0 %. Other sources for compression deficiency are the prediction inaccuracy near the cluster boundaries, decreased local adaptivity, and the dummy bits. Their total effect is about 6 %.

Another benefit of the block codes (besides the preview property) is the speed-up in the compression/decompression because there are less pixels to be processed by the time-consuming context modeling. On an average, the decompression of an EDM image takes 41 % of the time required by JBIG, see Table 3. The method thus achieves decompression times comparable to the G4 with a compression performance of only 10 % worse than JBIG.

The compression, on the other hand, takes longer than the decompression because of the semi-adaptive modeling. Nevertheless, the compression times are still faster; only 58 % of the time taken by JBIG. Note that the EDM method is expected to be even faster for higher resolution images because the increase in the resolution evidently means also an increased number of uniform blocks; and thus less pixel level data to be coded.

Table 1: Compression performance of JBIG and EDM.

	Storage size		**Compression ratio**		**Storage**
	JBIG	**EDM**	**JBIG**	**EDM**	**increase**
CCITT 1	14708	16819	34.89	30.51	14 %
CCITT 2	8491	9834	60.44	52.19	16 %
CCITT 3	21990	24553	23.34	20.90	12 %
CCITT 4	54291	61766	9.45	8.31	14 %
CCITT 5	25823	28272	19.87	18.15	9 %
CCITT 6	12552	14084	40.89	36.44	12 %
CCITT 7	56305	58453	9.12	8.78	4 %
CCITT 8	14229	16149	36.07	31.78	13 %
Total:	208389	229930	19.70	17.86	10 %

Table 2: Source of the codebits (in bytes).

	Cluster indices	**Block level data**	**Context models**	**Pixel level data**
CCITT 1	532	297	512	15421
CCITT 2	532	372	512	8361
CCITT 3	532	510	512	22943
CCITT 4	532	514	512	60152
CCITT 5	532	515	512	26656
CCITT 6	532	494	512	12489
CCITT 7	532	801	512	56550
CCITT 8	532	770	512	14278
Total:	2 %	2 %	2 %	94 %

Table 3: Running times (sec) of JBIG and EDM.

	Compression			Decompression		
	JBIG	**EDM**	**EDM/JBIG**	**JBIG**	**EDM**	**EDM/JBIG**
CCITT 1	35	13	37 %	32	8	25 %
CCITT 2	35	12	34 %	32	5	16 %
CCITT 3	36	20	56 %	32	13	41 %
CCITT 4	35	35	100 %	33	26	79 %
CCITT 5	35	20	57 %	33	13	39 %
CCITT 6	35	15	43 %	32	8	25 %
CCITT 7	35	31	89 %	33	23	70 %
CCITT 8	35	17	49 %	32	10	31 %
Average:	35.1	20.4	58 %	32.4	13.3	41 %

6. Conclusions

Main objectives of engineering document management systems were outlined. A method for meeting these objectives was proposed by making small modifications to the standard JBIG compression method. Fast decoding, quick preview option, and spatial access to the compressed image was sufficiently supported by clustering the image, using a semi-adaptive modeling, and by the use of block coding.

The compression performance of the proposed method is only 10 % worse than that of JBIG, when cluster size of 128×128 is used, and only 5 % with the size of 256×256. At the same time, speed-up in the decompression time by a factor of 2.5 was achieved. The proposed method is applicable, not only to engineering drawings, but to any binary images.

In the future, the EDM might support a hybrid vector-raster file format. In the beginning, the images were digitized and stored in raster format only. The vectorizing would happen in the course of time when the images are processed. In this way, no resources would be wasted on unnecessary work caused by converting every document into CAD format. Neither would any uncontrollable loss of data appear.

The hybrid file format would also improve the compression ratio of JBIG by utilizing global dependencies. Additional information can be obtained when global features that are typical for engineering drawings (such as straight lines) have been extracted from the image. This would improve the prediction of the pixel level context model resulting in higher compression ratios, assuming that the improvement compensates the overhead required by storing the extracted features.

Acknowledgments

The work of Pasi Fränti was supported by a grant of the Academy of Finland, and the work of Eugene I. Ageenko by a grant of Centre for International Mobility (CIMO).

References

[1] E.I. Ageenko and P. Fränti, “Storage system for document imaging applications”, *Proc. Picture Coding Symposium*, Berlin, Germany (1997).

[2] D.J. Wilson, “How to modernize your paper engineering drawings”, *Imaging World*, 1 June, (1996).

[3] D. Dori, Y. Linag, J. Dowell and I. Chai, “Sparse-pixel recognition of primitives in engineering drawings”. *Machine Vision and Applications*, **6**, 69-82 (Spring-Summer 1993).

[4] V. Nagasamy and N.A. Langrana, “Engineering drawing processing and vectorizing system”, *Computer Vision, Graphics, and Image Processing*, **49**, 379-397 (March 1990).

[5] R. Kasturi, S.T. Bow, W. El-Masuri, J. Shah, J.R. Gattiker and U.B. Mokate, “A system for interpretation of line drawings”, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **12**(10), 978-992 (October 1990).

[6] M. Röösli and G. Monagan, “A high quality vectorizing combining local quality measures and global constraints”. *IEEE Proc. 3rd Int. Conf. on Document Analysis and Recognition*, Montreal, Canada, 243-248 (August 1995).

[7] Arps R.B., Truong T.K., “Comparison of international standards for lossless still image compression”. *Proceedings of the IEEE*, **82**(6) , 889-899 (June 1994).

[8] S.J. Urban, “Review of standards for electronic imaging for facsimile systems”, *Journal of Electronic Imaging*, **1**(1), 5-21 (January 1992).

[9] JBIG, Progressive Bi-level Image Compression, ISO/IEC International Standard 11544, ITU Recommendation T.82 (1993).

[10] P. Fränti and O. Nevalainen, “A twosStage modeling method for compressing binary images by arithmetic coding”, *The Computer Journal*, **36**(7), 615-622 (1993).

[11] R. Pajarola and P. Widmayer, “Spatial indexing into compressed raster images: how to answer range queries without decompression”. *Proc. Int. Workshop on Multimedia DBMS*, Blue Mountain Lake, NY, 94-100 (1996).

[12] H. Samet, *Applications of Spatial Data Structures: Computer Graphics, Image Processing and GIS*, Addison-Wesley , Reading, MA (1989).

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

[14] H. Samet, *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA (1990).

[15] T. Markas and J. Reif, “Quad tree tructures for image compression applications”, *Information Processing & Management*, **28**(6), 1992, 707-721.

[16] A. Moffat, “Two-level context based compression of binary images”. *IEEE Proceedings Data Compression Conference*, Snowbird, Utah, 382-391 (1991).

[17] W.B. Pennebaker, J.L. Mitchell, G.G. Langdon, R.B. Arps, “An overview of the basic principles of the Q-coder adaptive binary arithmetic coder”. *IBM Journal of Research and Development*, **32**(6), 717-726 (1988).

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

[19] P. Fränti, “A fast and efficient compression method for binary images”, *Signal Processing: Image Communication*, **6**(1), 69-76 (1994).

[20] T. Endoh, S. Kato and Y. Yasuda, “Progressive coding scheme for binary images”, *Electronic and Communications in Japan*, part 1, **74**(8), 1-17 (1991).

[21] F.C. Mintzer and J.L. Mitchell, “Line-preserving binary image reduction algorithm”. ISO/IEC JTC1/SC2/WG8, no. 601 (October 1987).

[22] P. Fränti and E.I. Ageenko, “Semiadaptive modeling in JBIG image compression”, *Manuscript*, 1997. (submitted for publication)

[23] CCITT, Standardization of Group 3 Facsimile Apparatus for Document Transmission, ITU Recommendation T.4 (1980).

[24] CCITT, Facsimile Coding Schemes and Coding Control Functions for Group 4 Facsimile Apparatus, ITU Recommendation T.6 (1984).