

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

Eugene I. Ageenko
Pasi Fränti
University of Joensuu
Department of Computer Science
P.O. Box 111
FIN-80101 Joensuu
Finland
E-mail: franti@cs.joensuu.fi

Abstract. The main objectives of an engineering document management (EDM) system are outlined. Existing image compression algorithms [e.g., Group 4 (G4) and Joint Bilevel Image Experts Group (JBIG)] offer efficient solutions to the storage problem but do not sufficiently support other objectives such as spatial access and fast decoding. We propose a novel method based on JBIG in which the other objectives are also met. The compression performance of the proposed method is only 10% worse than that of JBIG, and at the same time, spatial access to a compressed file is achieved. The method is also 2.5 times faster in decompression than JBIG. This speedup is comparable to the G4 standard, but with better compression performance. The proposed method is applicable not only to engineering drawings but to binary images in any document imaging system. © 1998 Society of Photo-Optical Instrumentation Engineers. [S0091-3286(98)02705-6]

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

Paper 06097 received Sep. 6, 1997; revised manuscript received Nov. 24, 1997; accepted for publication Nov. 25, 1997.

1 Introduction

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

It has been estimated by International Data Corporation (IDC) that about eight billion drawings exist in the world.¹ Only about 13% of these 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 that do not support the CAD formats.

In a typical engineering document management system (EDM), paper documents are digitized and archived in compressed digital form to reduce the costs of archiving, updating, and reproducing of the documents. This process 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 of 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.

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.²⁻⁴ 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 and loss of data is apparent. Moreover, the process is often not automatic but requires human interaction, which makes it 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 Ref. 5.

Fortunately raster-to-vector conversion is not always necessary and a raster format with a suitable compression method is often sufficient. Using the latest compression technology,^{6,7} raster images can be compressed by approximately 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.

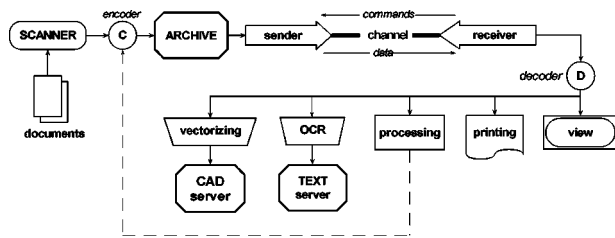


Fig. 1 EDM system.

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 Joint Bilevel Image Experts Group (JBIG), the latest binary image compression standard.⁸ Requirements 1 and 2 of EDM are already met by JBIG. 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 enable direct access to them. The contents of the clusters are separately compressed using a semiadaptive context modeling and a two-stage coding process. The two stages include block-level codes supporting the 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 (Ref. 9). The method is applicable to binary images in any document image management system, not merely in EDM.

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

2 EDM System Requirements

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, insufficient attention has been paid to supporting the other objectives. The main EDM objectives are outlined in this section.

2.1 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 is required. Faster transfer implies productivity gain because it makes Internet

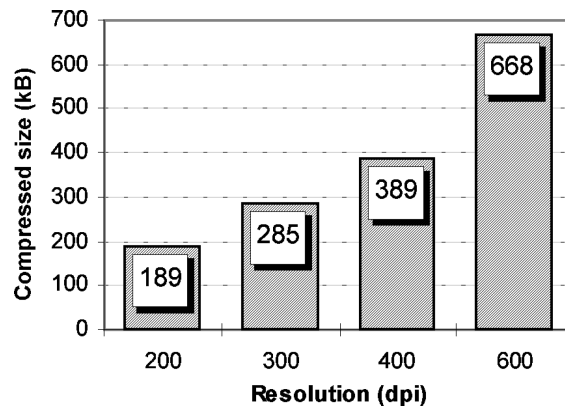


Fig. 2 Total size of the CCITT images when compressed by standard JBIG.

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 relatively low resolution of 200 dpi (1728×2376) takes about 0.5 Mbits, whereas a high-quality engineering drawing of size A1 at 400 dpi (4752×6912) requires 16 Mbits; and there is no upper limit. For example, typical images in Geographic Information Systems (GIS) take 100 Mbits and even more.^{10,11} 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 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, and (3) semiautomatic 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 can also be scaled and rotated at any angle without distortion.

2.2 Lossless Reconstruction

The quality of the digitized image depends on the scanning resolution. Figure 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 4. 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 semiautomatic vectorizing though, which would be performed only when requested. No resources would be wasted on converting every document into CAD format, because the conversion would be made only when so desired. Neither would there be any loss of data without the control of the user.

2.3 Fast Decompression

The purpose of the rest of the 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. The decompression itself must also be fast, however, at least faster than the data transmission so that the system does not lose its interactivity because of decompression delays.

2.4 Quick Preview

The quick preview property enables 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 quickly enough to avoid inconvenient delays.

2.5 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 (e.g., GIS images). Besides, high-speed channels are not always available. For example, most communications channels in Russia are 14,400 to 28,800 bits/s channels based on analog phone lines. The 64 to 128 kilobits/s bridges are used only for connecting separate city networks together (mostly via satellite links). The actual transmission speed practically never exceeds 1 kilobytes/s.

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 can be viewed at a time. When the image is scrolled, a new portion of the data is retrieved and decompressed. Spatial access together with a fast "on-the-fly" decompression enables 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 literature¹⁰ (for solutions in text compression, see Ref. 12). The current compression standards, for example, do not support spatial access, and the entire image prior to the accessed part must be decompressed. Spatial data structures such as quadtree enable both compact representation and spatial access to the image at the same time.^{11,13,14} Our motivation, however,

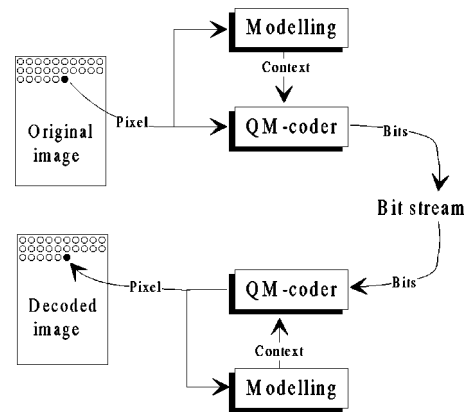


Fig. 3 Block diagram of JBIG.

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 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 relatively large number of neighboring pixels results in a reasonably small number of contexts. The larger the context, the more accurate is prediction (probability model) that can be obtained. However, with a large context the adaptation to the image statistics takes longer, which increases the learning cost.¹⁵ 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 $k = 1$ to 16 are shown in Fig. 4.

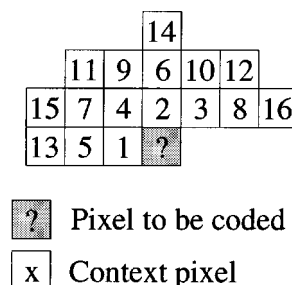


Fig. 4 Example of context templates up to 16 pixels. The pixels are included in the order given by the numbering.

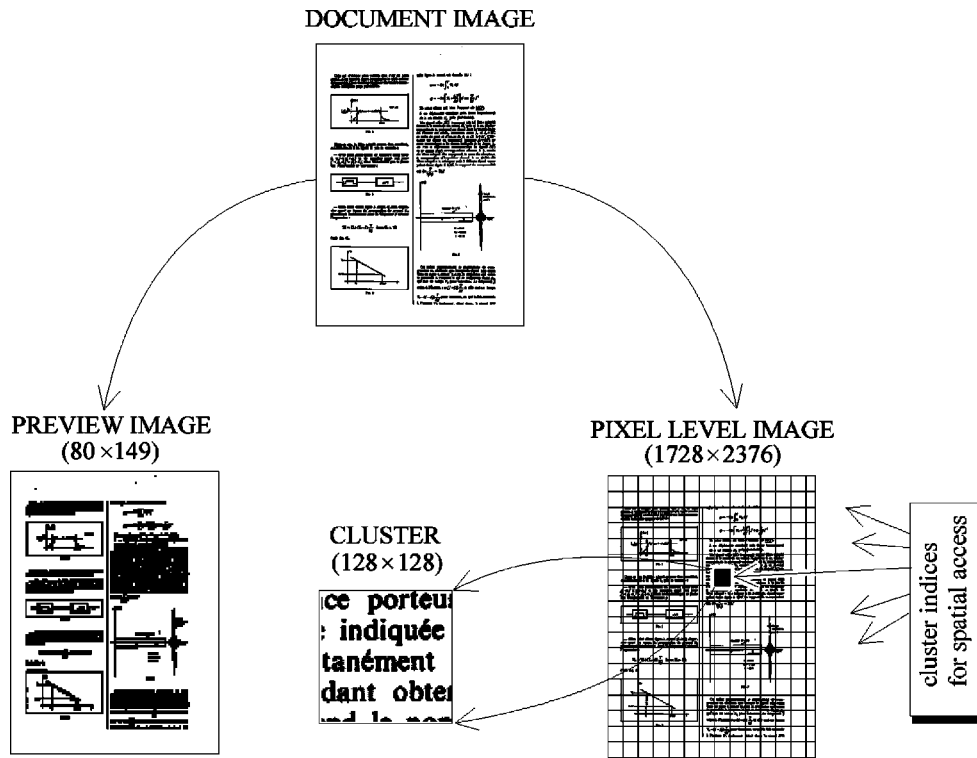


Fig. 5 Outline of the EDM compression system.

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. Here this is termed dynamic entropy because the probability changes from pixel to pixel. A binary arithmetic coder known as the QM-coder is adopted^{16,17} in JBIG. The QM-coder is an approximative implementation of arithmetic coding tailored for binary data, and its suboptimality is compensated by the sophisticated table-driven probability estimation, which has the properties of fast adaptation and local adaptivity.

JBIG also includes a progressive mode, where a reduced resolution version of the image is compressed first, 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 Ref. 17.

4 EDM Compression System

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

1. An image is segmented into separate clusters of $C \times C$ pixels.
2. Pointers (indices) to the clusters are stored.
3. Separate block-level codes (preview data) are included.
4. Semiadaptive context modeling is applied instead of dynamic modeling.

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 the decoding delay but the greater the overhead of the indices.

To enable 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 the history of all previously coded pixels and (2) the code stream produced by QM-coder is unbreakable. The first restriction is overcome using a semiadaptive 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 the simpler block modeling scheme of Refs. 9 and 18. The image data consists of two separate levels: block-level and pixel-level codes. The block codes

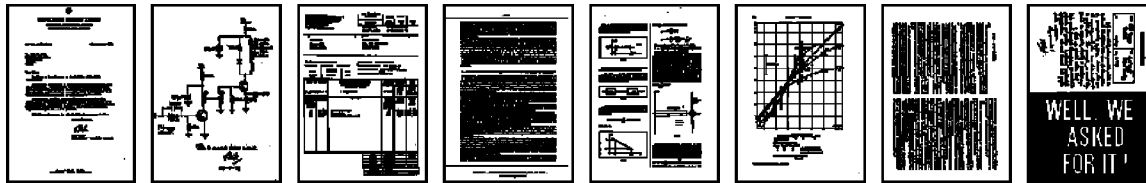


Fig. 6 Preview of the CCITT images using 16x16 block size.

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 must be compressed. This fact compensates the overhead due to the block codes completely.
3. The decrease in the pixel-level data also results in a speedup in decompression times by a factor of 2.5, on 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 Refs. 19 and 20.

4.1 Algorithm and Data Organization

The file data structure is shown in Fig. 7. Unlike in Ref. 18, the codes are not mixed, and 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 based on 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 of only 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

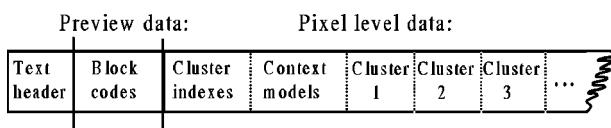


Fig. 7 Data organization of the EDM file.

stored, block types are compressed, and context models are 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.

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 ($= 8 \times 2^{16} = 524,288$ 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. If very small cluster sizes are used (resulting relatively large number of clusters), however, 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 presented method, we omit such compression schemes for simplicity.

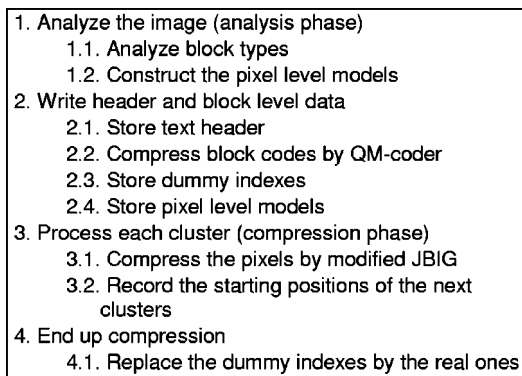


Fig. 8 Main structure of the compression algorithm.

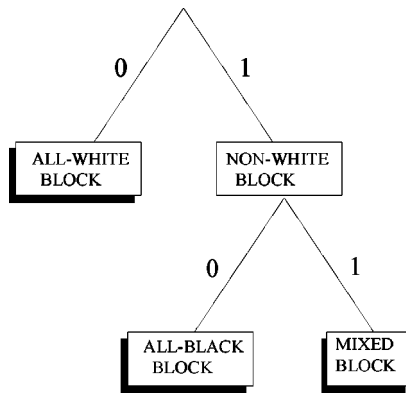


Fig. 9 Decision tree of block classification.

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 an all-white, an all-black, or a mixed block. The block classifications are coded by the 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 a simple second-order context model. The classification of the neighboring blocks to the left and above determines the context; there are $2 \times 3^2 = 18$ different contexts for the block codes in total. The block types of the entire image are constructed during the same pass.

4.4 Pixel-Level Data

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

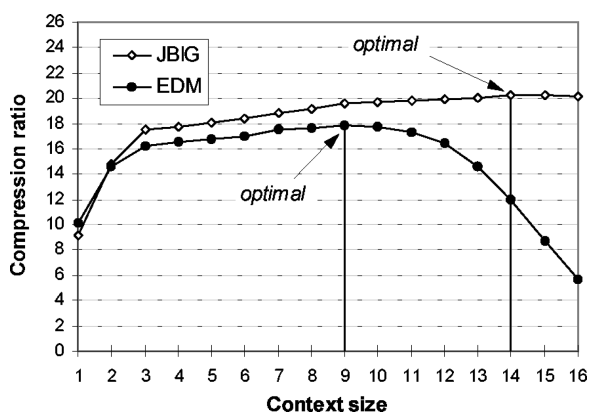


Fig. 10 Compression ratio as the function of context size.

In semiadaptive 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 the 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 1 bit to indicate the LPS symbol. The total overhead for storing the model is 2^k bytes for a k -pixel context (e.g., 1 kbyte for $k = 10$).

In semiadaptive 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 using the original routines of the QM-coder. The models can therefore adapt within the cluster and possibly exploit local differences in the statistics. It is not expected, however, that this would have any major effect on the compression performance. The only difference from the baseline JBIG is that the coder must be reinitialized at the beginning of each cluster and the models reset to the ones obtained in the analysis phase. For the details of the semiadaptive modeling scheme, see Ref. 21.

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), and 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 a 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 the sense that the compression ratio is hardly compromised at all. Faster decoding (and a higher

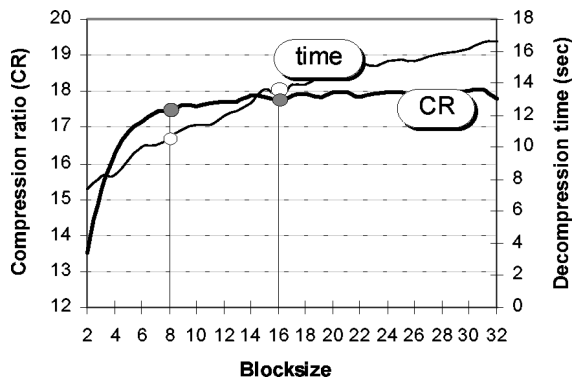


Fig. 11 Compression ratio and decompression time as the function of block size.

quality preview) could be achieved using a block size of as low as 8×8 , at the cost of only a 2% increase in the bit rate.

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

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 a quite reasonable cost for supporting quick preview and spatial access for the compressed file. Moreover, EDM outperforms the G3 and G4 standards,^{22,23} which are about 60 and 20% worse than JBIG. These facsimile standards are based on run-length encoding and 2-D READ-code.²⁴

The origin of the storage size for EDM is summarized in Table 2. The index table and the context models both take 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-

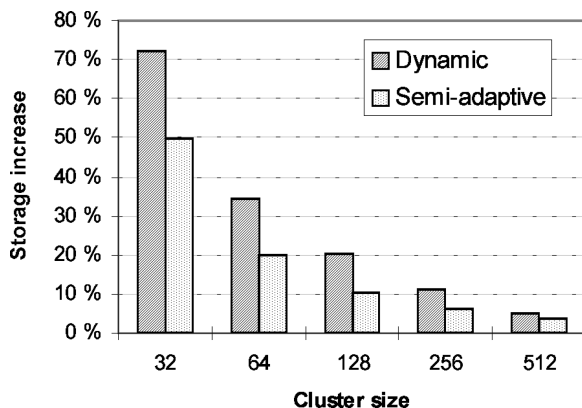


Fig. 12 Compression deficiency of EDM as the function of cluster size. Both dynamic and semiadaptive modeling are considered. Optimized context sizes 3, 4, 7, 9, and 9 were used for cluster sizes 32, 64, 128, 256, and 512 in dynamic modeling. In case of semiadaptive modeling the context size was always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

Table 1 Compression performance of JBIG and EDM.

	Storage Size		Compression Ratio		Storage Increase (%)
	JBIG	EDM	JBIG	EDM	
CCITT 1	14,708	16,819	34.89	30.51	14
CCITT 2	8,491	9,834	60.44	52.19	16
CCITT 3	21,990	24,553	23.34	20.90	12
CCITT 4	54,291	61,766	9.45	8.31	14
CCITT 5	25,823	28,272	19.87	18.15	9
CCITT 6	12,552	14,084	40.89	36.44	12
CCITT 7	56,305	58,453	9.12	8.78	4
CCITT 8	14,229	16,149	36.07	31.78	13
Total	208,389	229,930	19.70	17.86	10

compressible images. The compression ratio itself, however, is not interesting but the only critical matter is the absolute increase in the storage size. In summary, 4% redundancy is due to the index table and the context models. The remaining 6% redundancy 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 as is the overhead of the block codes. The net effect is thus $\pm 0\%$. Other sources of 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 speedup 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 an JBIG image (see Table 3). The method thus achieves decompression times comparable to the G4 standard.

The compression, on the other hand, takes longer than the decompression because of the semiadaptive 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 also results

Table 2 Source of the code bits (in bytes).

	Cluster Indexes	Block-Level Data	Context Models	Pixel-Level Data
CCITT 1	532	297	512	15,421
CCITT 2	532	372	512	8,361
CCITT 3	532	510	512	22,943
CCITT 4	532	514	512	60,152
CCITT 5	532	515	512	26,656
CCITT 6	532	494	512	12,489
CCITT 7	532	801	512	56,550
CCITT 8	532	770	512	14,278
Total	2%	2%	2%	94%

Table 3 Running times in seconds 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

in an increased number of uniform blocks, and thus less pixel-level data to be coded.

6 Conclusions

The main objectives of EDM 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 semiadaptive modeling, and by the use of block coding.

The compression performance of the proposed method is only 10% worse than that of JBIG when a cluster size of 128×128 is used, and only 5% worse with a size of 256×256 . At the same time, a speedup 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 occur over 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 occur.

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 of 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 from the Academy of Finland, and the work of Eugene I. Ageenko by a grant from the Centre for International Mobility.

References

1. D. J. Wilson, "How to modernize your paper engineering drawings," *Imaging World* 5(6), 52–53 (June 1996).
2. D. Dori, Y. Linag, J. Dowell, and I. Chai, "Sparse-pixel recognition

- of primitives in engineering drawings," *Mach. Vis. Appl.* 6, 69–82 (1993).
3. V. Nagasamy and N. A. Langrana, "Engineering drawing processing and vectorizing system," *Comput. Vis. Graph. Image Process.* 49, 379–397 (Mar. 1990).
4. 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. Pattern Anal. Mach. Intell.* 12(10), 978–992 (1990).
5. M. Rösli and G. Monagan, "A high quality vectorizing combining local quality measures and global constraints," in *IEEE Proc. 3rd Int. Conf. on Document Analysis and Recognition*, pp. 243–248, Montreal (Aug. 1995).
6. R. B. Arps and T. K. Truong, "Comparison of international standards for lossless still image compression," *Proc. IEEE* 82(6), 889–899 (1994).
7. S. J. Urban, "Review of standards for electronic imaging for facsimile systems," *J. Electron. Imaging* 1(1), 5–21 (1992).
8. "JBIG, Progressive Bi-level Image Compression," ISO/IEC International Standard 11544, ITU Recommendation T.82 (1993).
9. P. Fränti and O. Nevalainen, "A two-stage modeling method for compressing binary images by arithmetic coding," *Comput. J.* 36(7), 615–622 (1993).
10. R. Pajarola and P. Widmayer, "Spatial indexing into compressed raster images: how to answer range queries without decompression," in *Proc. Int. Workshop on Multimedia DBMS*, pp. 94–100, New York State Center for Advanced Technology in Computer Applications and Software Engineering (CASE) at Syracuse University, Blue Mountain Lake, NY (1996).
11. H. Samet, *Applications of Spatial Data Structures: Computer Graphics, Image Processing and GIS*, Addison-Wesley, Reading, MA (1989).
12. I. H. Witten, A. Moffat, and T. C. Bell, *Managing Gigabytes: Compressing and Indexing Documents and Images*, Van Nostrand Reinhold, New York (1994).
13. H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, MA (1990).
14. T. Markas and J. Reif, "Quad tree structures for image compression applications," *Inf. Process. Manag.* 28(6), 707–721 (1992).
15. A. Moffat, "Two-level context based compression of binary images," in *IEEE Proc. Data Compression Conf.*, pp. 382–391, Snowbird, UT (1991).
16. W. B. Pennebaker, J. L. Mitchell, G. G. Langdon, and R. B. Arps, "An overview of the basic principles of the Q-coder adaptive binary arithmetic coder," *IBM J. Res. Develop.* 32(6), 717–726 (1988).
17. W. B. Pennebaker and J. L. Mitchell, *JPEG Still Image Data Compression Standard*, Van Nostrand Reinhold, New York (1993).
18. P. Fränti, "A fast and efficient compression method for binary images," *Signal Process. Image Commun.* 6(1), 69–76 (1994).
19. T. Endoh, S. Kato, and Y. Yasuda, "Progressive coding scheme for binary images," *Electron. Commun. Jpn. Part 1* 74(8), 1–17 (1991).
20. F. C. Mintzer and J. L. Mitchell, "Line-preserving binary image reduction algorithm," ISO/IEC JTC1/SC2/WG8, no. 601 (Oct. 1987).
21. E. I. Ageenko and P. Fränti, "Forward-adaptive variant of the baseline JBIG for applications requiring spatial access," Technical Report A-1998-2, Univ. of Joensuu, Finland (1998).
22. "CCITT, Standardization of Group 3 Facsimile Apparatus for Document Transmission," ITU Recommendation T.4 (1980).
23. "CCITT, Facsimile Coding Schemes and Coding Control Functions for Group 4 Facsimile Apparatus," ITU Recommendation T.6 (1984).
24. R. B. Arps and T. K. Truong, "Comparison of international standards for lossless still image compression," *Proc. IEEE* 82, 889–899 (1994).



Eugene I. Ageenko received his MSc in applied mathematics from the University of Moscow, Uljanovsk, Russia, in 1995. Currently, he is a doctoral student in the Department of Computer Science, University of Joensuu, Finland, where he is involved in research on document image compression.



Pasi Fränti received MSc and PhD degrees in computer science from the University of Turku, Finland, in 1991 and 1994, respectively. From 1992 to 1995 he was with the University of Turku. Currently he is with the University of Joensuu as a researcher funded by the Academy of Finland. His primary research interests are in image compression, vector quantization, and clustering methods.