Pattern Analysis & Applications (1999)2:285–291

Vectorising and Feature-Based Filtering for Line-Drawing Image Compression

P. Fränti[1], E. I. Ageenko[1] and A. Kolesnikov[2]

[1]*Department of Computer Science, University of Joensuu, Joensuu, Finland;* [2]*Institute of Automation and Electrometry, Russian Academy of Sciences, Novosibirsk, Russia*

Abstract: A three-stage method for compressing bi-level line-drawing images is proposed. In the first stage, the raster image is vectorised using a combination skeletonising and line tracing algorithm. A feature image is then reconstructed from the vector elements extracted. In the second stage, the original image is processed by a feature-based filter for removing noise in the objects out-line. This improves image quality and compression performance. In the final stage, the filtered raster image is compressed using a standard compression technique, JBIG. For a set of test images, the method achieves a compression ratio of 40:1, in comparison to 33:1 of JBIG.

Keywords: Filtering; JBIG; Line-drawing images; Near-lossless compression; Preprocessing; Vectorising

1. INTRODUCTION

Lossless compression of bi-level images has been well studied in the literature, and several standards already exist [1]. In *JBIG* the image is coded pixel-by-pixel using a *context-based probability model* and *arithmetic coding* [2]. The combination of already coded neighbouring pixels defines the context. In each context, the probability distribution of the black and white pixels is adaptively determined, and the pixel is then coded by binary arithmetic coder, namely the *QM-coder* [3].

JBIG achieves compression ratios from 10 to 50 for a typical A4-sized image. The pixelwise dependencies are well used, and there is not much room for improvement. Remarkable improvement has been achieved only by specialising to some known image types and exploiting global dependencies. For example, the methods in Witten et al [4] and Howard [5] include pattern matching techniques to extract symbols from text images. The compressed file consists of bitmaps of the library symbols coded by a JBIG-style compressor, the location of the extracted marks as offsets, and a pixelwise coding of the matched symbols using a two-layer context template.

One way to improve compression is to preprocess the image by filtering for noise removal. Filtering reduces irregularities in the image caused by noise, and in this way makes the image more compressible without affecting the image quality. Noise appears in the image as randomly scattered noise pixels (additive noise), and as content-dependent noise distorting the contours of printed objects (lines, characters) by making them ragged. The noise level may be low enough not to significantly detract from the subjective quality, but it introduces unnecessary details that decrease the compression performance.

Several methods have been considered for image preprocessing by analysing the local pixel neighbourhood defined by a filtering template [6–8]. These filters (logical smoothing, variations of median filtering, isolated pixel removal and morphological filters [9]) use a set of rules to accept or reject the pixel, such as predefined masks or a quantitative description of the local neighbouring area. Recent research in mathematical morphology has shown that morphological filtering can be used as an efficient tool for pattern restoration in an environment with a lot of additive noise [10–12]. Such approaches, however, are not necessarily suitable for filtering content-dependent quantisation noise. Another problem is that the filtering may destroy fine image structures carrying crucial information if the amount of filtering is not controlled.

We study content-based noise removal for *line-drawing images* such as engineering drawings, cartographic maps, architectural and urban plans and circuits (radio electrical and topological) by using global spatial dependencies. This

Received: 25 November 1998
Received in revised form: 1 April 1999
Accepted: 12 April 1999

kind of image consists mainly of straight-line elements. Global information is gathered from the image by extracting *line features*, which are used in the filtering. The quality of the filtering is controlled by allowing only isolated groups of noise pixels to be changed. Objects that are not recognised by the feature extraction process are left untouched. The filtering is applied as part of an image compression system; the compression remains near-lossless because any uncontrolled loss in the image quality cannot appear.

We propose a three-stage compression method as outlined in Fig. 1. In the first stage (*vectorising*), vector elements are extracted from the image using raster-to-vector conversion. An equal size feature image is created from the extracted line segments to approximate the input image. In the second stage (*filtering*), the original raster image is preprocessed by a feature-based filtering for improving image quality. The feature image is used as a semantic model of the image consisting of non-local information about the image that cannot be used by using local spatial filters. In the third stage (*compression*), the filtered image is compressed by JBIG. The feature file is used only in the compression phase, and it therefore does not need to be stored in the compressed file.

Feature extraction and filtering are considered as preprocessing steps, and they are invisible in the decompression phase. The method uses a standard image compression component, JBIG. The resulting output files are therefore standard JBIG files, and the decompression is exactly the same as in JBIG. The method could also be used as a separate plug-in for various existing compression techniques, such as ITU Group 3 and Group 4 [13,14], and can thus be easily integrated into any existing compression method.

The vectorising is an essential part of the new method, but the method itself is independent of the chosen vectorising component. The quality of the vectorising affects only the amount of compression improvement that can be achieved; the quality of the output image is controlled by the filtering method. The vectorising and filtering parts can be implemented as optional components, and used only on-demand. Details of the three stages are discussed in the following sections.

2. FEATURE EXTRACTION USING VECTORISING

The vectorising process is outlined in Fig. 2. We apply the method described by Kolesnikov et al [15]. The motivation is to extract semantic information from the image in the form of rigid line segments. Each line segment is represented as its two end-points, and as the width of the line. The vector features are not stored in the compressed files, but the vectorising is considered as an intelligent preprocessing stage. A feature image is reconstructed from the vector

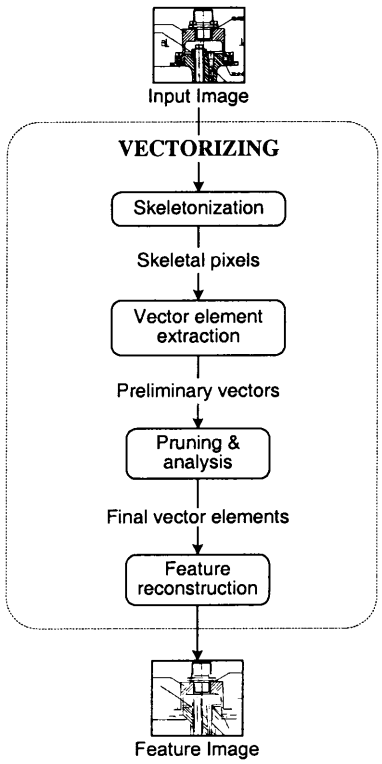

Fig. 2. Block diagram of vectorising.

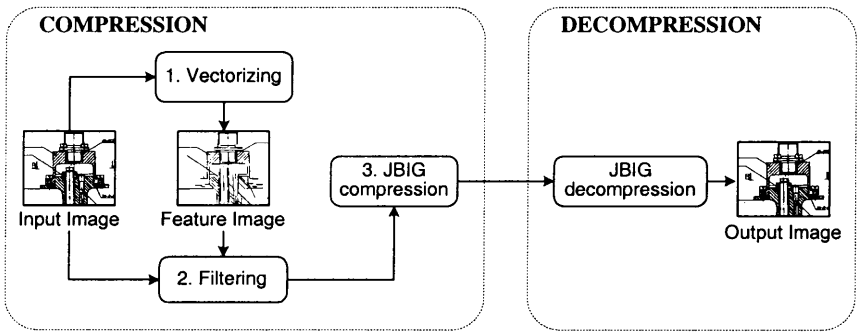

Fig. 1. Block diagram of the three-stage compression method.

representation and used in the filtering. Details of the vectorising process are described in the following subsections.

2.1. Skeletonising

The black-and-white raster image is first processed by a Distance Transform (DT) defined by 4-connectivity. We use the fast and memory efficient implementation of Kolesnikov and Trichina [17], which processes the image in smaller fragments. This eliminates the need for two passes over the image. The resulting distance labelled image is then thinned using the one-pass algorithm of Arcelli and di Baja [16]. Skeletal pixels are recognised by checking the 3×3 neighbourhood of each pixel. The pixels satisfying one of the so-called 'multiplicity conditions' are marked as skeletal pixels. The result of the algorithm is a width-labelled skeletal image.

2.2. Extraction of Vector Elements

The vector elements are extracted from the skeletal image using a fast and simple line-tracing algorithm. The branches of the skeleton are traced pixel-by-pixel from one delimiter (line end or crossroad) to another, and stored as chain codes. The direction for tracing is derived from a precalculated, two-dimensional Look-Up Table (LUT). The first index for accessing the LUT is the previous direction, and the second index is constructed from the 3×3 neighbouring pixel values of the current pixel. The resulting chain code is then processed to produce a piecewise-linear approximation of the branch with zero error [18]. The width of each line segment is calculated as the average width label of the skeletal pixels in the segment. The extracted segments of the same branch are stored as a chain of vector elements.

2.3. Pruning and Analysis

The extracted vector chains are further analysed for constructing larger elements. There are four classes of vector chains, each described by the two end-points and the width of the line:

- Single point: (x_1,y_1,w_1).
- Single vector: (x_1,y_1,w_1), (x_2,y_2,w_2).
- Chain of n vectors: $\{(x_k,y_k,w_k) \mid k = 1,\ldots,n+1\}$.
- Ring of n vectors: $\{(x_k,y_k,w_k) \mid k = 1,\ldots,n+1\}$ where $x_1 = x_n$ and $y_1 = y_n$.

Vector elements are combined (pruned) from primitives having a common end-point and the same orientation. Small gaps between the lines are filled, and false branches are removed. The remaining vector chains are then classified as either 'good' (linear) or 'bad' (noise and non-linear). The good chains are stored by their coordinate differentials using a variable-length code.

3. FEATURE-BASED FILTERING

In the second stage, the original image is processed by a feature-based filter for removing noise near the borders of the extracted line elements. This improves the image quality and results in a more compressible raster image. The filtering is based on a noise removal procedure as shown in Fig. 3. A mismatched image is constructed from the differences

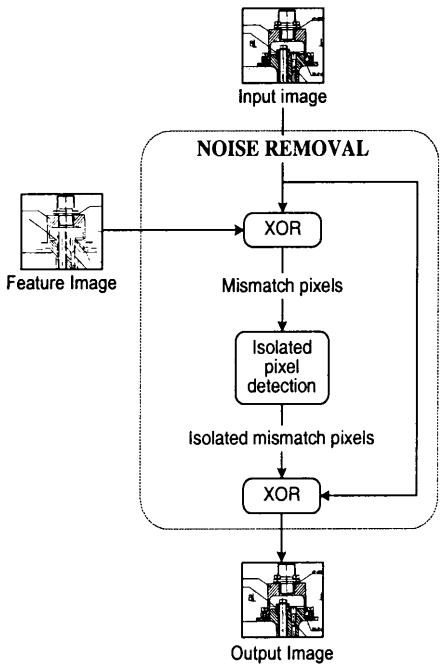

Fig. 3. Block diagram of the noise removal procedure.

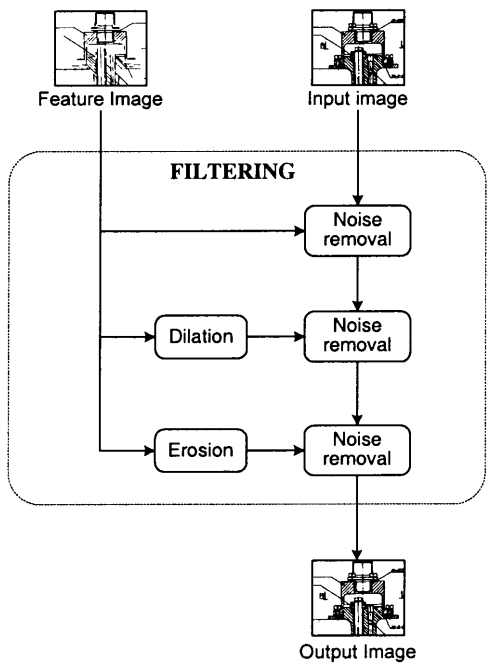

Fig. 4. Block diagram of the three-stage filtering procedure.

between the original and the feature image. Isolated mismatched pixels (and pixel groups of up to two pixels) are detected, and the corresponding pixel values in the original image are changed. This removes additive noise and smoothes the edges along the detected line segments.

The noise removal procedure is successful if the feature image is accurate. The vectorising method, however, does not always provide the exact width of the lines. The noise removal procedure is therefore iterated three times, as shown in Fig. 4. In the first stage, the feature image is applied; in the second stage the feature image is *dilated*; and in the third stage it is *eroded* before being input into the noise removal procedure. This compensates for most of the inaccuracies in the width detection. See elsewhere [9,11] for details of the morphological dilation and erosion.

The stepwise process is demonstrated in Fig. 5 for a small image sample. Most of the noise is detected and removed in the first phase. However, in some cases there are too many mismatched pixels grouped together because of an incorrect estimation of the line width, and therefore no pixels can be filtered. Even if these inaccuracies have a visually unpleasant appearance in the feature image, they do not necessarily prevent effective filtering. For example, the middle diagonal line in the feature image is too wide in

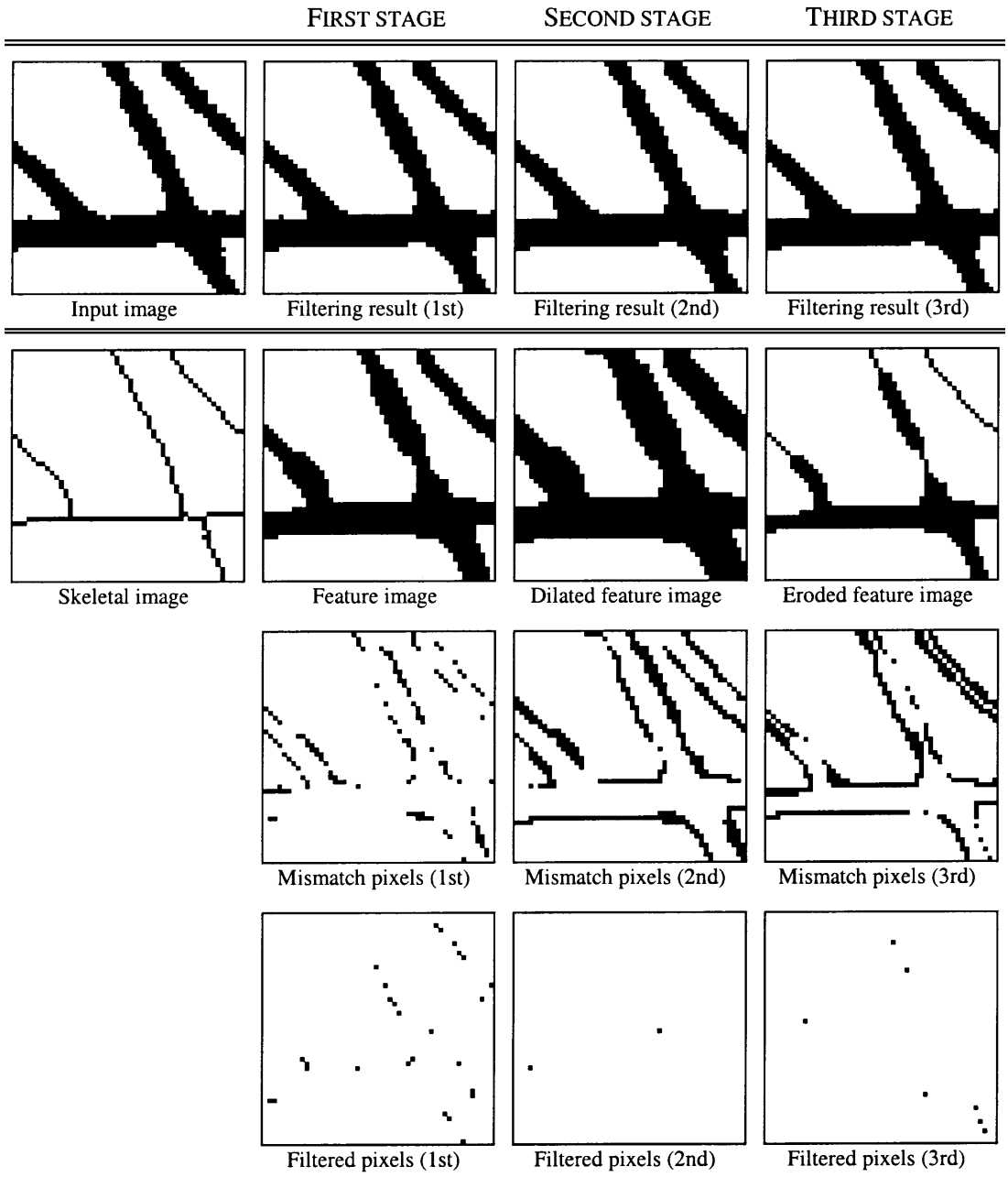

Fig. 5. Illustration of the three-stage filtering procedure.

some places, and the pixels are therefore not filtered in the first two stages. The eroded version, however, gives a more accurate approximation of the line, and more noise pixels can be detected and filtered in the third stage.

4. CONTEXT-BASED COMPRESSION

We apply sequential JBIG as the compression component, but basically there are no restrictions on using any other compression method. For example, the *progressive* JBIG [2] or ITU Group 3 or 4 could also be used. In sequential JBIG the pixels are coded by adaptive arithmetic coding, namely the QM-coder, on the basis of their probability estimates in respect to the context-based model. The context is defined by the combination of already coded neighbouring pixels. The adaptation process starts from scratch, and statistics are updated after each pixel is coded. It allows the model to adapt dynamically to the image during the coding process. The probability estimation and statistics update are derived from arithmetic coder renormalisation and implemented as a state automaton (see Pennebaker and Mitchell [3] for details).

5. TEST RESULTS

We compare the proposed method with two existing compression standards, JBIG and ITU Group 4. In JBIG we use the basic implementation with a three-line, ten-pixel context template with a default position of the adaptive pixel. ITU Group 4 is the older facsimile standard based on the two-dimensional READ-code [14]. We compress a set of 28 test images, divided into four classes: *electrical circuits*, *engineering drawings*, *cartographic maps* and *architectural and urban plans* (see Fig. 6). The images are taken from real-life applications, and amount to about 43 Mbytes in uncompressed form. The format of the images varies from A4 to A2 (see Table 1 for details of the test set). The compression results are summarised in Table 2. The compressed vector file represents the result of the vectorising when the chain-coded elements are compressed by ZIP (a commonly used file compression method). The vector representation is not space efficient because it cannot represent small objects efficiently. The feature-based filtering with JBIG can compress the raster images using less than half of the size required by the vector file. The corresponding compression ratios (in total) are 15:1 for the vector file, 33:1 for JBIG and 40:1 for the proposed method. At the same time, the quality of the decompressed images is visually the same as the original, since only isolated groups of mismatch pixels are filtered. The quality is sometimes even better, because the filtered pixels are mainly quantisation noise near the borders of line segments.

The preprocessing slows down the encoding, which is now about 2.7 times slower than without the preprocessing (see Fig. 7). The throughput of our prototype software is about 6.5 kilobytes of raw data per second in a 200 MHz Pentium processor. Currently, the bottleneck of the method is the filtering stage, which requires several separate passes. Although it is a good approach for modular implementation, the filtering could be implemented as a one-pass procedure, and essential speed-up would be gained. The vectorising stage itself is rather quick, requiring only 10% of the total running time. The decompression, on the other hand, is as fast (or as slow) as the compression component.

Finally, we consider existing filtering techniques when adopted for the same near-lossless context. We apply the

Table 1. Statistics of the test set

	No. of images	Total size	Smallest	Largest
Circuits	6	5.8 Mbytes	1480 × 2053	5522 × 4039
Drawings	8	13.2 Mbytes	1765 × 1437	7296 × 4903
Maps	5	12.9 Mbytes	3100 × 3475	6608 × 4677
Plans	9	10.0 Mbytes	1253 × 970	5888 × 5888
TOTAL	28	42.8 Mbytes	–	–

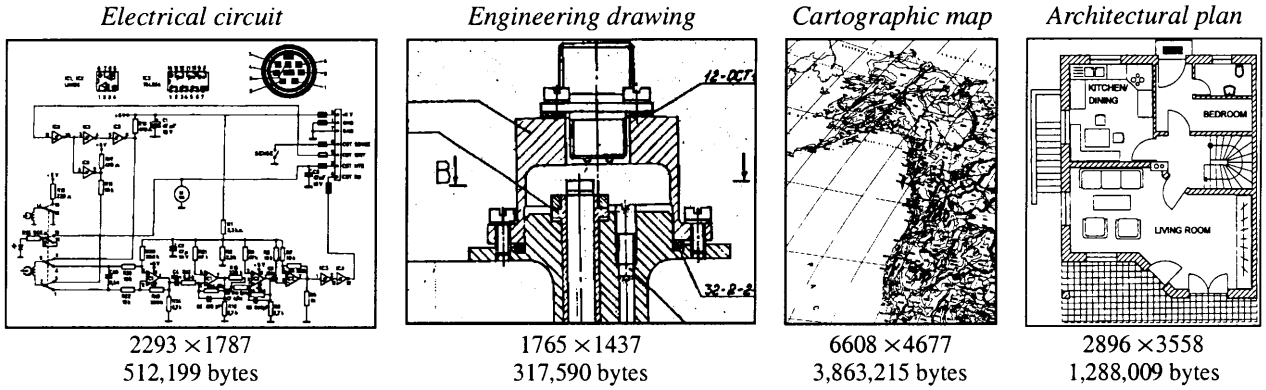

Fig. 6. Sample test images taken from each category.

Table 2. Summary of the compression results (in bytes)

	Original raster image	Compressed vector file	ITU Group 4	Filtering + Group 4	JBIG	Filtering + JBIG
Circuits	6,092,892	268,953	220,430	193,702	150,119	122,799
Drawings	13,807,484	488,210	413,732	397,028	254,917	231,715
Maps	13,476,580	1,720,864	1,040,105	858,243	706,080	557,402
Plans	10,460,683	429,792	353,375	336,205	206,010	184,447
TOTAL	43,837,639	2,907,819	2,027,642	1,785,178	1,317,126	1,096,363
Ratio	–	15.1	21.6	24.6	33.3	40.0

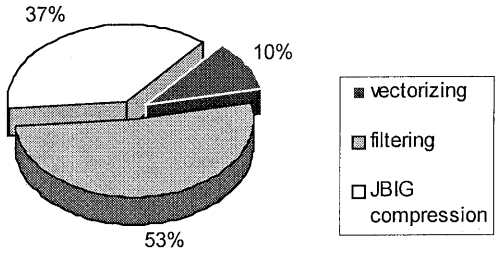

Fig. 7. Relative running times of the different components. Decompression time is virtually equal to the compression time (white slice).

traditional *Median filter* [13], and a combination of three morphological filters: *opening*, *closing* and *annular filter* [11]. The results of the filtering are fed into the noise removal process shown in Fig. 3 to allow only isolated groups of noise pixels to be filtered. In this way, the compression method remains near-lossless. The compression improvement due to these filtering methods are summarised in Fig. 8.

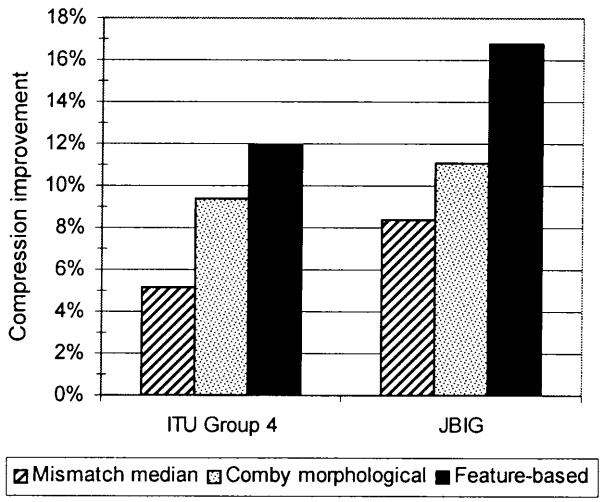

Fig. 8. Comparison of the various filtering methods when plugged-in with the two compression standards. The numbers are the relative reduction in file size when compressing the entire image set.

6. CONCLUSIONS

A three-stage compression method for compressing bi-level line-drawing images has been proposed. The method uses feature-based filtering for image preprocessing. The filtering removes additive and quantisation noise from the original image, restores image quality, and in this way results in a better compression performance. The actual compression is performed by JBIG, but any other existing method could be used. For a set of test images, the method improves the compression ratio by about 20% in comparison to JBIG.

A drawback of the method is that the compression phase is now more complex, and the method needs several passes over the image. Fortunately, vectorising can be performed rather quickly. Moreover, the vector features are not stored in the compressed file, and the process is therefore invisible in the decompression phase. The method can thus be considered as a preprocessing step to existing compression techniques, and standard decompression routines can be applied.

The method could be developed further by improving the quality of the vectorising. The width of the lines was sometimes mispredicted, resulting in a visually disturbing feature image. This does not degrade the output image quality, but it may weaken the compression performance. This was partially compensated for by the three-stage filtering process, which makes the noise removal less sensitive to the inaccuracies in the vectorising part.

A mixed vector-raster image representation could also be used by storing the extracted vector features in the compressed file. The vector information can be used in the compression by having a two-layer context template with pixels both from the original and from the feature images. Such an approach was considered by Fränti et al [19], but the experiments showed that the overhead from storing the vector elements usually outweighs the extra benefit in compression. The vector features, on the other hand, could also be used in image indexing and retrieval, but this is a point for future studies.

Acknowledgements

The work of Pasi Fränti was supported by a grant from the Academy of Finland.

References

1. Arps RB, Truong TK. Comparison of international standards for lossless still image compression. Proc IEEE, 1994; 82:889–899
2. JBIG, Progressive Bi-level Image Compression, ISO/IEC International Standard 11544, ITU Recommendation T.82, 1993
3. Pennebaker WB, Mitchell JL. JPEG Still Image Data Compression Standard. Van Nostrand Reinhold, 1993
4. Witten IH, Moffat A, Bell TC. Managing Gigabytes: Compressing and Indexing Documents and Images. Van Nostrand Reinhold, New York, 1994
5. Howard PG. Text image compression using soft pattern matching. The Computer Journal 1997; 40 (2/3):146–156
6. Ting D, Prasada B. Digital processing techniques for encoding of graphics. Proc IEEE 1980; 68 (7):757–769
7. Algazi VR, Kelly PL, Estes RR. Compression of binary facsimile images by preprocessing and color shrinking. IEEE Trans Commun 1990; 38 (9):1592–1598
8. Zhang Q, Danskin JM. Bitmap reconstruction for document image compression. SPIE Proc Multimedia Storage and Archiving Systems 1996; Boston, MA, Vol. 2916: 188–199
9. Serra J. Image Analysis and Mathematical Morphology. Academic Press, London, 1982
10. Schonfeld D, Goutsias J. Optimal morphological pattern restoration from noisy binary images. IEEE Trans Pattern Analysis and Machine Intelligence 1991; 13 (1):14–29
11. Heijmans HJAM. Morphological Image Operators. Academic Press, Boston 1994
12. Dougherty ER, Astola J (eds). Nonlinear Filters for Image Processing, SPIE Optical Engineering Press, 1997
13. CCITT, Standardization of Group 3 Facsimile Apparatus for Document Transmission, ITU Recommendation T.4, 1980
14. CCITT, Facsimile Coding Schemes and Coding Control Functions for Group 4 Facsimile Apparatus, ITU Recommendation T.6, 1984
15. Kolesnikov AN, Belekhov VI, Chalenko IO. Vectorization of raster images. Pattern Recognition and Image Analysis 1996; 6 (4):786–794
16. Arcelli C, di Baja GS. A one-pass two-operation process to detect the skeletal pixels on the 4-distance transform. IEEE Trans on Pattern Analysis and Machine Intelligence 1989; 11 (4):411–414
17. Kolesnikov AN, Trichina EV. The parallel algorithm for the binary images thinning. Optoelectronics, Instrumentation and Data Processing 1995; (6):7–13
18. Hung SHY, Kasvand T. Linear approximation of quantized thin lines. In Haralick RM (ed) Pictorial Data Analysis, Springer-Verlag, Berlin, 1983; 15–28
19. Fränti P, Ageenko EI, Kälviäinen H, Kukkonen S. Compression of line-drawing images using Hough transform for exploiting global dependencies, Proc JCIS 1998; 4: 433–436

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

Eugene I. Ageenko received his MSc degree in applied mathematics from Moscow State University, 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.

Alexander Kolesnikov received his MSc degree in physics in 1976 from Novosibirsk State University, Russia. Currently he is a senior research scientist in the Institute of Automatics and Electrometry, Russian Academy of Sciences, Novosibirsk, Russia. His primary research interests are in signal and image processing, image compression and medical imaging.

Correspondence and offprint requests to: P. Fränti, Department of Computer Science, University of Joensuu, Box 111, FIN-80101 Joensuu, Finland