Fast and memory efficient text image compression with JBIG2

Yan Ye and Pamela Cosman*
Electrical and Computer Engineering Dept., MC 0407
University of California at San Diego
La Jolla, CA 92093-0407
Email: {yye,pcosman}@code.ucsd.edu
Phone: (858)822-0157 Fax: (858)822-3426
EDICS: 1-STIL(Still Image Coding)

Abstract

In this paper, we investigate ways to reduce encoding time, memory consumption and substitution errors for text image compression with JBIG2. We first look at page striping where the encoder splits the input image into horizontal stripes and processes one stripe at a time. We propose dynamic dictionary updating procedures for page striping to reduce the bit rate penalty it incurs. Experiments show that splitting the image into two stripes can save 30% of encoding time and 40% of physical memory with a small coding loss of about 1.5%. Using more stripes brings further savings in time and memory but the return diminishes. We also propose an adaptive way to update the dictionary only when it has become out-of-date. The adaptive updating scheme can resolve the time vs. bit rate trade-off and the memory vs. bit rate trade-off well simultaneously. We then propose three speedup techniques for pattern matching, the most time-consuming encoding activity in JBIG2. When combined together, these speedup techniques can save up to 75% of the total encoding time with at most 1.7% of bit rate penalty. Finally, we look at improving reconstructed image quality for lossy compression. We propose enhanced prescreening and feature monitored shape unifying to significantly reduce substitution errors in the reconstructed images.

Keywords

JBIG2, text image compression, soft pattern matching, pattern matching and substitution.

I. INTRODUCTION

The JBIG2 standard [1], [2] is the new international standard for bi-level image compression. Bi-level images have only one bit-plane, where each pixel takes one of two possible colors. Prior to JBIG2, facsimile standards such as ITU-T recommendations T.4, T.6, and T.82 (JBIG1) [3], [4], [5], [6], [7] provide only for lossless compression of bi-level images. JBIG2 is the first one that also provides for lossy compression. A properly designed JBIG2 encoder not only achieves higher lossless compression ratios than the other existing standards, but also enables very efficient lossy compression with almost unnoticeable information loss [8].

A typical JBIG2 encoder first segments an image into different regions [9] and then uses different coding mechanisms for text and for halftones. In this paper, we are concerned with compressing text images. We define text images as bi-level images which consist mainly of repeated text characters and possibly some general graphic data (e.g., line art) but no halftones. In JBIG2, the coding of text is based on pattern matching techniques [1]. JBIG2 defines two modes for text compression: *pattern matching and substitution* (PM&S) [10], [11] and *soft pattern matching* (SPM) [12]. For general graphic data not identified as text, the encoder uses a basic bitmap coder such as specified by JBIG1 or T.6. This is called cleanup coding.

On a typical page of text, there are many repeated characters. We call the bitmap of a character instance a "symbol." We can extract symbols from the input image using a standard connected component analysis algorithm [13]. In English and many other languages, most characters are represented by one connected piece and hence are extracted as one symbol. For characters that contain separated parts, e.g., English letter "i" or "j", we use a postprocessing step to identify a dot (a small symbol) and put it back onto its stem to form one symbol.

Rather than coding all the pixels of each symbol on the page, we code the bitmaps of a representative subset and put them into the *symbol dictionary*. Then, each symbol on the page is coded by giving its position on the page and the index of its best matching symbol in the dictionary. In the PM&S mode, the bitmap of the best match dictionary symbol gets directly substituted for the current symbol on the reconstructed page. In the SPM mode, we transmit a lossless coding of the current symbol's actual bitmap based on that of its matching dictionary symbol. This lossless coding, called refinement coding, is done by context-based arithmetic coding using a context drawn from both the best match bitmap, and the already coded part of the current bitmap [1]. In our work, we use the Hamming distance based matching criterion which measures the percentage of different pixels between two symbols.

The idea of text image compression based on pattern matching appeared several decades ago [10], [11]. However, the main obstacle to its practical implementation was its high cost. From the point of view of physical memory consumption, buffering the entire input page (or a big portion of it as in page striping) is much more expensive than buffering only a few lines of the input page, as needed in T.4, T.6 or T.82 (JBIG1). From the point of view of encoding time, pattern matching is computationally very intensive. Recent advances in the CPU and memory technologies have made it possible to practically implement pattern matching based text image coding systems. However, it is still of great importance for many applications to limit physical memory consumption and/or to encode faster. In this paper, we investigate several techniques to lower memory consumption and to reduce encoding time in JBIG2. To save physical memory, JBIG2 allows *page striping* where the encoder splits a page image into horizontal stripes of approximately equal sizes and processes one stripe at a time. Because the encoder deals with fewer symbols at a time, page striping also reduces encoding time. The disadvantage of page striping is that it offers lower compression efficiency compared to coding the page as a whole. However, since text symbols on the same page are usually very similar, when coding the current stripe, some of the existing dictionary symbols can be re-used to reduce the compression loss. In JBIG2, this is done by sending a 1-bit flag for each dictionary symbol to signal to the decoder whether the current symbol is to be retained or discarded after the current stripe is decoded. In this paper, we propose dynamic dictionary updating procedures to retain useful dictionary symbols and discard obsolete ones [15]. We also investigate two encoding trade-offs in page striping: the coding time *vs*. bit rate trade-off and the memory usage *vs*. bit rate trade-off. We propose an adaptive dictionary updating scheme that can resolve both trade-offs favorably at the same time.

A JBIG2 coding system for text images consists of several components: symbol extraction, pattern matching, arithmetic/Huffman integer/bitmap coding, and so on. To speed up arithmetic bitmap coding, JBIG2 allows typical prediction (TP) as specified in JBIG1 [5] and typical prediction for residue (TPR) as proposed in [14]. In this paper, we focus instead on reducing the encoding time spent on pattern matching. In lossless SPM mode, our experiments show that, even using the simple Hamming distance matching criterion, pattern matching can take as much as 90% of the total encoding time. In this paper, we propose three speedup techniques that significantly reduce the amount of pattern matching time while losing little in coding efficiency. These speedup techniques are more efficient than page striping in terms of trading off compression and coding time. Nevertheless, page striping is still necessary for applications with limited physical memory.

In lossy compression, we consider one more figure of merit, the reconstructed image quality. Criteria such as Peak Signal to Noise Ratio (PSNR) commonly used in measuring gray-scale image quality are not suitable for bi-level images. For bi-level text images, it is very important to be able to correctly recognize as many text characters as possible at the receiver. If a pair of corresponding characters in the original and the reconstructed images are perceived to be different by a human observer, then a substitution error has occurred. In this paper, we use the number of substitution errors as a quantitative measure for the reconstructed image quality, and we propose techniques for suppressing substitution errors.

This paper is organized as follows. In Section II, we elaborate on page striping and propose to update the current dictionary from dictionaries for previous stripes. We give results on the savings in time and memory usage and the bit rate penalty incurred. We also compare the performance of five dictionary construction schemes when used in page striping. In Section III, we propose three speedup techniques for pattern matching. In Section IV we propose ways to suppress substitution errors in the reconstructed images for lossy PM&S and lossy SPM. We conclude our paper in Section V.

II. Dynamic dictionary construction for page striping

In this section, we quickly review four previous dictionary designs, the one-pass, singleton exclusion, class-based and tree-based dictionaries, and propose a new static dictionary design called the modified-class design. To reduce the bit rate penalty incurred by page striping, we propose dynamic dictionary updating techniques for the singleton exclusion dictionary and the modified-class dictionary. We investigate the encoding trade-offs between memory and bit rate and between coding time and bit rate when using different dynamic dictionary construction schemes. In particular, we propose an adaptive dictionary updating scheme that can resolve both trade-offs favorably at the same time.

A. Static symbol dictionary design

In this section, we take lossless compression as an example. We will address lossy compression in further detail in Section IV. The one-pass (OP) dictionary [12] is formed in a sequential way. The encoder matches each newly extracted symbol with the current dictionary. If the lowest mismatch found is below a preset threshold, the new symbol is encoded with refinement coding using the best match as its reference. Otherwise, the new symbol is encoded directly using a JBIG1 type of arithmetic coder; this is called direct coding. Either way, the new symbol is added to the dictionary. The main disadvantage of the OP dictionary is that it contains many singletons which are symbols never referenced by any other symbols [17]. Singletons are detrimental to coding efficiency because they do not provide any useful

reference information yet dictionary indices are assigned to them anyway, thus increasing the average length of all indices. By excluding singletons from the OP dictionary, we obtain the singleton exclusion (SE) dictionary.

Previously we have proposed the class-based (CLASS) [18] and tree-based (TREE) [19] symbol dictionary designs for SPM-based JBIG2. Compared with the simpler OP and SE dictionaries, the CLASS and TREE dictionaries can improve compression by up to 8% for lossless and 17% for lossy compression [8]. In this paper, we propose a new dictionary design called the modified-class (MC) design [15] which combines the ideas of the CLASS and TREE designs. Design of the MC dictionary follows two steps. At the first step, as in the CLASS design, we group all extracted symbols into classes by pointing them to their closest match. For each class, we choose its representative as the symbol with the lowest average mismatch within the class. We put all representatives into the dictionary. The second step follows the idea of the TREE design. We connect each pair of symbols with a weighted edge where the weight is the mismatch score between the two symbols; if the mismatch score is bigger than the threshold, then there is no edge connecting the symbol pair. This way we obtain matching graphs among all dictionary symbols (i.e., class representatives). We then construct minimum spanning trees (MSTs) from these matching graphs using Kruskal's algorithm [20]. For each MST, we choose its root randomly as any node with degree bigger than 1 [19], [8]. The MC design improves over the CLASS design because the reference relationships among all dictionary symbols as given by the MSTs have the lowest total mismatch (the CLASS design uses the concept of super-classes which are suboptimal). The MC design is also computationally less complex than the TREE design.

B. Dynamic dictionary update

Page striping is an encoding mode defined in the JBIG2 standard [1] that allows the encoder to split the input page into horizontal stripes of approximately equal sizes and encode one stripe at a time. Page striping lowers memory requirements for both the encoder and the decoder. Another benefit of page striping is that it reduces the encoding time by reducing the time spent on pattern matching. To decide which symbols from the input page will go into the dictionary, the encoder needs to perform pattern matching on all extracted symbols. Therefore, if the input page contains N symbols in total, and page striping is not used, the time needed for pattern matching is proportional to N^2. By splitting the image into two stripes, the pattern matching time can be approximately cut in half ($2 \times (N/2)^2 = N^2/2$). However, page striping lowers compression efficiency if the encoder sends completely separate dictionaries for each stripe. To reduce this coding loss, rather than coding each stripe completely separately, we reuse some of

the dictionary symbols from previous stripes to code the current stripe. This is based on the observation that the fonts and sizes of the text characters in one page are usually very similar. We propose separate updating processes for the SE dictionary and the MC dictionary.

Updating an SE dictionary is straightforward. For each new symbol in the current stripe, the encoder matches it with not only all the previous symbols in the current stripe, but also all the symbols from the dictionary used for the previous stripe. The encoder then uses its closest match as its reference symbol and adds the new symbol to the dictionary. After the current stripe is processed, the encoder examines the new dictionary and excludes all singletons from it. Those previous dictionary symbols that are not used by any symbol in the current stripe are also expunged. This way, new symbols useful for the current stripe get included in the new dictionary, and old dictionary symbols that are obsolete are discarded.

The design of an MC dictionary consists of two steps, the first of which is to form classes and choose representatives. In the dictionary updating procedure, we perform this step on the combined set of all previous dictionary symbols and all new symbols from the current stripe. If a previous dictionary symbol has the lowest average in-class mismatch, it will be naturally selected as the representative, which means the encoder can directly reuse its bitmap without sending it to the decoder again. In the case that the symbol with the lowest in-class mismatch is not an existing dictionary symbol, if there is an existing dictionary symbol whose average in-class mismatch is slightly higher than the lowest one but the difference between them is below a preset threshold, we still choose the existing dictionary symbol as the representative. This allows us to make use of many previous dictionary symbols; we only choose a new symbol as the representative if all the existing ones are too inaccurate. The second design step is to form MSTs for the dictionary symbols. In Figure 1, we show pre-existing dictionary symbols in gray and new ones from the current stripe in black. Numbers along the edges indicate the mismatch scores between the symbols. With these mismatch values (Figure 1 (a)), Kruskal's algorithm will produce an MST that includes edges connecting the four gray nodes. However, the mismatch scores and reference relationships among the pre-existing symbols are meaningless because the decoder already has their bitmaps. Therefore, we can connect all existing symbols with zero-weight edges (the dash-dotted gray edges in Figure 1 (b)). We then go on and apply the usual Kruskal's algorithm. This guarantees that each resulting MST has at most one gray node representing a previous dictionary symbol; some MSTs may have no gray nodes if they consist of new symbols from the current stripe only. For an MST containing one existing dictionary symbol, this symbol is used as the tree root since its bitmap is already known to the decoder. For an MST containing only symbols from the current stripe, its root is selected randomly as any node with degree bigger than 1

as in the static design. After the new dictionary is decided, those previous dictionary symbols that are not used in the new dictionary are considered obsolete and will be excluded.

C. Encoding trade-offs in page striping

Page striping reduces memory usage and encoding time but incurs a bit rate penalty. In this paper, we focus on two encoding trade-offs in page striping: the trade-off between encoding time and bit rate and the trade-off between memory usage and bit rate. We compare several dictionary construction schemes for page striping in terms of their performances in both trade-offs.

Depending on the characteristics of the text in the input page, we should use different dictionary construction schemes for page striping. If the text contained in the first stripe is a very accurate representation of the text in the entire page, then we can design the dictionary only once from the first stripe and use it throughout the entire page. We call this the *static scheme*. On the contrary, if the text in the current stripe is completely different from that in the previous stripe (e.g., the previous stripe contains regular English text and the current stripe contains math symbols), then we should design a completely isolated dictionary for the current stripe using only text symbols from the current stripe. We call this the *isolated scheme*. The more general case is that some text symbols in the current stripe are similar to those in previous stripes but there are also new symbols not seen before. In this case, we should use the proposed dynamic updating procedures to reuse certain previous dictionary symbols, discard those that are obsolete, and add new symbols from the current stripe into the dictionary if necessary. We can update the dictionary for every new stripe (we call it the *dynamic scheme*), or we can update the dictionary for every other stripe (we call it the *dynamic-2 scheme*). Compared to the static or the isolated scheme, the dynamic scheme reduces the bit rate penalty incurred by page striping but also takes longer to encode. This is because the dynamic scheme needs to perform additional pattern matching between symbols in the current stripe and symbols in the previous dictionary, and decide which ones to reuse, to discard, or to add. Compared to the dynamic scheme, the dynamic-2 scheme reduces the encoding time by updating the dictionary half as frequently. However, how often the dictionary is updated should ultimately depend upon the rate at which text symbols change from stripe to stripe. Since this text change rate is not known beforehand and is often not constant within a page, we propose an adaptive dictionary updating technique that automatically decides if the existing dictionary has become out-of-date (i.e., enough symbols in the current stripe can not be represented by the existing dictionary symbols) and updates the dictionary *only* when it is out-of-date. We will call this the *adaptive scheme*.

C.1 Adaptive dictionary update

One property that the adaptive dictionary updating scheme must have is that the decision about whether the dictionary has become out-of-date must be made quickly. A complicated decision will prolong the encoding time and negatively affect the time vs. bit rate trade-off. We propose a simple and fast procedure to automatically decide if the dictionary is out-of-date. The dictionary is updated at most every two stripes. This means if the dictionary has just been updated for the previous stripe, then the encoder will use it directly to code the current stripe. But during the coding of the current stripe, the encoder calculates two values, the average mismatch and the percentage of unmatched symbols for the current stripe. These two values show how well the symbols in the current stripe can be represented by the existing dictionary symbols. The encoder then compares these two values for the current stripe with those for the previous stripe. If either value has increased significantly, i.e., *either* the current average mismatch is more than 1.5 times as big as the previous average mismatch *or* the current unmatched percentage is twice the previous unmatched percentage, then the encoder decides that the existing dictionary has become out-of-date. The encoder then switches on the UPDATE_DICT flag and updates the dictionary for the subsequent stripe. Note that the calculation of the average mismatch and unmatched percentage is very fast since it can be carried out at the same time as the current stripe is being encoded.

D. Experimental results

Unless otherwise stated, all experimental results presented in this paper are obtained from a set of twelve test images from two sources.

1. Two CCITT images that are mainly textual: f01_200 and f04_200. Their resolution is 200 dpi, size 1728×2339 pixels;

2. Ten images (IG0H, J00O, N03F, N03H, N03M, N046, N04D, N04H, N057 and S012) selected from the University of Washington Document Image Database I [21]. This database contains about 980 scanned document images. The 10 images we selected are mostly streak-free, not obviously skewed, from various sources, and contain mainly text, little line art and no halftones. All ten images have 300 dpi resolution. Eight of the images have the same size 2592×3300 pixels, while N03H has size 2480×3508 and S012 2536×3308.

All experiments are carried out on a Pentium Pro 200MHz, running Red Hat Linux 6.0, with 64MB physical memory. We measure encoding time (in sec) using the function “clock()” and peak memory usage (in MB) using the Unix command “top”. We give results that are averaged over all test images. Our

code was not specifically optimized for speed or memory efficiency.

D.1 The modified-class dictionary

Table I summarizes the lossless and lossy coding efficiencies of all the five dictionaries (OP, SE, CLASS, TREE and MC). Detail on how lossy coding is performed will be presented in Section IV. We show the average coded file sizes and also the percentages of improvement over the least efficient OP dictionary. Compared to the OP dictionary, the compression improvements from the CLASS, TREE and MC dictionaries are approximately the same, about 8% for lossless coding and 16-18% for lossy coding. For lossless compression, the proposed MC design is basically the same as the CLASS design while slightly worse than the TREE design. However, for the TREE design, the numbers listed are the best compression achieved at the optimal dictionary sizes; the encoder has to exhaustively search for these optimal sizes [19], [8]. For lossy compression, the MC design achieves the best compression.

D.2 Encoding trade-offs in page striping

In this section, we show the savings in encoding time and memory usage when page striping is applied. Figure 2 plots encoding time, peak memory usage, and coded file size as functions of the number of stripes into which a page is split. The dynamic scheme and the isolated scheme using the SE design and the MC design are compared. The results shown are for lossless compression; similar results are obtained for lossy compression.

The savings in encoding time from page striping are shown in Figure 2 (a). By splitting a page into two stripes, the isolated scheme reduces encoding time by 45% for both dictionaries (close to the theoretical savings of 50%); the dynamic scheme reduces encoding time by 32% for the MC dictionary and 26% for the SE dictionary. The dynamic scheme provides less time reduction because, instead of starting from scratch for each stripe, it needs to consider those previous dictionary symbols and decide how to use them. Splitting the input image into more stripes brings more savings in encoding time but the returns diminish. Comparing the four curves with the curve $1/n$ (dotted curve) in Figure 2 (a), we see that the four curves deviate from the curve $1/n$ as the number of stripes increases. This is because the total encoding time consists of two parts, pattern matching and other encoding activities (e.g., symbol extraction, arithmetic bitmap and integer coding, etc.). While the pattern matching time is roughly inversely proportional to the number of stripes, the time spent on the other activities does not go down as the number of stripes increases. In the next section, we will show that, for a fixed input image, the time spent on these other encoding activities is almost fixed.

Figure 2 (b) shows the peak memory usage as a function of the number of stripes used. We see that the dynamic and isolated schemes using both SE and MC designs require basically the same amount of physical memory. This is because although the dictionaries of the four curves are of different sizes, the memory needed to buffer the dictionaries only accounts for a very small percentage of the total memory usage. Most of the memory is for buffering a page or page stripe. By splitting a page into 2 stripes, we save about 40% of the peak memory consumption. Using more stripes brings more savings in memory consumption but with diminishing returns. The curves flatten out after 6 stripes as each stripe becomes small enough that the memory needed to buffer it no longer dominates.

While page striping reduces encoding time and memory usage, this comes at the price of reduced compression efficiency. As shown in Figure 2 (c), the compressed bit rates increase steadily as the page is coded using more stripes. Using dynamic dictionaries minimizes this bit rate penalty. For the MC design, the isolated scheme with 8 stripes has 18% higher bit rate than with 1 stripe, but the dynamic scheme reduces this bit rate penalty to 13%; for the SE design, the isolated scheme with 8 stripes has 18% higher bit rate than with 1 stripe, but the dynamic scheme reduces this bit rate penalty to 11%.

Figure 3 provides a convenient way to evaluate the trade-off between coding time and bit rate by showing what bit rate can be achieved at a given coding time using a certain dictionary scheme. Figure 3 compares the five dictionary schemes aforementioned. The number of stripes used varies from 1 to 8. Using only 1 stripe (i.e., whole page) encodes the slowest but produces the smallest coded file size (the lower-right corner in Figure 3); using 8 stripes runs the fastest but produces the biggest coded file size (the upper-left corner in Figure 3). The dashed lines in Figure 3 are the lower convex hull for all the operating points. Points on this lower convex hull achieve the best compression using the shortest encoding time. In Figure 3, this lower convex hull is defined by the static scheme (square markers). The proposed adaptive scheme ("+" markers) operates very close to the lower boundary of convex hull, achieving time *vs*. bit rate trade-off similar to that of the static scheme. The dynamic scheme ("x" markers) is the least time efficient dictionary scheme as it operates the farthest from the lower convex hull.

To compare the performance in the memory *vs*. bit rate trade-off by the five dictionary schemes, we plot coded file size as a function of peak memory consumed in Figure 4. Similarly, the input images are coded as 1 to 8 stripes. The dashed lines show the lower convex hull for all the operating points. The dynamic scheme ("x" markers) now defines this lower convex hull, meaning that it achieves the best compression using the least system memory. The static scheme (square markers), which achieves the best time *vs*. bit rate trade-off, becomes the least efficient in resolving the memory *vs.* bit rate trade-off. The proposed

adaptive scheme ("+" markers) still operates very close to the lower convex hull. Combined with the results shown in Figure 3, we conclude that the adaptive scheme is a robust scheme in resolving both encoding trade-offs well. Hence the adaptive scheme is a suitable choice for most applications, where system memory and encoding time are both very important system parameters.

D.3 Multi-page document compression

Multi-page document images are a set of images scanned from the same source, preferably from consecutive pages. Some issues of compressing multi-page document images are addressed in [22]. In multi-page document compression, the same dictionary updating processes used in page striping can also be applied to take advantage of the text correlation across pages.

Tables II and III compare the coding efficiency on three multi-page document image sets using three dictionary schemes (the isolated, static, and dynamic schemes) combined with the SE design and the MC design, respectively. Among the three test sets, two are from the University of Washington Document Image Database I, one of 4 pages (N04H, N04I, N04L and N04M) and the other of 5 pages (N01F, N01G, N01H, N01I and N01J). They are from the same source, but not from consecutive pages. Their scanning conditions are unknown. The third set is an 11-page document we scanned in from [23], at 300 dpi. The scanned pages are consecutive and the scanning conditions are consistent for all pages.

Tables II and III show that, compared to the isolated scheme, the dynamic scheme can improve compression by up to 8% for the MC design and 10% for the SE design. Another interesting phenomenon is that the static scheme using the SE design also achieves 4-5% of improvement over the isolated scheme (see Table II). When coding a single-page document, the SE dictionary is usually twice as big as the MC dictionary, containing redundant bitmap information; it is therefore less efficient due to high index coding cost [8]. When coding a multi-page document, however, it is advantageous to use a bigger and more redundant dictionary throughout all the pages because it gives the symbols from later pages a broader range of choices. We hardly see any improvement from the static scheme using the MC design because the MC design is too specifically designed for only the first page. Figure 5 shows the dictionary size growth curves from page to page. For our 11-page test set, from the fourth page on, the dictionary size becomes steady, showing that the encoder has gathered most useful bitmap information contained in this document set. The other two test sets do not contain enough pages to show this trend.

Figure 6 shows the time *vs*. bit rate trade-off for the five dictionary schemes when tested on multi-page documents combined with page striping. The results are averaged over the three multi-page test sets. Five values for the number of stripes per page are used, 1, 2, 4, 8 and 16. At the lower-right corner in the

Figure, each page is encoded as a whole (the number of stripes is 1). At the upper-left corner, each page is processed as 16 stripes. The lower convex hull (given as the dashed lines) is still mostly defined by the static scheme. The adaptive and dynamic-2 schemes operate very close to the lower boundary, with a couple of points falling on it. For the memory *vs*. bit rate trade-off, we observe the same relationship between the five schemes as shown in Figure 4.

Summary: Page striping reduces encoding time and physical memory usage with reasonably low bit rate penalty. In page striping, compared to sending isolated dictionaries for each stripe, dynamically updating the dictionary can significantly reduce the bit rate penalty incurred. The proposed adaptive dictionary updating scheme is robust and can resolve both the time vs. bit rate trade-off and the memory vs. bit rate trade-off favorably at the same time. The same dynamic dictionary updating techniques can be applied to multi-page document image compression and improve the compression ratio by up to 8-10%.

III. Speedup techniques for pattern matching

In the previous section, we propose dictionary construction schemes for page striping that can reduce memory usage and encoding time with minimal sacrifice in coding efficiency. In this section, we propose three speedup techniques for pattern matching. Compared to page striping, these techniques can better resolve the trade-off between coding time and bit rate.

A. Limited dictionary symbol search

To design the MC dictionary, we group all symbols into classes, choose class representatives to go into the dictionary, and form MSTs for all the dictionary symbols. Suppose a symbol S belongs to a certain class C, whose representative is symbol R, which, after the MST construction procedure, lands in MST T. Therefore we know that the mismatch between symbol S and symbol R must be small (though not always the smallest), and that symbol R is similar (to different degrees) to all the other symbols in tree T. In addition, we know that all other trees are sufficiently dissimilar to tree T because no edge between them has weight lower than the threshold. Therefore, to find the best match for symbol S in the dictionary, it is likely that we need to search among only those symbols that belong to MST T. To do this, we maintain a *tree-ID* value for each symbol on the page, which specifies the MST to which this symbol's representative belongs. To find the matching dictionary symbol for the current symbol, we only search among those dictionary symbols that have the same *tree-ID*. This can significantly reduce the number of dictionary symbols against which the current symbol is matched. Whether this limited search algorithm will suffer significant bit rate penalty depends on how many symbols actually belong to the same MST as

their best dictionary matches. Later in this section, we show that this limited search algorithm can save encoding time at almost no coding loss.

B. Early jump-out based on previous best match

When matching one symbol with another, we save the previous lowest mismatch score; the pattern matcher compares on-the-fly the current accumulated mismatch score against the previous lowest one. If the current mismatch is already above the previous lowest, then we terminate the current matching process. Computing the Hamming distance between two symbols is fast because it only requires the exclusive-OR (XOR) operation and incrementing the mismatch score accordingly. Since comparing the two mismatch scores also takes time, and we do not want this time to be comparable to the Hamming distance calculation where we hope to save time, we do the integer comparison of mismatch scores only once per line. At the end of each line, the current accumulated mismatch is checked; if it exceeds the previous lowest, the pattern matching process terminates.

C. Enhanced prescreening

Before matching a pair of symbols, it is advantageous to prescreen them by certain features. There is no need to apply pattern matching to two symbols that are obviously dissimilar. For example, symbols that differ greatly in size (e.g. a capital “D” and a comma “,”) are obviously dissimilar. The encoder in [12] prescreens using symbol sizes; only symbols with similar sizes (defined as not more than 2 pixels different in either dimension) are given to the pattern matcher. Prescreening is intended to reduce the number of unnecessary pattern matching calls that will not return a match. At the same time, prescreening should not rule out potentially good matches. Otherwise it will incur a high bit rate penalty. Therefore, the ideal prescreening rules out all “unmatchable” symbols and passes on all “matchable” symbols to the more expensive pattern matching subroutine.

Other features can be used in prescreening besides symbol size. One such example is to use symbol area and/or perimeter [13], [24]. However, these two features are not particularly helpful for two reasons: they are correlated with symbol size, and they are usually sensitive to scanning noise and digitization parameters such as contrast [13]. A useful feature for prescreening introduced in [13] is called the quadrant centroid distance. It is calculated as follows. We divide each symbol into four quadrants and calculate the centroid for each quadrant. To prescreen two symbols, we calculate the distance between each pair of corresponding quadrant centroids, sum the four distances and compare the total to a preset threshold. A small total distance means that the two symbols have similar mass distribution in all four quadrants; only

such symbol pairs are passed on to pattern matching to be further examined.

According to our experiments, in the English language, using the Hamming distance based matching criterion, letter pairs that are among the most easily confused include “b” and “h,” “c” and “e,” and “i” and “l.” In this paper, we propose two topological features for prescreening: number of holes and number of connected components. Prescreening by these two features can effectively prevent the above symbol pairs from being handed over to the pattern matcher.

D. Experimental results

In this section, we show experimental results on the three speedup techniques proposed, the limited dictionary search algorithm based on tree-ID (TID), early jump-out (EJO), and enhanced prescreening (PRESCRN). We consider two figures of merit, the encoding time saved and the bit rate penalty incurred.

We use the same twelve test images and the same computer platform as in Section II-D. Results are averaged over all test images. Table IV gives the total encoding time, time spent on pattern matching, and coded file size for each individual technique and different combinations of them, for a lossless SPM JBIG2 encoder. Table V shows the corresponding results for a lossless PM&S JBIG2 encoder. We only show lossless coding results here because for SPM, lossy coding takes extra time to preprocess the input image, while lossy PM&S will encode faster because no residual coding (coding the original image again based on the lossy version already sent using refinement coding) [2], [8] is needed. For both cases, the amount of extra time needed or saved is fixed for a given input image. Therefore, we only consider lossless coding now; lossy coding will be considered in further detail in the next section. The first rows (NONE) in Tables IV and V refer to using no speedups and prescreening only by size, using a size offset threshold of 2 pixels (size difference can not be bigger than 2 pixels in either dimension). Using tighter size offset thresholds (i.e., 1 or 0 pixels) can further reduce the encoding time but at the price of higher coding loss. In SPM (see Table IV), pattern matching accounts for up to 90% of the total encoding time. The rest of the encoding time is a fixed value of around 8.6 seconds. For the PM&S mode (see Table V), pattern matching accounts for up to 45% of the total encoding time. The rest of the encoding time is a bigger fixed value of around 13.3 seconds. Using the NONE rows as the basis for comparison, we give the percentages of time saved and coding loss incurred from each individual speedup technique and several combinations of them. The limited dictionary search technique (TID) saves 15% of the pattern matching time, while causing almost no coding loss. Note that TID is only applicable to the SPM mode using the MC dictionary design. The early jump-out technique (EJO) saves 16% and 12% of the pattern matching time in SPM and PM&S, respectively. EJO incurs no bit rate penalty. In SPM, we can combine TID and

EJO together to achieve a pure 31% time gain with no coding loss. Enhanced prescreening is the most efficient way to save encoding time. Adding the quadrant centroid distance to the size prescreening (S+Q) saves almost 3/4 of the pattern matching time, while incurring a bit rate penalty of around 1%. Adding the numbers of holes and connected components (S+H+C) saves 40% of the pattern matching time, which is less efficient than the Q feature. However, H+C incurs only a 0.5% bit rate penalty. Combining all these speedup techniques together saves 81% and 76% of the pattern matching time in SPM and PM&S, respectively. In terms of total encoding time, these numbers translate into savings of 74% and 33%, respectively. The bit rate penalty incurred is relatively small, 1.7% for SPM and 1.3% for PM&S.

Without the TID technique, each symbol searches among all dictionary symbols for its best match. For our test image set, this means the average search range is 638 dictionary symbols. With the TID limited search method, however, the average search range is reduced to only 34 dictionary symbols, a 95% reduction. Consequently, the time spent on finding dictionary matches for all symbols is reduced to 5.30 seconds with TID from 19.66 seconds without TID. Without the EJO technique, the pattern matcher will examine in full every pair of symbols passed on to it, i.e., it will go over 100% of the bitmap area before making a decision. With EJO, however, experiments show that on average only 44% of the bitmap area will be examined. Furthermore, on average 89% of all the pattern matching calls result in early termination. Although EJO has to spend extra time comparing integer mismatch scores, it still reduces the average number of CPU clock cycles used to match two symbols from 68 to 60. An important advantage of the TID and EJO techniques is that they save encoding time almost “for free”, meaning without bit rate penalty (see Tables IV and V). To see how enhanced prescreening helps effectively rule out unlikely matches, we list the percentages of prescreening passed in Table VI. Using the symbol size (S) feature alone is not efficient enough; around 20% of the symbol pairs will still be given to the pattern matching process. Adding the number of holes and number of connected components (S+H+C) reduces the pass rate to 12%; adding the quadrant centroid distance (S+Q) only 5% of the symbol pairs can pass prescreening. Note that adding the Q feature also results in a bit rate penalty twice as big as adding H+C (see Tables IV and V). By combining all three features together with symbol size (S+Q+H+C), we can further reduce the prescreening pass rate.

Figure 7 compares the impact on the time *vs*. bit rate trade-off from the proposed speedup techniques and from page striping. Results from the dynamic scheme (‘x’ markers) and the static scheme (square markers) are shown because in page striping these two schemes bound the performance curves (see Figure 3). The MC dictionary results are shown as an example. The lower convex hull (dashed lines) is defined

by the static scheme using the speedup techniques (black square markers). For the dynamic scheme, a big performance gap between using the speedup techniques (black 'x' markers) and not using the speedup techniques (gray 'x' markers) is observed. The same performance gap for the static scheme is far less significant. This is because the dynamic scheme involves more pattern matching than the static scheme; the propose speedup techniques all aim at reducing the pattern matching time. Note that with the speedup techniques, the dynamic scheme now operates very closely to the lower convex hull. Since the dynamic dictionary achieves a given trade-off point using more stripes, it is more memory efficient.

Finally, in Figure 8 we compare lossless SPM (black markers) and PM&S (gray markers) with and without the proposed speedup techniques being applied ("NONE" and "ALL" markers). We show results using three size offset thresholds, 2, 1, or 0 pixels. Clearly SPM completely defines the lower convex hull (dashed lines) in Figure 8. In [8] we showed that SPM achieves better lossless compression at the price of longer encoding time. SPM is more time consuming mostly because it requires more extensive pattern matching. However, with the proposed speedup techniques for pattern matching, the SPM encoding time can be significantly reduced; since these techniques only incur very small bit rate penalties, SPM's higher coding efficiency is still mostly retained. If achieving high coding efficiency is of the utmost importance for an application, then it should use SPM with a loose prescreening criterion (e.g., set size offset threshold to 2 pixels). If the application is willing to tolerate a small coding loss in order to encode faster, then it should use SPM with all speedup techniques and use very tight prescreening thresholds (e.g., set size offset threshold to 0). Note that EJO and TID should always be used when applicable. For other applications with intermediate requirements, different combinations of the speedup techniques and page striping offer different trade-offs.

Summary: The three proposed speedup techniques can reduce encoding time by as much as 75% while only suffering a small coding loss of at most 1.7%. These techniques offer better trade-offs between coding time and bit rate than page striping. By applying these techniques to the SPM mode, we obtain text image coding systems that encode both efficiently and fast.

IV. Reconstructed image quality control in lossy compression

All the results given in the previous section are for lossless coding. In this section, we concentrate on lossy coding by taking into account one more figure of merit, the number of substitution errors in the reconstructed images. We propose to effectively suppress substitution errors in lossy PM&S and SPM by using the features introduced in Section III-C.

A. Lossy PM&S: enhanced prescreening

In PM&S, when a matching dictionary symbol is found, the encoder substitutes it for the actual current symbol. Therefore, PM&S is inherently lossy. When lossless coding is required, after transmitting the lossy image, the encoder uses a residual coder to refine the lossy image to its original version [2]. Using the Hamming distance matching criterion and a mismatch threshold of 20%, lossy PM&S results in many substitution errors between letter pairs such as "i" and "l," "b" and "h," "u" and "n," and so on. To reduce substitution errors, a tighter mismatch threshold (e.g., 10%) can be applied; however, this increases the encoding time and the coded file size. Alternatively, a more sophisticated matching criterion (e.g., the CTM technique proposed in [27]) can be applied, but such criteria are usually very computationally intensive. A simple and effective way to suppress substitution errors is to use the enhanced prescreening as proposed in Section III-C. For example, prescreening with the feature number of holes can easily prevent "b" and "h" from being confused; using the number of connected components easily distinguishes between "i" and "l;" and quadrant centroid distance can often tell "u" and "n" apart.

B. Lossy SPM: feature-monitored shape unifying

To achieve lossy compression with SPM, the encoder preprocesses the input image to introduce information loss. In [12] three processing techniques are proposed: *speck elimination*, *edge smoothing* and *shape unifying*. Speck elimination wipes out very tiny symbols (symbols no bigger than 2×2). Edge smoothing fixes jagged edges by flipping protruding single black pixels or indented single white pixels along text edges. Shape unifying tries to make the current symbol bitmap as similar as possible to its reference bitmap, without introducing too much visual change. This is achieved by flipping pixels in the current bitmap if they are isolated areas of difference with the reference bitmap. We use the term "isolated" to mean a 1×1, 1×2, or 2×1 block of pixels. The modified bitmap is then losslessly coded with refinement coding.

The advantage of permitting only isolated errors in shape unifying is that visual information loss in the reconstructed image is almost imperceptible. However, such a restriction also puts a limit on the lossy coding efficiency. To improve the coding efficiency, shape unifying should allow not just isolated errors, but some clustered ones as well, as long as the risk of character substitution is kept low. To limit this risk, we propose to monitor the shape unifying procedure using two features, the number of holes and the number of connected components. For each cluster of differences between the current bitmap and its match, if eliminating it will not cause the features to change, we go on with shape unifying and

eliminate this difference cluster; otherwise, we preserve it to prevent a likely substitution error from occurring. As an example, Figure 9 shows the "b" and "h" pair and the "i" and "l" pair and the difference maps between them. In Figure 9 (a), we can change the "b" bitmap not only at the isolated single-pixel location, but at all the gray pixel locations, as they will not cause the internal hole in "b" to disappear. But, the black 10-pixel cluster of differences down at the bottom must be preserved. Otherwise a reader would perceive an "h" instead of a "b." Similarly, in Figure 9 (b), we can change the "i" bitmap at all the gray locations but not at the black ones because changing the black locations will cause the "i" bitmap to be connected into one whole piece, resulting in a substitution error. Though not shown in Figure 9, feature monitoring can also help prevent substitutions between certain letter pairs that have the same features, e.g., "n" and "u" or "e" and "o." In comparing the bitmaps of "n" and "u," there are basically two areas where a substantial number of clustered pixels differ: the center top and the center bottom. Modifying the upper cluster of pixels in the "n" bitmap to match the "u" bitmap will cause the "n" to split into two separate connected components. Monitoring based on the number of connected components will prevent this. Likewise, modifying the lower cluster of pixels in the "n" bitmap to match the "u" will cause the lower opening in the "n" to close, generating one internal hole. Monitoring based on the number of holes will prevent this. If we were to consider modifying the upper and lower pixel clusters simultaneously, the "n" bitmap could become a "u" bitmap and the topological features remain the same. But we do not do that. By considering each difference cluster separately, the topological features block the bitmap alteration, thereby preventing a substitution error. Modifying the current symbols at more locations improves refinement coding efficiency by making symbols more similar to their references. At the same time, ensuring certain feature values are maintained allows us to suppress many cases of character substitutions.

C. Experimental results

We first look at lossy PM&S and show how enhanced prescreening effectively suppresses substitution errors in addition to reducing encoding time. Table VII shows the encoding time, coded file size, and percentage of substitution errors for lossy PM&S using different mismatch thresholds and prescreening features. A tight mismatch threshold of 10% results in very rare substitution errors (about 1 in every 1,000 symbols). With a looser threshold of 20%, when prescreening just by size, the system suffers excessive substitution errors of around 3%, and the reconstructed images look confusing and sometimes objectionable. With enhanced prescreening, the substitution risk is made 12 times lower at 0.25%. Moreover, encoding is made 28% faster. Although the average coded file size is 33% bigger (13,434 bytes as

opposed to 10,105 bytes), at 13,434 bytes/image the reconstructed images have satisfying quality; at only 10,105 bytes/image, some important text information from the original images is lost, which is expressed in the form of many substitution errors that we see. Although not shown here, our experiments also showed that without enhanced prescreening, the bit rate goes down steadily as the mismatch threshold goes up. With enhanced prescreening, however, further loosening the mismatch threshold will not result in further reduction in bit rate; instead the bit rate hits a floor. This again shows that enhanced prescreening can guard against excessive loss of important text information in the images. Compared to using the tight 10% mismatch threshold, the substitution risk from enhanced prescreening is only 2 times higher, while the encoding is 20% more efficient and 66% faster.

For lossy SPM, we list in Table VIII the coded file size, encoding time, and percentage of substitution errors for shape unifying with and without feature monitoring. We use three error size thresholds, 2%, 4% and 6% of the symbol size. We restrict the size of a permissible error cluster because big error clusters (even if they do not change the features) cause significant visual information loss. As a result, the reconstructed image will contain a large number of distorted text characters. Such distorted "garbage" characters, if they exist, are also counted as substitutions and included in the numbers shown in Table VIII. A bigger symbol can tolerate a bigger error cluster. Therefore, we set the error size threshold to be proportional to the symbol size, i.e., difference clusters smaller than a certain percentage of the symbol size are deemed ignorable. Compared to the unmonitored version, feature monitored shape unifying suffers 55-65% fewer substitution errors at all three error size thresholds, meaning that it can more effectively avoid losing visually important text information. However, feature monitored shape unifying is more computationally demanding because every cluster of differences with size below the threshold has to be checked to see if ignoring it will result in change of features. The feature monitored version takes about 40% longer to encode than its unmonitored counterpart.

To compare lossy PM&S using enhanced prescreening with feature monitored lossy SPM, we compare the two shaded entries in Tables VII and VIII. At similar bit rates (13,024 and 13,434), monitored lossy SPM suffers 6 times fewer substitution errors (0.04% compared to 0.25%) but also takes 15 times longer to encode (156 sec compared to 10 sec). Furthermore, compared to using a tight 10% mismatch threshold in PM&S (last row in Table VII), monitored SPM using a 2% threshold (shaded entry in Table VIII) is 20% more efficient (13,024 compared to 16,687) and 2/3 less subject to substitution errors (0.04% compared to 0.12%) but takes 5 times longer to encode. For an application that does not require real-time communications, lossy SPM is a better choice because it offers better reconstructed image quality

at lower or comparable bit rates. A real-time application, however, should choose the PM&S mode with enhanced prescreening because it is much faster and offers satisfactory quality.

Finally, Fig. 10 shows a portion of the original image N03H (Fig. 10 (a)) and a set of reconstructed images from lossy PM&S and SPM using different system setups. For lossy PM&S (Fig. 10 (b)-(d)), enhanced prescreening effectively suppresses the substitutions between "b" and "h" and "c" and "e," achieving quality similar to the tighter 10% threshold. For lossy SPM, at all three error size thresholds, feature monitoring (Fig. 10 (e)-(g)) successfully retains the internal hole in "b" that is important for correct letter identification.

Summary: When used in lossy PM&S, in addition to reducing 30% of encoding time, enhanced prescreening can also effectively suppress 11 out of every 12 substitution errors. For lossy SPM, the proposed feature monitored shape unifying can successfully suppress more than half of all substitution errors. In comparing lossy PM&S with SPM, we found that SPM offers better reconstructed image quality (significantly fewer substitution errors) at similar or lower bit rates, but at the price of longer encoding time.

V. Conclusion

In this paper, we propose several ways to reduce the encoding time, memory consumption, and substitution errors for text image coding with JBIG2. We first look at page striping and propose dictionary updating procedures for the singleton exclusion and modified class dictionaries. With these dynamic updating techniques, page striping using 2 stripes gives 30% of savings in encoding time and 40% of savings in memory consumption, while suffering only 1.5% of bit rate penalty. More savings in time and memory can be obtained by using more stripes but with diminishing returns. We investigate two encoding trade-offs in page striping: the time *vs*. bit rate trade-off and the memory *vs*. bit rate trade-off. We propose an adaptive dictionary updating scheme that can achieve robust performance in both trade-offs when compared with other non-adaptive dictionary construction schemes. We then propose three speedup techniques for pattern matching. When combined together, these techniques can reduce coding time by up to 75% while incurring at most 1.7% coding loss. Compared with page striping, the proposed speedup techniques can better resolve the time *vs*. bit rate trade-off. However, page striping is still necessary for memory-limited applications. For lossy compression, in addition to bit rate, coding time, and memory usage, we also consider the number of substitution errors as the measure for the reconstructed image quality. For lossy PM&S, we use enhanced prescreening to reduce character substitutions by 12 times and save encoding time by 30% at the same time. For lossy SPM, we propose feature monitored shape

unifying to suppress 1/2 to 2/3 of the total substitution errors. Compared to lossy PM&S, lossy SPM using feature monitored shape unifying achieves better reconstructed image quality at similar or lower bit rate, but at the price of longer encoding time.

REFERENCES

[1] ISO/IEC JTC1/SC29/WG1 N1545. *JBIG2 Final Draft International Standard*, Dec. 1999.

[2] P. Howard, F. Kossentini, B. Martins, S. Forchhammer, W. Rucklidge, F. Ono. The Emerging JBIG2 Standard. *IEEE Trans. on Circuits and Systems for Video Technology*, pp. 838–48, Vol. 8, No. 5, Sept. 1998.

[3] CCITT. Standardization of Group 3 Facsimile Apparatus for Document Transmission. *CCITT Recommendation T.4*, 1980.

[4] CCITT. Facsimile Coding Schemes and Coding Control Functions for Group 4 Facsimile Apparatus. *CCITT Recommendation T.6*, 1984.

[5] CCITT. Progressive Bi-level Image Compression. *CCITT Recommendation T.82*, 1993.

[6] R. B. Arps and T. K. Truong. Comparison of international standards for lossless still image compression. *Proceedings of the IEEE*, pp. 889–99, vol. 82, No. 6, June 1994.

[7] R. Hunter and A. H. Robinson. International digital facsimile coding standards. *Proc. of IEEE*, pp. 854–67, Vol. 68, July 1980.

[8] Y. Ye and P. Cosman. Dictionary design for text image compression with JBIG2. *IEEE Trans. on Image Processing*, vol. 10, no. 6, pp. 818–828, June 2001.

[9] D. Tompkins and F. Kossentini. A Fast Segmentation Algorithm for Bi-level Image Compression Using JBIG2. *Proc. of the 1999 IEEE International Conference on Image Processing (ICIP)*, pp. 224–228, Kobe, Japan, Oct. 1999.

[10] R.N. Ascher and G. Nagy. Means for Achieving a High Degree of Compaction on Scan-digitized Printed Text. *IEEE Trans. on Computers*, pp. 1174–79, Nov. 1974.

[11] K. Mohiuddin. Pattern Matching with Application to Binary Image Compression. Ph.D. dissertation. Stanford University. 1982.

[12] P. Howard. Lossless and Lossy Compression of Text Images by Soft Pattern Matching. In J.A. Storer and M. Cohn, editors, *Proc. of the 1996 IEEE Data Compression Conference (DCC)*, pp. 210–19, Snowbird, Utah, March 1996.

[13] I.H. Witten, A. Moffat, and T.C. Bell. *Managing Gigabytes*. Morgan Kaufmann, 1999.

[14] C. Constantinescu and R. Arps. Fast residue coding for lossless textual image compression. *Proc. 1997 IEEE Data Compression Conference (DCC)*, pp. 397–406, Snowbird, Utah, March 1997.

[15] Y. Ye and P. Cosman. Fast and memory efficient JBIG2 encoder. *Proc. 2001 IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, Salt Lake City, Utah, May 2001.

[16] Y. Ye and P. Cosman. Feature monitored shape unifying for lossy SPM-JBIG2. *Proc. of the Sixth Intl. Symposium on Signal Processing and its Applications (ISSPA 2001)*, Kuala Lumpur, Malaysia, August 2001.

[17] ISO/IEC JTC1/SC29/WG1 N339. *Xerox Proposal for JBIG2 Coding*, June 1996.

[18] Y. Ye, D. Schilling, P. Cosman, and H. H. Koh. Symbol dictionary design for the JBIG2 standard. *Proc. of 2000 IEEE Data Compression Conference (DCC)*, pp. 33–42, Snowbird, Utah, March 2000.

[19] Y. Ye and P. Cosman. JBIG2 symbol dictionary design based on minimum spanning tree. *Proc. of the First Intl. Conf. on Image and Graphics (ICIG)*, pp. 54–57, Tianjin, China, Aug. 2000.

[20] R. Gould. Graph Theory. *The Benjamin/Cummings Publishing Co., Inc.*, Chapter 3, pp. 68-72, 1988.

[21] E. S. Askilsrud, R. M. Haralick and I. T. Phillips. A quick guide to UW English Document Image Database I, version 1.0. CD-ROM. Intelligent Systems Lab, University of Washington. August 1993.

[22] S. Inglis. Lossless document image compression. Ph.D. dissertation. Chap. 7. Univ. of Waikato, New Zealand, 1999.

[23] R. Habib. The early T.S. Eliot and western philosophy. Chap. 1, pp. 1-11, Cambridge University Press, 1999.

[24] W. K. Pratt, P. J. Capitant, W. Chen, E. R. Hamilton and R. H. Wallis. Combined symbol matching facsimile data compression system. *Proc. of the IEEE*, pp. 786–796, Vol. 68, No. 7, July 1980.

[25] B. Horn. Robot Vision, Chapter 3. MIT Press. 1986.

[26] A. Jain. Fundamentals of Digital Image Processing, Chapter 9. Prentice Hall. 1989.

[27] S. Inglis and I. H. Witten. Compression-based Template Matching. In J.A. Storer and M. Cohn, editors, *Proc. of the 1994 IEEE Data Compression Conference (DCC)*, pp. 106–15, Snowbird, Utah, March 1994.