Title
Speedup techniques for text image compression with JBIG2

Permalink
https://escholarship.org/uc/item/9q97r6tt

Authors
Ye, Y
Cosman, P

Publication Date
2007-11-01

Peer reviewed
Speedup Techniques for Text Image Compression with JBIG2

Yan Ye and Pamela Cosman
Electrical and Computer Engineering Department
University of California, San Diego
La Jolla, CA, 92093-0407
Emails: {y ye,pcosman}@code.ucsd.edu
Tel:(858)822-1250  FAX:(858)822-3426

Abstract

Pattern matching is the most time consuming process in text image compression with JBIG2. In this paper we propose three techniques to speed up the pattern matching process. By limiting the search range for matching symbols in the dictionary, the first technique saves 15% of encoding time with almost no bit rate penalty. By making early decisions about the pattern matching outcome, the second technique saves another 15% of encoding time with no coding loss. The third technique looks at enhanced prescreening using additional symbol features besides symbol size. Using certain topological features, enhanced prescreening can save up to 75% of encoding time with at most 1.6% of bit rate penalty.

1 Introduction

The JBIG2 standard [1, 2, 3] 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. A typical JBIG2 encoder first segments an image into different regions [4] and then uses different coding mechanisms for text and for halftones. In this paper, we are concerned with compressing text images with JBIG2. Text images consist mainly of repeated text characters and possibly some general graphics (e.g., line art). In JBIG2, the coding of text is based on pattern matching techniques [2, 3]. JBIG2 defines two modes for text compression: pattern matching and substitution (PM&S) [5] and soft pattern matching (SPM) [6].

On a typical page of text, there are many repeated characters. The bitmap of a character instance on the page is called a “symbol.” We can extract symbols from the input image using connected component analysis [7]. Rather than coding all the pixels of all the symbols 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, the index of its best matching symbol in the dictionary, and, in the SPM mode, possibly its actual bitmap which is refinement coded using its matching dictionary symbol [1, 2]. This type of bitmap coding, called refinement coding, is done by context-based arithmetic coding using a context drawn from both the best match bitmap from the dictionary, and the already coded part of the current bitmap [8, 6]. General graphic data not identified as text is encoded at the end using a basic bitmap coder such as specified by JBIG1 [9] or T.6 [10].

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 [9] and typical prediction for residue (TPR) as proposed in [11]. In this paper, we focus on reducing the encoding time spent on pattern matching. In our work we use the Hamming distance based matching criterion. We measure the percentage of different pixels between two symbols. For SPM-based JBIG2, even using our simple matching criterion, the time spent on pattern matching accounts for as much as 90% of the total encoding time. In this paper we propose three categories of speedup techniques that can significantly reduce the amount of pattern matching time while causing only a very small loss in coding efficiency.

This paper is organized as follows. In Section 2 we propose the three speedup techniques for pattern matching. In Section 3 we show experimental results on coding time saved and bit rate penalty incurred from using these speedup techniques. We conclude our paper in Section 4.

2 Speedup techniques for JBIG2 encoding

2.1 Limited dictionary symbol search

We proposed the modified-class (MC) dictionary design for the SPM-based JBIG2 in [13]. Experiments showed that the MC dictionary achieves competitive coding performance with relatively low complexity. The design of an MC dictionary consists of two steps. At the first step,
we point each symbol to its closest match among all other symbols; we only draw a pointer between a symbol and its best match if the mismatch between them is below the preset threshold. This way the entire symbol set is segmented into small connected graphs, each of which is called a class. We then choose one representative for each class as the symbol with the lowest average mismatch within the class. All class representatives go into the dictionary. The second design step decides the reference relationships among all dictionary symbols, i.e., class representatives. This is done by calculating the matching graphs for all dictionary symbols and forming minimum spanning trees (MSTs) out of these graphs [12, 13].

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 symbol $S$ and symbol $R$ are similar, and that symbol $R$ is similar (to different degrees) to the symbols in tree $T$. Therefore, when symbol $S$ searches for its best match in the dictionary, we only search among all the symbols that belong to MST $T$. To do this, for each symbol on the page, we maintain a tree-ID value that specifies the MST in which this symbol's representative belongs. Hence, in the previous example, symbols $S$ and $R$ and all other symbols in the MST $T$ will have the same tree-ID. When the current symbol is matched with the dictionary, it only searches among those dictionary symbols that have the same tree-ID. This significantly reduces the number of dictionary symbols with which the current symbol is matched. Whether this limited search algorithm will suffer significant bit rate penalty depends on how many symbols and their best dictionary matches actually belong to the same MST. Section 3 shows that this limited search algorithm can save encoding time at almost no coding loss.

### 2.2 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 score 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. On the other hand, comparing the two integer mismatch scores also takes time. Therefore, we do the integer comparison of mismatch scores only once for each row of pixels in the bitmap. At the end of each row, the current accumulated mismatch is checked; if it exceeds the previous lowest, the pattern matching process terminates.

### 2.3 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 ","), obviously do not match. The original SPM system as proposed in [6] prescreens symbols using size; only symbols with similar sizes (defined as not more than 2 pixels different in either dimension) are given to the pattern matcher which computes their mismatch score. 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 [7, 14]. 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 [7]. 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 [16]. Prescreening by these two features can effectively prevent these symbol pairs from being handed over to the pattern matcher (see Figure 1).

Another useful feature for prescreening is introduced in [7]. We call it 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 threshold, which is preset to 3 pixels in our implementation. 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.

### 3 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.

Our experiments use a set of twelve test images, two
Table 1. Using the proposed three speedup techniques in SPM JBIG2.

<table>
<thead>
<tr>
<th></th>
<th>total time sec</th>
<th>match time sec</th>
<th>coded size bytes</th>
<th>% gain</th>
<th>% gain</th>
<th>% loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>NONE</td>
<td>99.44</td>
<td>-</td>
<td>36,738</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TID</td>
<td>85.82</td>
<td>14</td>
<td>36,742</td>
<td>15</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>EJO</td>
<td>85.12</td>
<td>14</td>
<td>36,738</td>
<td>16</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>TID+EJO</td>
<td>71.18</td>
<td>28</td>
<td>36,742</td>
<td>31</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>PRESCRN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S+Q</td>
<td>31.95</td>
<td>68</td>
<td>37,128</td>
<td>1.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S+H+C</td>
<td>61.85</td>
<td>38</td>
<td>36,938</td>
<td>0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S+Q+H+C</td>
<td>28.70</td>
<td>71</td>
<td>37,342</td>
<td>1.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>26.20</td>
<td>74</td>
<td>37,359</td>
<td>1.7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2. Using the proposed three speedup techniques in PM&S JBIG2.

<table>
<thead>
<tr>
<th></th>
<th>total time sec</th>
<th>match time sec</th>
<th>coded size bytes</th>
<th>% gain</th>
<th>% loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>NONE</td>
<td>24.05</td>
<td>-</td>
<td>41,404</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>EJO</td>
<td>22.81</td>
<td>5</td>
<td>41,404</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>PRESCRN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S+Q</td>
<td>16.28</td>
<td>32</td>
<td>41,730</td>
<td>0.8</td>
<td></td>
</tr>
<tr>
<td>S+H+C</td>
<td>19.48</td>
<td>19</td>
<td>41,566</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>S+Q+H+C</td>
<td>16.07</td>
<td>33</td>
<td>41,925</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>16.04</td>
<td>33</td>
<td>41,925</td>
<td>1.3</td>
<td></td>
</tr>
</tbody>
</table>

Table 3. Prescreening pass rates when different features are used.

<table>
<thead>
<tr>
<th>features</th>
<th>S</th>
<th>S+Q</th>
<th>S+H+C</th>
<th>S+Q+H+C</th>
<th>% passes</th>
</tr>
</thead>
<tbody>
<tr>
<td>% passes</td>
<td>19.7</td>
<td>4.7</td>
<td>11.8</td>
<td>4.0</td>
<td></td>
</tr>
</tbody>
</table>

Over a channel of fixed bandwidth, transmission of a bigger file takes longer time. Some applications, e.g., sending an international fax, favor the shortest channel time possible. For such applications, achieving the best compression is the most important, even if it takes some extra encoding time. Other applications, especially real-time applications, can only tolerate a small delay between the sender and receiver. For these applications, the goal is to achieve the best compression within a short encoding time. However, better compression usually requires longer encoding time. Figure 2 shows the trade-off between coding time and coding efficiency by plotting compression achieved as a function of total encoding time spent. The SPM system is shown as an example. Similar observations are made for the PM&S system. From Figure 2 we clearly see the trade-off between these two figures of merit. The lower boundary of the convex hull, as shown by the dashed line segments, represents the best trade-off that our techniques can achieve between coding time and efficiency. Although the H and C features are not as efficient as the Q feature in terms of this trade-off (the PRE(S+Q) marker lies on the lower boundary but the PRE(S+H+C) marker does not), in [17] we showed that the H and C features can help effectively control the reconstructed image quality in
Figure 1. Examples of similar bitmaps that have different features. Bitmaps “b” and “h” differ in the number of holes; and bitmaps “i” and “l” differ in the number of connected components.
lossy coding by reducing the number of character substitutions.

4 Conclusion

In this paper we propose three techniques to speed up the pattern matching process in text image compression with JBIG2. Experiments show that the limited dictionary symbol search technique and the early jump-out technique can each bring about 15% of savings in encoding time without loss in coding efficiency. Depending on the specific features used, the enhanced prescreening technique can save up to 75% of encoding time while only suffering a small bit rate penalty of at most 1.6%. These speedup techniques are effective for both SPM-based and PM&SB-based JBIG2.

Acknowledgements

This research was supported by the Center for Wireless Communications at UCSD and by the CoRe program of the State of California.

References


