Context-Dependent Multilevel Pattern Matching for Lossless Image Compression*

Yunwei Jia and En-hui Yang†

Abstract — The performance of the multilevel pattern matching (MPM) code for lossless image compression is first analyzed. It is shown that the worst-case redundancy of the MPM code against all finite 2D context arithmetic codes is $O(1/\sqrt{\log n})$, where n is the number of pixels in the image to be compressed. This result is in contrast to the redundancy of $O(1/\log n)$ in the case of 1D data and is caused by the so-called 2D boundary effect. To alleviate the 2D boundary effect, we then extend the MPM code to the case of context modeling, yielding a context-dependent MPM code. Our experimental results show that the context-dependent MPM code, with a simple context model, significantly outperforms the original MPM code on a wide range of bi-level images. Comparing with JBIG, the context-dependent MPM code outperforms JBIG in progressive coding mode, and is comparable with JBIG in sequential coding mode.

I Introduction

Recently, Kieffer et al.[1] proposed a lossless data compression code called the multilevel pattern matching code (MPM code). The MPM code is universal in the sense that it can achieve asymptotically the entropy rate of any stationary source. The authors of [1] also proved that the MPM code has an $O(1/\log n)$ worst-case redundancy both relative to any finite-state arithmetic code and relative to any finite-state source. In this study, we shall first analyze the performance of the MPM code for lossless image compression. Let us begin with a brief review of the MPM code.

Let $x = x_1 x_2 \cdots x_n$ be a sequence with a finite alphabet $\mathcal{A}$. At the heart of the MPM code is a multilevel representation $(T_0, T_1, \cdots, T_I)$ of the input sequence x, where the parameter $I \sim \log\log n$ is positive integer. At each level $i (0 \le i \le I)$, T_i is a sequence of tokens each of which represents a substring of x of length r^i, where $r \ge 2$ is a fixed positive integer. T_0 is constructed in the following way. Fix a set $\mathcal{T} = \{t_0, t_1, \cdots\}$ disjoint from $\mathcal{A}$. First, partition x into non-overlapping substrings of length r^I. (For simplicity and without losing the essentials of the MPM code, we assume the input sequence length n is a multiple of r^I.) Define this partition as $S_0(x)$. Next, we replace each distinct substring in $S_0(x)$ with a token in $\mathcal{T}$, according to its order of appearance in $S_0(x)$, i.e., the first distinct substring in $S_0(x)$ is replaced with t_0, the second with t_1, and so on. The resulting token sequence is T_0. For $1 \le i \le I$, T_i is constructed recursively in the following manner. Let $\mathcal{T}^i = \{t_0, t_1, \cdots, t_{|\mathcal{T}^i|-1}\}$ be the set of distinct tokens appearing in $T_i, 0 \le i \le I-1$. The token sequence $t_0 t_1 \cdots t_{|\mathcal{T}^i|-1}$ represents a sequence of length $r^{I-i}|\mathcal{T}^i|$ from $\mathcal{A}$. Partition the sequence from $\mathcal{A}$ represented by $t_0 t_1 \cdots t_{|\mathcal{T}^i|-1}$ into non-overlapping substrings of length r^{I-i-1}, and denote this partition as $S_{i+1}(x)$. We then replace each distinct substring in $S_{i+1}(x)$ with a token in $\mathcal{T}$, in the same way as that in constructing T_0. The resulting token sequence is T_i. As for the bottom level, T_I is simply the same sequence from $\mathcal{A}$ represented by $S_I(x)$. The sequence $(T_0, T_1, \cdots, T_I)$ is called the multilevel representation of x, from which x can be fully reconstructed. Thus, to compress x, we just need to compress this multilevel representation. This task is accomplished by encoding each $T_i (0 \le i \le I)$ separately using an adaptive arithmetic code. (T_I may be coded differently. See [1] for details.) We also need to encode the input sequence length n.

In [1], the authors also mentioned a version of the MPM code, called QUAD code, for lossless image compression. To compress an image X, the QUAD code first transforms X into a one-dimensional (1D) sequence through a scanning method called quadrisection scanning. One then applies the MPM code to this 1D sequence with $r = 4$. Their experiments on compression of bi-level archival images of size 512×512 show that, though the QUAD code is competitive with JBIG, there is a obvious gap between the compression rates of the QUAD code and those of JBIG. For the eight tested images in [1], the QUAD code has an average rate of 0.2132 bits/pixel, and that of JBIG in progressive coding mode is 0.1786 bits/pixel.

To improve the performance of the MPM code for lossless image compression, we will extend the MPM code to the case of context modeling in this study. We first analyze the compression performance of the MPM code in lossless image compression in next section. Particularly, we compare the MPM code with template-based arithmetic codes (such as JBIG) in lossless image compression. The analysis will shed some light on why we should use con-

*This work was supported in part by the Natural Sciences and Engineering Research Council of Canada under Grant RGPIN203035-98, by the Communications and Information Technology Ontario, by the Premier's Research Excellence Award, and by the Canada Research Chairs Program

†The authors are with the Dept. of Electrical and Computer Engineering, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1. E-mails: yjia@bbcr.uwaterloo.ca; ehyang@bbcr.uwaterloo.ca.

text modeling and how to design a context model in the MPM code. These two topics, together with performance analysis of the context-dependent MPM code, will be discussed in Section III. Experimental results are presented in the last section.

II 2D MPM Code — Without Context Models

In this section, we first present a version of the MPM code for lossless image compression, called two-dimensional (2D) MPM code. This code is different from the QUAD code. We then analyze its compression performance.

II-A Algorithm description

Throughout this subsection, we fix $r \geq 2$ and a nonnegative integer I. Let X be an image with a finite alphabet $\mathcal{A}$ and of size $N \times M$, where N is the number of rows and M is the number of columns of X. Let $n = NM$ be the number of pixels in X. Later in this paper we will interchangeably use X, $X_{N\times M}$ or X_n to represent this image. For simplicity we assume that both N and M are multiples of r^I. Fix a token set $\mathcal{T} = \{t_0, t_1, \cdots\}$ disjoint from $\mathcal{A}$. Let \$ be a symbol not in $\mathcal{A} \cup \mathcal{T}$. There are two steps to compress X using the 2D MPM code: construction of a multilevel representation of X and encoding of the multilevel representation.

1) Construction of a multilevel representation of X: First, we partition X into non-overlapping blocks of size $r^I \times r^I$. Fix a block scanning method. (This scanning method can be arbitrary; in contrast in the QUAD code only the quadrisection scanning is used.) Read the blocks according to the scanning method, and denote the obtained sequence as $S_0(x) = X^1X^2\cdots X^{|S_0(X)|}$, where $X^i (1 \leq i \leq |S_0(X)|)$ represents an $r^I \times r^I$ block of pixels of X. Replace each distinct block in $S_0(X)$ with a token in $\mathcal{T}$, t_0 for the first distinct block, t_1 for the second, and so on. Then, in the resulting token sequence, replace each distinct token with \$ at its first appearance, and do not touch its later appearances if any. Denote the final sequence as $T_0 = (u_1, \cdots, u_{|T_0|})$, where $u_i \in \{\$, t_0, t_1, \cdots, t_{\langle T_0\rangle - 1}\}$ with $\langle T_0\rangle$ being the number of distinct tokens in T_0. The other sequences in the multilevel representation of X, $T_1, \cdots, T_I$, are obtained recursively in the following way. For any sequence z, let $\eta(a|z)$ be the number of occurrences of the symbol a in z. Let $\mathcal{T}^i = \{t_0, \cdots, t_{\eta(\$|T_i)-1}\}$, where $0 \leq i \leq I-1$ and $t_j (0 \leq j \leq \eta(\$|T_i) - 1)$ represents an $r^{I-i} \times r^{I-i}$ block of pixels. Partition the $r^{I-i} \times r^{I-i}$ block represented by $t_j \in \mathcal{T}^i$ into r^2 smaller blocks, each of which is of size $r^{I-i-1} \times r^{I-i-1}$. Arrange these smaller blocks according to the scanning method. Then concatenate all these smaller blocks corresponding to $t_0, t_1, ..., t_{\eta(\$|T_i)-1}$ in the indicated order. Let $S_{i+1}(X)$ represent the resulting sequence. Now, as we did to $S_0(X)$, we replace each distinct block in $S_{i+1}(X)$ with a token in $\mathcal{T}$, and replace each distinct token with \$ at its first appearance. The resulting sequence is T_{i+1}. As for the bottom level, T_I is the same sequence from $\mathcal{A}$ represented by $S_I(X)$.

2) Encoding of the multilevel representation of X: Let $(T_0, T_1, \cdots, T_I)$ be the multilevel representation of X obtained from the previous step. Let $T_i = u_1 \cdots u_{|T_i|}, 0 \leq i \leq I-1$. From the previous step, we see that $u_1 = \$$ and $u_{k+1} \in \{\$, t_0, t_1, \cdots, t_{\eta(\$|u_1^k)-1}\}, 1 \leq k \leq |T_i| - 1$. Thus, the following is a valid probability distribution of u_{k+1}:

$$p(u_{k+1}|u_1^k) = \begin{cases} \frac{\eta(\$|u_1^k)}{|u_1^k|+\eta(\$|u_1^k)}, u_{k+1} = \$, \\ \frac{\eta(t_j|u_1^k)+1}{|u_1^k|+\eta(\$|u_1^k)}, u_{k+1} = t_j, j = 0, 1, \cdots, \eta(\$|u_1^k) - 1. \end{cases}$$

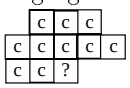
Since each of the probabilities in this distribution is positive, we can use an adaptive arithmetic code to encode/decode $T_i, 0 \leq i \leq I-1$, using this probability distribution. As for T_I, since all symbols in T_I are from $\mathcal{A}$, we can use the following probability distribution to encode/decode it by an adaptive arithmetic code:

$$p(u_{k+1}|u_1^k) = \frac{\eta(u_{k+1}|u_1^k) + 1}{|\mathcal{A}| + |u_1^k|}, u_{k+1} \in \mathcal{A}.$$

Of course, we need to encode the size of X before encoding $T_0, T_1, \cdots, T_I$. This can be done using the simple coding scheme E_1 described in [1]. Note that from the size of X we can determine $|T_0|$. From $\eta(\$|T_i)(0 \leq i \leq I-1)$ we can determine $|T_{i+1}|$. Thus we need not to send additional information about $|T_0|, |T_1|, \cdots |T_I|$ to the decoder. (These lengths are needed in the arithmetic code for T_i as a termination condition.)

II-B Algorithm analysis

In this subsection, we compare the 2D MPM code with template-based arithmetic codes for lossless image compression. For this purpose, we need the following definitions. A *template* is a geometric pattern of pixels which define a neighborhood around a pixel to be coded. The pixels in a template take on values when the template is aligned to a particular part of an image. The values of the pixels in the template define a context for the pixel to be coded. The following figure shows one of the templates used in JBIG[2].



The pixel denoted by "?" corresponds to the pixel to be coded and is not part of the template. The pixels denoted by "c" correspond to pixels in the template, and shall be used to determine the context for the pixel "?". Let T be a template. For a pixel $x \in X$, we use T_x to denote the set of pixels in its template, and use $T(x)$ to denote its context defined by T_x. If any of the pixels in T_x lies outside the boundary of X, we say that x has an *incomplete* template. Denote the set of all pixels in X with incomplete templates as $\Lambda_T(X)$. Intuitively, $\Lambda_T(X)$ contains the boundary pixels of X. For example, for the template T and the image X shown in Figure 1, $\Lambda_T(X) = \{1, 2, 3, 4, 5, 9, 13\}$.

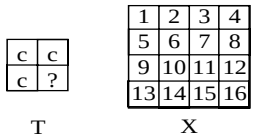


Figure 1: A template and an image

Let s be a fixed positive integer. Let $\mathcal{I}(s) = \{1, 2, \cdots, s\}$. Let $\mathcal{T}_s(\mathcal{A})$ be the family of all templates T which induce s contexts. (For example in Figure 1, if each of the three "c" pixels takes on values 0 and 1, then the template T induces 8 contexts.) Fix the mapping from $T(x)$ into $\mathcal{I}(s)$. Let $\mathcal{P}_s(\mathcal{A})$ be the family of all functions p from $\mathcal{I}(s) \times \mathcal{A}$ into $[0, 1]$ such that $\sum_{a \in \mathcal{A}} p(a|u) = 1$ for any $u \in \mathcal{I}(s)$, where $p(a|u)$ denotes the value of the function p at (u, a).

Definition 1 *Let X_n be an image. The s-context unnormalized empirical entropy of X_n is the nonnegative real number $H^s(X_n)$ defined by*

$$H^s(X_n) = \inf_{p \in \mathcal{P}_s(\mathcal{A})} \inf_{T \in \mathcal{T}_s(\mathcal{A})} \min_{T(x) \in \mathcal{I}(s),\ x \in \Lambda_T(X_n)} - \log \prod_{i=1}^{n} p(x_i|T(x_i)). \quad (1)$$

The quantities $H^s(X_n), s = 1, 2, \cdots$, represent the shortest codeword lengths assigned to X_n by any template-based arithmetic code with s contexts. The following theorem bounds the codeword length assigned to X_n by the 2D MPM code in terms of $H^s(X_n)$.

Theorem 1 *Let s be any positive integer. Let $L(X_n|r)$ be the codeword length assigned to X_n by the 2D MPM code when the parameter I is chosen as $O(\log\log n)$. Then*

$$\max_{X_n} \frac{1}{n} [L(X_n|r) - H^s(X_n)] = O(\frac{1}{\sqrt{\log n}}). \quad (2)$$

Outline of proof: First, we show that

$$\sum_{i=0}^{I} |T_i| = O(\frac{n}{\log n}), \quad (3)$$

where $\{T_i, i = 0, 1, \cdots, I\}$ is the multilevel representation of X furnished by the 2D MPM code. Then we show that

$$L(X_n|r) \le \sum_{i=0}^{I} H_0(\tilde{T}_i) + \sum_{i=0}^{I} 3|T_i| + 2\log n + 2|\mathcal{A}|, \quad (4)$$

where $\tilde{T}_i$ is the sequence obtained from T_i by striking all \$'s, and $H_0(\tilde{T}_i)$ is the zeroth-order entropy of $\tilde{T}_i$. (See [1] for the definition of $H_0(\cdot)$). Next, we show that

$$\sum_{i=0}^{I} H_0(\tilde{T}_i) \le H^s(X_n) + \sum_{i=0}^{I} |\Lambda_T(X_{r^{I-i} \times r^{I-i}})| \cdot |\tilde{T}_i| \log s, \quad (5)$$

where $\Lambda_T(X_{r^{I-i} \times r^{I-i}})$ is the set of pixels in an $r^{I-i} \times r^{I-i}$ block with incomplete templates. For a $k \times k$ block $X_{k \times k}$ and a template with finite contexts, $|\Lambda_T(X_{k \times k})|$ is $O(k)$. (Intuitively, that is because the number of boundary pixels in a $k \times k$ block is $O(k)$.) From (3),(4) and (5) we obtain (2).

Theorem 1 tells us that the worst-case redundancy of the 2D MPM code relative to any template-based arithmetic code with finite contexts is $O(1/\sqrt{\log n})$, as a function of the image size n. Recall that in 1D case, the worst-case redundancy of the MPM code relative to any finite-state arithmetic code is $O(1/\log n)$. Why do we have a worse redundancy bound in the 2D case than in the 1D case? How do we improve it? These two questions will be answered in next section.

III 2D MPM Code — With Context Models

To answer the two questions mentioned at the end of the previous section, we need to understand how the redundancy is generated in the MPM code. In the 2D case, one compares the 2D MPM code with template-based arithmetic codes. Let T stand for the template used in the arithmetic codes. In the 2D MPM code, an image X is partitioned into 2D blocks of different sizes. Within each block B, the pixels in $\Lambda_T(B)$ are "not fully conditionally coded" because their templates are incomplete. If B consists of $|B|$ pixels, $|\Lambda_T(B)|$ is $O(\sqrt{|B|})$, which is dependent on the size of the block. This is different from the 1D case. In the MPM code for 1D data, an input sequence is partitioned into substrings of different lengths, and the number of "not fully conditionally coded" symbols in each substring is constant and does not depend on the length of the substring. It is this difference that results in the term $|\Lambda_T(X_{r^{I-i} \times r^{I-i}})|$ in (5), and consequently an $O(1/\sqrt{\log n})$ redundancy for the 2D MPM code.

After understanding the cause of the redundancy of the 2D MPM code, the solution to a better redundancy is clear. That is, we need to conditionally encode the boundary pixels within each block in the multilevel representation of X. We achieve this by extending the 2D MPM code to the case of context modeling, yielding a context-dependent MPM code. First, let us define the context of a block of pixels. Let $Y_{a \times b}$ be a block of pixels. Let T be a template and $\Lambda_T(Y)$ be the set of pixels in Y with incomplete templates. Define $C_T(Y) = \{x : x \in \cup_{y \in \Lambda_T(Y)} T_y, x \notin Y\}$. We call $C_T(Y)$ (or more precisely, the geometric pattern of $C_T(Y)$) the context of Y. For example, using the template T and for the block Y shown in Figure 2, we have $\Lambda_T(Y) = \{1, 2, 3, 4, 5, 9, 13\}$ and the context of Y is $C_T(Y) = \{c1, c2, \cdots, c9\}$.

Similar to the 2D MPM code without context models, there are two steps in the context-dependent MPM code to compress an image X: construction of a mul-

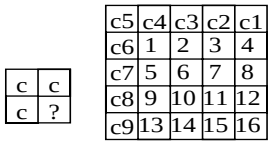


Figure 2: Context of a block

tilevel representation of X and encoding of this representation. The difference is that, in both steps in the context-dependent MPM code, we take into account context information. First, X is partitioned into non-overlapping blocks of size $r^I \times r^I$. Denote this partition as $S_0(x) = X^1 X^2 \cdots X^{|S_0(X)|}$. For each $X^i \in S_0(X)$ let $C_T(X^i)$ be its context. We shall call the sequence $(C_T(X^1), C_T(X^2), \cdots, C_T(X^{|S_0(X)|}))$ the context sequence of $S_0(x)$, and use $C = \{c_1, c_2, \cdots, c_k\}$ to denote the set of distinct contexts in it. T_I is constructed and then encoded conditioning on C, as described next. For each context $c \in C$, let $S_0^c(x) = X^{i_1^c} X^{i_2^c} \cdots X^{i_{k_c}^c}$, where $1 \le i_1^c < i_2^c < \cdots < i_{k_c}^c \le |S_0(X)|$ and all the blocks $X^{i_j^c} \in S_0^c(x)$ share a common context c. Performing the same operations to $S_0^c(x)$ as those to $S_0(x)$ in the the 2D MPM code without context models described in Section II-A, we obtain a sequence T_0^c. Combining these sequences $T_0^c, c \in C$, according to their original ordering in the context sequence of $S_0(X)$, we obtain T_0. Then, to encode T_0, we use $|C|$ separate adaptive arithmetic codes, one for each distinct context c in C and for the sequence T_0^c. Of course, these $|C|$ adaptive arithmetic codes work in an interleaving manner according to the ordering in the context sequence of $S_0(X)$. $T_1, T_2, \cdots, T_I$ are constructed and encoded similarly.

Next, we analyze the performance of the context-dependent MPM code for lossless image compression. Fix a template T. For an image X_n, let $L(X_n|r,T)$ be the codeword length assigned to X_n by the context-dependent MPM code when the parameter I is chosen as $O(\log\log n)$. Define a quantity

$$H^s(X_n|T) = \inf_{p \in \mathcal{P}_s(\mathcal{A})} \min_{T(x) \in \mathcal{I}(s),\ x \in \Lambda_T(X_n)} -\log \prod_{i=1}^{n} p(x_i|T(x_i))$$

where $p, \mathcal{P}_s(\mathcal{A}), T(x)$ and $\mathcal{I}(s)$ are defined the same as in (1). $H^s(X_n|T), s = 1, 2, \cdots$, represent the shortest codeword lengths assigned to X_n by any arithmetic codes using T as template. The following theorem compares $L(X_n|r,T)$ with $H^s(X_n|T)$.

Theorem 2 *Let s be a positive integer and T is a template. Then*

$$\max_{X_n} \frac{1}{n} [L(X_n|r,T) - H^s(X_n|T)] = O(\frac{1}{\log n}). \quad (6)$$

The proof of Theorem 2 is similar to that of Theorem 1, and is omitted due to page limit. Theorem 2 tells us that, when using a template T, the context-dependent MPM code has an $O(1/\log n)$ worst-case redundancy relative to any arithmetic codes using the same template. Comparing with the $O(1/\sqrt{\log n})$ redundancy in the 2D MPM code without context models, we see that the context modeling we introduced to the 2D MPM code does improve the compression performance. The experimental results presented in next section also confirm this. To further investigate the effect of context modeling, we are currently working on the comparison of $L(X_n|r,T)$ with $L(X_n|r)$.

Table 1: Compression rates in bits/pixel

Image	2D MPM	MPM(con)	JBIG(p)	JBIG(s)
lena	0.197	0.148	0.149	0.135
baboon	0.598	0.498	0.532	0.488
sailboat	0.210	0.146	0.156	0.143
peppers	0.174	0.120	0.121	0.107
barbara	0.304	0.245	0.249	0.201
tiffany	0.035	0.022	0.028	0.023
airplane	0.033	0.030	0.033	0.025
airport	0.253	0.203	0.219	0.208
man	0.211	0.161	0.167	0.154

IV Experimental results

In Table 1 we report some compression results on bi-level images. The first six images in the table are of size 512×512. The last three images are of size 1024×1024. We list the compression rates in bits/pixel of the 2D MPM code without context models, the context-dependent MPM code, and JBIG in progressive coding mode and sequential coding mode, at columns 2 to 5 in the table. In the context-dependent MPM code, we use the simple template T shown in Figure 2. As can be seen from the table, the context-dependent MPM code outperforms the 2D MPM code without context models significantly for all the tested images. Compared with JBIG, the context-dependent MPM code outperforms JBIG in progressive coding mode for all the tested images, and is comparable to JBIG in sequential coding mode. One can expect a better compression performance if a more complicated context model is used in the context-dependent MPM code. About complexity issue, it can be shown that the context-dependent MPM code has time complexity $O(n)$ and storage complexity $O(n)$, where n is the number of pixels in an input image to be compressed.

References

[1] J. C. Kieffer, E.-H. Yang, G. Nelson, and P. Cosman, "Universal lossless compression via multilevel pattern matching," *IEEE Trans. Inform. Theory*, Vol. 46, pp. 1227-1245, July, 2000.

[2] ITU-U Recommendation T.82. Information technology – coded representation of picture and audio information - progressive bi-level image compression. 1993.