

On the Use of Context Tree for Binary Image Compression

Pasi Fränti and Eugene Ageenko

*Department of Computer Science, University of Joensuu
Box 111, FIN-80101 Joensuu, FINLAND
Email: franti,ageenko@cs.joensuu.fi*

Abstract

We consider the use of a static context tree for binary image compression. The contexts are stored in the leaves of a variable-depth binary tree. The tree structure itself is fully static and optimized off-line for a training image. The structure of the tree is similar for different images of the same type. The benefit from optimizing the tree for each input image separately is usually outweighed by the overhead required from storing the tree structure. The static approach is therefore applicable in most situations as the compression can be performed much faster and during a single pass over the image.

1. Introduction

Lossless compression of bi-level images has been well studied in literature and several standards already exist [1]. JBIG in the sequential mode codes the image pixel by pixel in raster scan order using *context-based probability modeling* and *arithmetic coding* [2]. It achieves a compression ratio of 19.7 for the CCITT image set. Remarkable improvement to this has been achieved only by specializing to some known image types (e.g. text images) and exploiting global dependencies, or by lossy compression.

The next standard, JBIG-2, will include pattern matching technique to extract symbols from text images. The compressed file consists of bitmaps of the library symbols coded by a JBIG-style compressor, location of the extracted marks as offsets, and a pixelwise coding of the matched symbols using two-layer template [3, 4]. The core of the new standard is still the context-based coding of the individual pixels, as in JBIG.

JBIG uses a static, 10-pixel context template. Despite the high number of contexts, only a small fraction of all 1024 contexts are really important. Typically 50 % of the code bits originate from about ten most important contexts only. Moffat, on the other hand, has demonstrated the potential of even larger context templates by applying template sizes up to 24 pixels [5]. A larger template size

gives theoretically better prediction but at the cost of higher learning cost. The number of contexts grows exponentially as the function of the template size. Thus, the learning cost grows faster than the benefit in compression and exceeds it for template size of 14 pixels.

We study the use of a variable-size context template, where the number of context pixels depends on the combination of the neighboring pixel values. We store the contexts in a binary tree of variable depth, denoted as *context tree*. The context selection is deterministic and only the leaves of the tree are used as contexts. The total number of the contexts is kept of the same order than that of JBIG to keep the learning cost tolerable. The tree structure is fully static and the compression phase has the same order of the complexity as that of JBIG.

For a set of newspaper images, the proposed method achieves an improvement of about 14 % in comparison to JBIG when trained with another image of similar type. An improvement of about 7 % was obtained when the same context tree was applied for the CCITT test images. Most of the improvement originates from a selective context expansion. Larger context templates are utilized without overwhelming the learning cost. Furthermore, the proposed method is essentially the same as JBIG, only the context selection is different. The ideas should therefore apply to JBIG-2, as well.

2. JBIG

In *JBIG* the image is coded pixel by pixel in raster scan order using *context-based probability model* and *arithmetic coding*. The combination of already coded neighboring pixels defines the context. In each context the probability distribution of the black and white pixels are adaptively determined. The current pixel is then coded by *QM-coder*, the binary arithmetic coder adopted in JBIG. The probability estimation and update are performed using the state automaton of *QM-coder*. The standard includes also a progressive mode, which is not discussed here.

By default, JBIG uses the 10-pixel context template shown in Fig. 1. This is referred here as JBIG₁₀. The context is determined by combining index out of the neighboring pixel values and accessing to the model using a look-up table. With 10 pixel template there are $2^{10} = 1024$ different contexts in total.

Despite the high number of context in JBIG, only a small fraction of all 1024 contexts are really important. For example, in the case of *CCITT-5* test image, 50 % of the code bits originate from the nine most important contexts only. These contexts and their statistics are shown in Fig. 2. Furthermore, 99 % of the code bits originate from 183 contexts, and 429 out of the 1024 contexts are never used at all.

The context size is a trade-off between prediction accuracy and learning cost. The larger the context, the more accurate probability model it is possible to obtain. However, with a large template the adaptation to the image statistics takes longer and thus the higher is the coding deficiency in the early stage of compression. The optimal context size for the *CCITT* test images seems to be 14 according to our experiments [6]. Similar observation was also made for a different set of test images in [5].

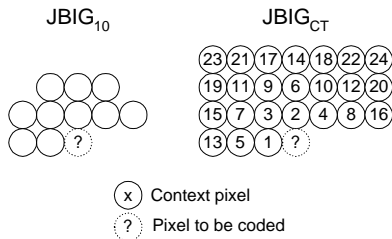


Figure 1. The default three-line 10-pixel context template of JBIG (left), and the 24-pixel *ordered neighborhood* for the context tree (right).

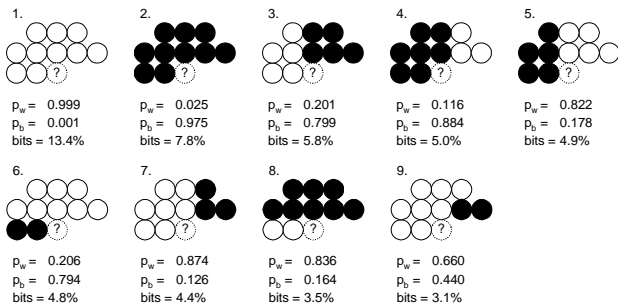


Figure 2. The most important contexts of JBIG in the case of *CCITT-5* image at 200 dpi.

3. Tree-structured context model

The compression method, referred as *Context tree*, is essentially the same as JBIG. Only the context selection is made using a context tree instead of a fixed size template. Each node in the tree represents a single context. The two children of a context correspond to the parent context augmented by one more pixel. The nodes at a level k correspond to a context model of order k .

3.1 Different ways for using the context tree

There are several alternative ways for utilizing the context tree. Martins and Forchhammer [7] applies a full k -level tree that corresponds to a complete collection of all possible contexts up to the order k . The context is dynamically chosen along the path from root to leaf using *predictive minimum description length principle* (PMDL) [8]. Only one context is used at a time but statistics are updated for all contexts in the path. The two-level context model of [5] is a special case of this approach. An improvement of about 8 % was reported in [7] but at the cost of high running time because several contexts must be considered for each input pixel to be compressed.

A better approach is to prune the tree by removing all unnecessary branches, and to use only the leaves of the variable-depth tree. A semi-adaptive approach was considered in [7] by optimizing the tree for each image separately. The tree structure must also be stored. The on-line construction of the tree, however, makes the compression an order of magnitude slower than JBIG.

We take slightly different approach by using static context tree. This is possible because different trees optimized for different images of similar type do not remarkably differ from each other. According to our experiments, the compression improvement due to the semi-adaptive approach is usually less than the overhead required from storing the tree. In most cases, it is therefore better to use the static approach because the tree can be generated off-line. In this way the compression phase remains essentially as fast and simple as that of JBIG.

3.2 Construction of the tree

The tree is constructed using a top-down splitting algorithm as described in Fig. 3. The process starts by constructing the models for all contexts at a predefined *minimum* level k_{MIN} . The next level contexts are tentatively constructed and compared to their parent contexts. Statistics are calculated for both children contexts. If compression gain is expected due to the splitting, the parent node is permanently split. The procedure is then repeated for each new context at the

next level. The process continues until a predefined maximum level k_{MAX} is reached, or when no new nodes were created during the process of a single level.

```

ConstructContextTree( $k_{MIN}, k_{MAX}$ )
{
   $k \leftarrow k_{MIN}$ .
  GenerateTreeStructure( $k$ ).
  CollectStatistics( $k$ ).
  REPEAT
     $k \leftarrow k + 1$ .
    ConstructLevel( $k$ ).
    CollectStatistics( $k$ ).
    PruneLevel( $k$ ).
  UNTIL  $k = k_{MAX}$  OR no new nodes were created.
  StoreTreeStructure.
}

```

Figure 3. Algorithm for top-down construction of the tree.

3.3 Splitting criterion

In principle, the splitting should be done whenever saving is expected in the code length. The saving is calculated as:

$$Saving(C, C_0, C_1) = l(C) - l(C_0) - l(C_1), \quad (1)$$

where C is the parent context and C_0 and C_1 are the two children nodes. The code length l denotes the total number of output bits from the pixels coded using the context.

We use a Bayesian sequential estimator where the probability of a pixel is calculated on the basis of the observed frequencies:

$$p_0^i(C) = \frac{n_0^i(C) + \delta}{n_0^i(C) + n_1^i(C) + 2\delta}, \quad (2)$$

$$p_1^i(C) = 1 - p_0^i(C), \quad (3)$$

where n_1^i and n_0^i are the time-dependent frequencies, and p_1^i and p_0^i are probabilities for background (0) and foreground color (1) respectively, and $\delta = 0.45$ as in JBIG. The code length is calculated by summing up the entropy estimates of pixels as they occur in the image:

$$l(C) = \sum_i \log(p^i(C)) \quad (4)$$

It can be efficiently calculated as the Bernoulli code length from only the final counts n_0 and n_1 using fast approximation as in [7]. However, as the tree is constructed off-line we accumulate the code length directly in the training phase by summing up the observed entropy estimates of (4).

3.4 Delayed pruning

In certain cases the expansion of a sub-tree terminates too early because of the locality of the splitting criterion. For example, consider the tree of Fig. 4. The inclusion of the third pixel in the context has only a marginal effect on the prediction; the entropy of the all-white contexts at the levels 2 and 3 are practically equal. The saving is therefore drowned out by the learning cost because the third context pixel does not reduce the frequency of black pixels in the all-white context. The inclusion of a fourth pixel, on the other hand, provides a remarkable saving.

Delayed pruning is therefore proposed to overcome the *locality problem*. The expansion is not being stopped directly when saving is not observed. Instead, the expansion is allowed to continue one level further. If neither of the children node gives improvement in the case of a further split, the expansion is stopped and the children are removed from the tree. Delayed pruning is not applied in case of deterministic contexts when either n_1^i or n_0^i equals to zero.

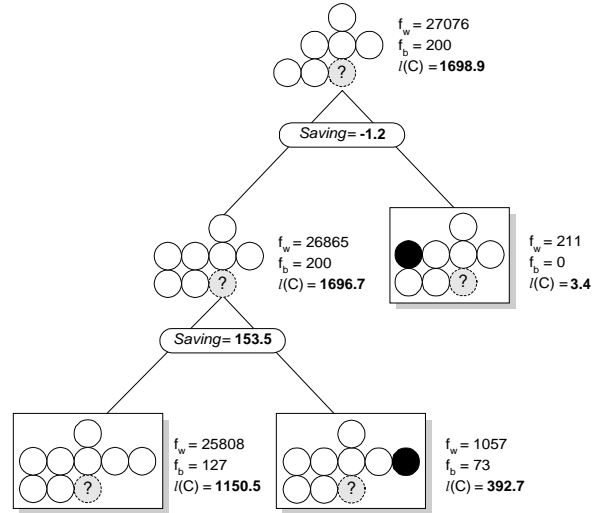


Figure 4. The locality problem of the splitting. The first split delivers saving of -1.2 and the second split of +153.5 bits.

3.6 Free tree

Another improvement to the context tree is to consider the neighboring pixels in variable order instead of the fixed order shown in Fig. 1. The *free tree* method in [7] selects the split pixel for each context separately by considering all unselected pixels in a given search area. In the semi-adaptive approach, this will give a more compact and usually better tree structure at the cost of increasing overhead because the position of the split pixel must also

be stored. The disadvantage is that the running time is multiplied approximately by the number of pixels in the search area.

In the static approach, there is no overhead from storing the tree and the running time is not as critical. The optimal selection of the split pixel reduces the depth of the tree. It therefore allows a faster compression because less neighboring pixels need to be considered in the compression phase. On the negative side, the resulting tree would be more dependent on the choice of the training image.

4. Results

The performance of the context tree is tested using two sets of A4-size images. The first set (*CCITT*) consists of the eight *CCITT* images scanned at 200 dpi, and the second set (*Newspaper*) of eight typical newspaper images scanned at 300 dpi, containing different variations of text and graphics. For training, we use a separate newspaper image of the same size and resolution.

Two variants are implemented and tested: *Context tree* refers to the one with a fixed order of the context pixels, and *Free tree* to the one with variable ordering. For these, we consider both the semi-adaptive and static approaches. The static tree (*Static*) is generated off-line using the training image, and the semi-adaptive tree (*S-A*) on-line using the input image. In the semi-adaptive approach, the cost of storing the tree is included into the splitting criterion. The additional cost is 2 bits per node for *Context tree*, and $2 + \lceil \log(40) \rceil = 8$ bits per node for *Free tree* (with 40-pixel search template). The rest of the parameter setup is given in Table 1.

Table 1. Parameter setup.

	Context tree:	Free tree
k_{MIN}	6	2
k_{MAX}	24	24
<i>Search template</i>	–	40

The overall compression performance of the static approach is summarized in Table 2. The delayed pruning gives improvement in all cases. In the case of *Context tree*, most of the improvement comes from a selective context expansion, as the compression performance remains virtually the same when k_{MIN} is increased from 6 to 10.

Table 3 shows that the static approach compares favorable to the semi-adaptive approach. It gives similar or better compression performance without the need of heavy computation in the compression phase. The actual running times of the static approach are about 2 times (*Context tree*), or 1.8 times (*Free tree*) longer than that of

the sequential JBIG. The semi-adaptive approach would require about 30 minutes (*Context tree*), or several hours (*Free tree*), depending on the size of the search template.

From the two variants, *Free tree* is the better one when applied for images with similar type (*Newspaper*) to the training image. For *CCITT* images, the *Free tree* variant is slightly better.

Table 2. Effect of the delayed pruning in the static approach. The numbers gives improvement to JBIG₁₀.

Set:	Context tree		Free tree	
	normal	delayed	normal	delayed
Newspaper	8.5 %	10.9 %	11.1 %	14.5 %
CCITT	5.2 %	7.2 %	2.5 %	6.6 %

Table 3. Comparison of the static and semi-adaptive approaches. The numbers gives improvement to JBIG₁₀.

Set:	Semi-adaptive		Static	
	Con.tree	Free tree	Con.tree	Free tree
Newspaper	8.6 %	13.4 %	10.9 %	14.5 %
CCITT	4.0 %	8.4 %	7.2 %	6.6 %

In semi-adaptive approach, the number of contexts is optimized for the input image whereas in static approach, it is optimized for the training image. For experimental purposes, we adjust the number of contexts in the following using an additional growth control parameter w , which is subtracted from (1). Positive values of w decrease and negative values increase the number of contexts. The minimum number of contexts (with $w = \infty$) is $2^6 = 32$. The maximum number of contexts (with $w = -\infty$) depends on the training image, and was observed as 14400 in case of *Context tree* and 43980 in case of *Free tree*, see Fig. 5. The effect of the parameter w on the compression performance is shown in Fig. 6.

5. Conclusions

Static context tree is considered for binary image compression. The contexts are stored in the leaves of a variable-depth binary tree. The tree structure itself is fully static. The method is a one-pass method and therefore suitable for fax communication. The use of the context tree is not significantly more complicated than JBIG. Only the context selection is modified and the other parts of the method are the same as in JBIG.

The static approach performs similar to that of the semi-adaptive approach without the cost of huge running time. When applied for a set of images similar to the training image, an improvement of about 14 % was recorded, in comparison to JBIG. For the set of *CCITT* test images, the method improves about 7 %.

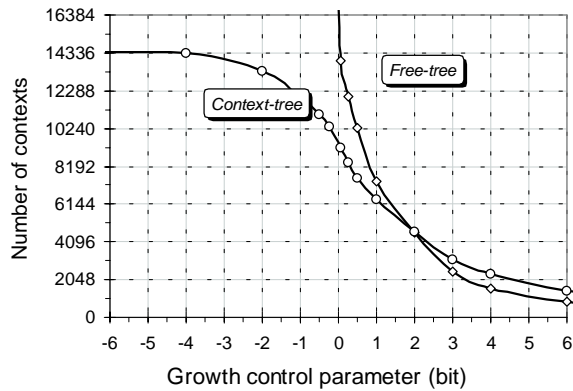


Figure 5. Number of contexts as a function of w .

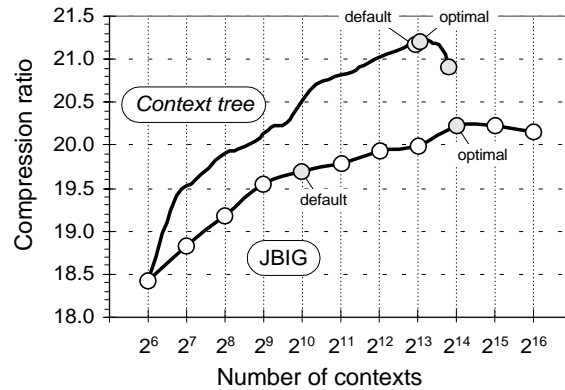


Figure 6. Compression ratio (for CCITT images) as a function of the number of contexts (*Context tree*).

Table 4. Summary of the compression performance (bytes) of the discussed methods for the CCITT test set: JBIG with 10 and 14 pixel context template (JBIG₁₀ and JBIG₁₄); Context tree and Free tree: static, semi-adaptive (S-A), and theoretical maximum improvement (Max.).

Image #	JBIG		Context Tree			Free tree		
	JBIG ₁₀	JBIG ₁₄	static	S-A	Max.	static	S-A	Max.
CCITT 1	14717	14618	14279	14402	13907	14465	14180	12889
CCITT 2	8500	8186	7678	8052	7445	7723	7664	7101
CCITT 3	21999	21263	20328	20694	19944	20870	19924	18847
CCITT 4	54300	52652	49393	50263	48591	48267	48012	42091
CCITT 5	25832	25239	24196	24587	23722	24213	23379	21694
CCITT 6	12561	12202	11287	11961	11156	11630	11200	10203
CCITT 7	56316	55116	53189	56670	52946	54480	53705	48660
CCITT 8	14238	13762	13021	13571	12906	12978	12935	12127
TOTAL	208463	203038	193371	200200	190617	194626	190999	173612
Improv.	–	2.6 %	7.2 %	4.0 %	8.6 %	6.6 %	8.4 %	16.7 %

Acknowledgements

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

References

- [1] R.B. Arps, T.K. Truong, "Comparison of international standards for lossless still image compression". *Proceedings of the IEEE*, **82**, 889-899, June 1994.
- [2] JBIG, Progressive Bi-level Image Compression, ISO/IEC International Standard 11544, ITU Recommendation T.82, 1993.
- [3] P.G. Howard, "Text Image Compression Using Soft Pattern Matching", *The Computer Journal*, **40**, 146-156, 1997.
- [4] I.H. Witten, A. Moffat and T.C. Bell, *Managing Gigabytes: Compressing and Indexing Documents and Images*. Van Nostrand Reinhold, New York, 1994.
- [5] A. Moffat, "Two-level context based compression of binary images". *IEEE Proceedings Data Compression Conference*, Snowbird, Utah, 382-391, 1991.
- [6] E.I. Ageenko and P. Fränti, "Enhanced JBIG-based compression for satisfying objectives of engineering document management system", *Optical Engineering*, **37** (5), 1530-1538, May 1998.
- [7] B. Martins, S. Forchhammer, "Bi-level image compression with tree coding". *IEEE Transactions on Image Processing*, **7** (4), 517-528, April 1998.
- [8] J. Rissanen, *Stochastic Complexity in Statistical Inquiry*. World Scientific, 1989.