Bi-level image compression with tree coding

Martins, Bo; Forchhammer, Søren

Published in:
Proceedings of the Data Compression Conference

Link to article, DOI:
10.1109/DCC.1996.488332

Publication date:
1996

Document Version
Publisher's PDF, also known as Version of record

Bi-level Image Compression with Tree Coding

Bo Martins and Søren Forchhammer
Dept. of Telecommunication, 343, Technical University of Denmark
DK-2800 Lyngby, Denmark
phone: +45 4525 2525, e-mail: bm@tele.dtu.dk and sf@tele.dtu.dk

Abstract

Presently, tree coders are the best bi-level image coders. The current ISO standard, JBIG, is a good example. By organising code length calculations properly a vast number of possible models (trees) can be investigated within reasonable time prior to generating code. Three general-purpose coders are constructed by this principle. A multi-pass free tree coding scheme produces superior compression results for all test images. A multi-pass fast free template coding scheme produces much better results than JBIG for difficult images, such as halftonings. Rissanen's algorithm 'Context' is presented in a new version that without sacrificing speed brings it close to the multi-pass coders in compression performance.

1 Introduction

Predictive coding usually means determining the state of the current unknown symbol, U_t, and coding with the probability estimate of that state. Symbol probabilities of different states are usually treated as independent and the symbols being generated in a given state are considered independent of one another – the model becomes a collection of independent Bernoulli sources often arranged in a tree structure. The assumptions of independance are questionable with real image data, nonetheless very good results in terms of compression ratio have be achieved. The baseline JBIG [3] is one such coder. Tree coding fundamentally consists of four parts: 1) Choosing the template/tree pixels and possibly ordering them, 2) Calculating the probability estimate during coding (determining the coding depth of the tree), 3) Updating the statistics tree(s) during coding, 4) Performing the arithmetic coding.
In this paper x_t denotes the tth pel relative to raster scan order. The unknown symbol at time t_+, x_{t+1}, is also denoted u_t or u. X_t, and U are stochastic variables. An ordered set of symbols, $c_0 \cdots c_k$, is denoted c_0^k.

2 Fast Bernoulli Code Length Calculation

In coding schemes where we investigate different tree models we often wish to calculate the code length of the events directly from the counts. Let $\ell(n_0, n_1|\delta)$ denote the code length of a binary string with n_0 zeroes and n_1 ones using a Bernoulli model where

1068-0314/96$5.00 © 1996 IEEE

the prior initially is beta-distributed with the nuisance parameters $\delta_0 = \delta_1 = \delta$. We have:

$$\ell(n_0, n_1|\delta) = \begin{cases} 0 & \text{if } n_0 = 0 \text{ and } n_1 = 0 \\ -\log \frac{\prod_{j=0}^{n_0-1}(\delta+j)\ \prod_{j=0}^{n_1-1}(\delta+j)}{\prod_{j=0}^{n_0+n_1-1}(2\delta+j)} & \text{if } n_0 \neq 0 \text{ and } n_1 \neq 0 \\ -\log \frac{\prod_{j=0}^{n_0+n_1-1}(\delta+j)}{\prod_{j=0}^{n_0+n_1-1}(2\delta+j)} & \text{if } n_0 \neq 0 \text{ xor } n_1 \neq 0 \end{cases} \quad (1)$$

The value of δ is optimized to 0.45 in [3] for a large number of bi-level images. $\delta = 0.5$ minimizes the stochastic complexity relative to the class of all prior distributions [8]. The sequential estimator that corresponds to this expression is:

$$\hat{p}(X_{n_0+n_1+1} = 0|n_0, n_1, \delta) = \frac{n_0 + \delta}{n_0 + n_1 + 2\delta} \quad (2)$$

To make a fast calculation of $\ell(n_0, n_1|\delta)$ we use a table, T_2, to look up the value for $n_0 < N \wedge n_1 < N$. ($N = 100$ is suitable). For larger values of n_0 and/or n_1 we use Stirling's formula to approximate the value:

$$\ell(n_0, n_1|\delta) \simeq \begin{cases} (n_0 + n_1 + 2\delta - 0.5)\ln(n_0 + n_1 + 2\delta) & \\ -(n_0 + \delta - 0.5)\ln(n_0 + \delta) & \\ -(n_1 + \delta - 0.5)\ln(n_1 + \delta) + \xi_1 & \text{if } n_0 \geq N \text{ and } n_1 \geq N \\ T_1(n_1) + (n_0 + \delta - 0.5)\ln\frac{n_0+n_1+2\delta}{n_0+\delta} & \\ +(n_1 + \delta)(\ln(n_0 + n_1 + 2\delta) - 1) + \xi_2 & \text{if } n_0 \geq N \text{ and } n_1 < N \\ T_1(n_0) + (n_1 + \delta - 0.5)\ln\frac{n_0+n_1+2\delta}{n_1+\delta} & \\ +(n_0 + \delta)(\ln(n_0 + n_1 + 2\delta) - 1) + \xi_2 & \text{if } n_0 < N \text{ and } n_1 \geq N \\ T_2(n_0, n_1) & \text{if } n_0 < N \text{ and } n_1 < N \end{cases} \quad (3)$$

where $\xi_1 = -\ln\frac{\sqrt{2\pi}\Gamma(2\delta)}{\sqrt{\Gamma(\delta)}}$ and $\xi_2 = -\ln\frac{\sqrt{\Gamma(2\delta)}}{\sqrt{\Gamma(\delta)}}$ are constants. $T_1(n)$ tabelizes $\ell(n, 0|\delta)$.

3 Free Template Tree Coding

3.1 Coding of the Data and the Tree

By free k-order template tree coding we understand coding with a balanced tree of depth k of the form U_t given C_t. The code length of the image x_1^T is given by $\sum_{t=1}^{T} L(u_t|(c_0^{k-1})_t)$, where the i-th context bit is a static function of the past: $(C_i)_t = s_i(X_{t-1}, \cdots, X_{t-K})$. In its simplest form free template tree coding involves a ranking of the past:

$$(C_i)_t = X_{t-r(i)} \quad (4)$$

The tree structure of a free template tree requires little coding. For a tree of depth k, we need to transmit k and $r(0) \cdots r(k-1)$. Each of the numbers $r(i)$ may be indexed with something like 11-13 bits, i.e. virtually nothing. We shall not consider this cost in the following. Related work for free template tree coding is given in [2] [1].

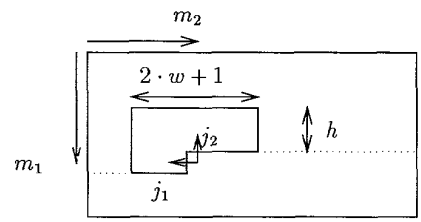


Figure 1: Search area for context pels. Definition of absolute and relative coordinates.

3.2 Determination of the Ranking - Greedy Build-up of Template

3.2.1 Full Search

Pure free template tree coding with a tree of depth k means using the leaves as the coding states. The kth context bit, C_k, may be chosen among a number of candidates (Figure 1) $\{A_1 \cdots A_{N_a}\}$, $N_a = h \cdot (2w+1) + w$. The code length that would result from picking A_j may be calculated as follows:

$$L(x_1^T) = \sum_{c_0^{k-2}=0_0^{k-2}}^{1_0^{k-2}} \sum_{a_j=0}^{1} \ell(n_{0|c_0^{k-2}a_j}, n_{1|c_0^{k-2}a_j}|\delta) \tag{5}$$

To calculate the code length for a large number of possible kth order tree codings we only need a single pass of the data, updating a (large) table as we go along. With Eq. 3 it becomes feasible to calculate the code length for a number of candidates since all that is required is a collection of statistics. Still, the full search speed is poor and its memory consumption substantial. We seek means to trade off compression ratio for speed in the template selection phase and for the memory requirement. We assume a limited memory for the statistics tables(M_{st}) which we use to pick the next context pixel. The balanced tree coding reported here still requires enough memory to buffer the entire image.

3.2.2 Limiting the candidate list for new context pixels

A greedy search for a (k+1)'th context bit requires a memory of $2^{k+2}N_a \cdot sizeof(\text{counters})$ bytes. Therefore, we choose N_a to be inversely proportional to k for $k > k_{sh}$. Which pels should remain depends on the type of image. For most images the pels can be ranked according to their 1-norm distance to U with good results. By a more complex scheme graphics arts halftonings stand to gain approximately 5 per cent extra.

3.2.3 Limiting the number of pixels (subsampling)

If a context coder is faced with an unknown halftoning it is important that the search area for context pixels contains at least one grid intersection and preferably more. For clustered dot halftonings we cannot count on the grid being placed in a specific position, hence, the search area should initially be dense and quite large. The initial number of candidates is therefore substantial.
The time for performing a greedy search for a $(k+1)$'th context bit is (with the fast calculation of $\ell(n_0, n_1|\delta)$) approximately the time it takes to collect statistics:

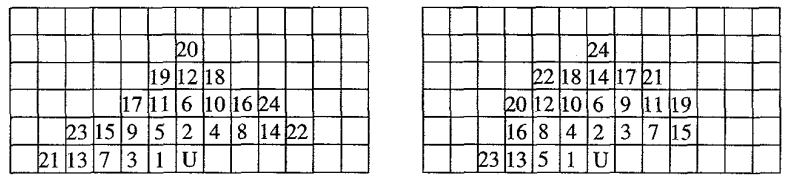
k	0	1	2	3	4	5	6	7	8	9	10	11	>11
k_s	5	5	5	5	5	5	5	4	3	2	1	1	0

Table 1: A subsampling scheme. $k_s = 0$ means 'no subsampling'

$T(N_a K_1 + K_2 \cdot k)$, K_1 and K_2 are constants with respect to N_a and T. To reduce this time we replace T with $n(k)$, basing the choice of the $(k+1)$'th context pel on a subset of the image. One way to implement this is to cut out what is supposedly a representative part of the image. This procedure is extremely sensitive to instationarities so in our work we therefore choose to emulate coding of pels which are scattered over the entire image. A simple way to control the fraction of pels to code as well as a fast way of calculating the position of them is to pick the sample pels of a subsampling of the image but letting them stay in their original context. Subsampling in two dimensions is described in [12](ch. 2, M. Vetterli): points on the input lattice (m_1, m_2) are associated with points on a subsampling lattice (u_1, u_2) We only consider scaled hexagonal subsampling of the form 6. By hexagonal subsampling the smallest 2-norm distance between samples is maximal. The risk of using a regular subsampling pattern on halftonings is reduced with hexagonal subsampling compared to a (faster) scheme where m_1 and m_2 are separable.

$$\begin{bmatrix} m_1 \\ m_2 \end{bmatrix} = k_s \cdot \begin{bmatrix} 2 & 1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{6}$$

When using subsampling there's a definite risk that the initialization cost of the bigger tree will outweight the reduction in entropy, so that the tree building terminates prematurely. To see this consider the asymptotic expression for the difference of the initialization cost (model cost) of balanced trees of depth $k+1$ and k, respectively: $\Delta I_n \simeq (2^k - 2^{k-1}) \log n$. The entropy difference is $\Delta H_n = (H(U|C_0^{k+1}) - H(U|C_0^k))n$. When subsampling, n is less than the number of pels of the image, T, the effect being that not enough weight is being put on the entropy differences when making the expansion decision by PMDL (see Sect. 6) based on n. The equations indicate that we might use another criterion for expanding the tree than the PMDL principle. In theory we might estimate ΔI_n and ΔH_n, calculate ΔI_T and ΔH_T by compensating with $\frac{\log T}{\log n}$ and $\frac{T}{n}$, respectively, and accept expansion iff $\Delta H_T - \Delta I_T > 0$. In practice, the asymptotic expressions cannot be carried that far, especially not when we choose a model based on all the data – the initialization cost expression is amiss. To avoid premature termination of the tree building, we instead decrement the subsampling constant, k_s, with the tree order, k. We use the subsampling scheme of table 1. The subsampling scheme is guided by an attempt to keep the variance of the estimate of $H(U|C_0^{k+1})$ the same as that of the estimate of $H(U|C_0^k)$ (we choose $n(k+1) \simeq n(k)\sqrt{2}$) while taking into account the experimental fact that $H(U|C_0^{k+1}) - H(U|C_0^k)$ is a decreasing function of k, implying that subsampling can be coarser for small values of k. As $n = \frac{T}{4 \cdot k_s^2}$ there is no need to make k_s any larger than 5 for very small values of k. Table 2 shows the impact of this subsampling scheme. It is pleasing to notice that the tree building stopped too early for those images only that do not benefit from free template tree coding opposed to JBIG coding with its default 3-line template. A simple coding scheme that covers the entire test set is to try out free template tree building with the suggested subsampling scheme. If the tree construction terminates while $n = T$, we use the found template in the coding of the data – if not we code with the default JBIG 3-line template.



1-norm 2-norm

Figure 2: Ranking of the past. The 10 first pixels of smallest 2-norm constitute the 3-line JBIG template with default positioning of the AT pixel.

3.3 Non-leaf coding

The template was picked so as to optimize leaf coding with the balanced tree of depth k, so this is an obvious and a fast choice. We do, however, have several other choices: Context Tree Weighting (CTW) [11], Context [5] [6] [9], and leaf coding with a sub-tree.
In [10] it is proven that a minimum redundancy estimator for a source within the class of FSMX-sources (of maximal depth k and a fixed ordering of the past symbols) is obtained by CTW. CTW has some implementational defects, that makes it an order of magnitude slower than 'Context'. It follows from [11] that CTW (quickly) converges to a particular sub-tree of the balanced tree. It therefore seems a better use of encoding time to find this sub-tree. Algorithm 'Context' presents one solution – it is treated later on. Leaf coding with a sub-tree cuts off initialization costs of the balanced tree while maintaining acceptable speed. For the test images a new 'Context' version improves the compression results by 1-5 per cent depending on k and the type of image while leaf coding with a sub-tree improves the coding by 1-3 per cent depending on the order. Encoding speed suffers substantially.

4 Free Tree Coding

4.1 Coding of the data

By free tree coding we understand coding of U_t given C_t. The code length of the image x_1^T is given by $\sum_{t=1}^{T} L(u_t|(c_0^{k_t-1})_t)$. The i-th context bit is a time varying function as it depends on c_0^{i-1}: $(C_i)_t = s_i(X_{t-1}, \cdots, X_{t-K}, (c_0^{i-1})_t)$. Free tree coding leaves the problem of determining k_t. Ordinarily we shall choose to code with a leaf.

$$(C_i)_t = X_{t-r((c_0^{i-1})_t)} \tag{7}$$

An (unrealistically small) example is given in Figure 3. The coordinates (j_1,j_2) that identify the splittings are relative to the position of U (Figure 1).

4.2 Coding of the tree

The cost of coding the free tree may be very large and the free tree algorithm must therefore balance the gain in coding the identity of another splitting with the lessened cost of coding the data when the splitting is given.

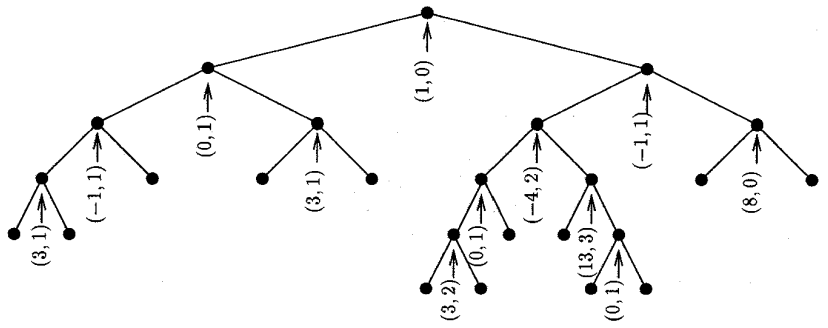


Figure 3: Free tree for c04a200 (with $b = 4000$ bits). The cost of coding the data is 493328 bits ($C_r = 8.32$). The cost of coding the (anonymous) tree structure is 25 bits, to code the identity of the 12 splittings we use $12 \cdot 11$ bits. The codestring for the (anonymous) tree structure is 1111000100111100010100100.

The free tree is unbalanced, so besides the identity of the splittings, we need to code the (anonymous) tree structure itself. To do so we adopt the procedure of [4]:
We create a binary string y to describe the anonymous tree structure where y_j equals 0 if node no. j is a leaf. The nodes of the tree are visited from root to leaf, left to right. Consider the example of figure 3 - the nodes are visited as follows: root $\rightarrow$ 0 $\rightarrow$ 00 $\rightarrow$ 000 $\rightarrow$ 0000 $\rightarrow$ 0001 $\rightarrow$ 001 $\rightarrow$ 010 $\rightarrow$ 011 $\rightarrow$ 1 aso.
Let the number of leaves in the tree be denoted $m+1$, then the total number of nodes is $2m + 1$ and the number of splitting nodes (internal nodes) is m.
A rough estimate for the cost of coding the tree is formed by the case where we do not compress it at all: To code y we require $2m + 1$ bits. The identity of a splitting requires $\log\lceil |w + h(2w + 1)| \rceil$ bits per. splitting. With a suitable prefix code and $w = h = 2^j$ the cost of coding a splitting becomes $\simeq 2j + 1$ bits. It is clear that both y_1^{2m+1} and the string of identities, i_1^m, may be compressed by a universal code. Indeed, the best coding in terms of compression ratio requires an interaction between the procedure of building the tree and of coding y_1^{2m+1} and i_1^m. However, simulating smaller (optimistic) tree coding costs does not affect the code length of the data much, so what may be gained by this very complex tree coding scheme is basically a reduction of the tree coding costs which constitute only about 5 per cent of the total code length. The simplest way to control the growth of the tree is to assume a fixed cost, b, of coding a splitting. To increase speed one may very well choose a larger value of b than $2j + 3$ as the most compression is usually obtained with the first few branches of the big coding tree. The example of figure 3 illustrates this point: a free tree with as few as 13 leaves can compress $c04a200$ 8.32 times.

5 Free Tree Coding Without Explicit Coding of the Tree

In the previous section we considered coding of the data as a two-step procedure where we transmit the tree first and the data next. Can we do better by a procedure where both coder and decoder build the tree as they go along? This question relates to the fundamental problem of when to choose some model. Our results show that in order

Image	Free template						Free tree	
	$k_{sh} = 12$ $M_{St} = 32$M		$k_{sh} = 6$ $M_{St} = 0.5$M					
	No subs.		No subs.		subs.			
	bits	C_r	bits	time	bits	time	bits	C_r
s01a400	96883	137.99	97011	2:08:53	-	1:43	82294	162.46
s02a400	115384	115.87	115660	2:09:04	-	1:43	101742	131.40
s03a400	1050527	12.73	1056234	2:10:53	1067570	6:32	936140	14.28
s04a400	329684	19.08	332964	1:01:13	333284	2:41	310177	20.28
s04b400	302783	20.78	306675	1:01:08	304199	2:38	276651	22.74
s04d400	277102	22.70	276634	1:01:05	285802	2:44	265127	23.73
s05a400	235646	56.73	235856	2:08:36	239870	5:25	215808	61.95
s06a400	1236790	10.81	1290013	2:11:33	1362140	6:14	1054812	12.67
s07a400	172631	77.44	172631	2:09:06	175720	6:08	153440	87.13
s08a400	38026	245.59	39030	1:30:02	40968	4:41	30450	306.70
s09a400	553197	1.90	556449	11:14	576837	28	544079	1.93
s10a400	85987	278.95	89632	3:52:55	95636	12:12	65583	365.74
c01a200	117173	35.04	117359	39:41	-	27	98607	41.64
c02a200	60782	67.55	60801	39:19	-	30	54998	74.65
c03a200	163132	25.17	163344	39:28	-	52	141268	29.06
c04a200	379601	10.82	385027	40:14	400237	1:46	296148	13.86
c05a200	190899	21.51	191973	39:24	-	31	163745	25.07
c06a200	90333	45.45	90440	39:18	-	39	80193	51.20
c07a200	426998	9.62	426998	39:59	-	31	384521	10.68
c08a200	104942	39.12	104803	39:12	-	31	91761	44.74
a80c	3056190	9.78	3158518	4:56:44	3308654	14:09	2529636	11.81

Table 2: Multi-pass results. Initial search area:$(h, w) = (16, 16)$, limitation of search area (1-norm) for $k > k_{sh}$. A stroke in the subsampling column indicates that the expansion of the tree stopped when $k_s > 0$. Free tree expansion cost: $b = 11$bits. Time:(h:m:s)

to match let alone beat the multi-pass coding schemes we cannot use the predictive minimum description length principle because statistics are too thin at the proper time of decision. This leaves us only with a weighting approach which is disregarded because of the huge number of possible tree structures.

6 Context Variants

Balanced tree coding means coding with a leaf. Algorithm context provides the means to adaptively select that subtree of the larger statistics tree that provides the best coding. Algorithm context applies to any kind of tree.
By context coding we understand coding of U_t given C_t. The code length of the image x_1^T is given by $\sum_{t=1}^{T} L(u_t|(c_0^{k_t-1})_t)$. The definition of the i-th context bit is irrelevant, whereas the computation of k_t is important.
A number of context variants have been identified [5] [6] [9]. They are all based on the predictive minimum description length principle (PMDL) [7]: having collected statistics for a number of models and computed the code length of the data given each model, we choose to code U with the probability estimate of the one model that has

done best in the past. With statistics being collected in the nodes of a tree, the model selection problem is one of choosing a particular node in that path, P, of the tree that defines the current context. The context schemes [5] [6] [9] only consider father-son comparisons; they require that one stores at each son node the floating point account of the difference in code length between the father model and the son model. The incidents in the account are the events that occured in both father and son. We may view this in another way (one that avoids the need for storing floating point accounts in the tree):

Let the length of P be denoted l (in practical applications l has some maximal value). A father-son PMDL-optimal (FSPMDL-optimal) selection of the coding node may be done in at most l steps: Compute the FSPMDL-optimal context in longer and longer subpaths of P, naming the FSPMDL-optimal contexts $C^*(1), C^*(2), \cdots, C^*(l)$ and their values $c^*(1), c^*(2), \cdots, c^*(l)$. Let the value of the context which corresponds to the subpath of length s be named c_0^s. Now we declare $C^*(s) = C_0^{s-1}$ if

$$\ell(n_{0|c_0^s}, n_{1|c_0^s}|\delta) + \ell(n_{0|c_0^{s-1}} - n_{0|c_0^s}, n_{1|c_0^{s-1}} - n_{1|c_0^s}|\delta) \geq \ell(n_{0|c_0^{s-1}}, n_{1|c_0^{s-1}}|\delta) \qquad (8)$$

and $C_0^{k-1} = C_0^{s-1}$ and terminate the search otherwise. If the search did not terminate before depth l, $C_0^k = C_0^l$.

The FSPMDL scheme produces poor results (compared to JBIG) because it ignores the fact that for usual images U is always strongly correlated with its neighbours; the algorithm should not be so hestitant in climbing the tree (in the beginning). Gilbert Furlan ([9]) saw this and initializes the code length difference between some node and its father by a positive amount (5 bits). The PMDL principle tells us to use the father node when the difference is negative, so the sons are given an edge by the initialization. The updating of the statistics tree becomes slow in Furlans algorithm as l code lengths have to be calculated after having observed U.

Our bid for an improved baseline version of 'Context' (we name it Full Path Algorithm Context-Baseline) is an l-step algorithm to perform the selection of one of the nodes. (In our implementation l has some not-too-large maximal value). We compute the 'optimal' context in longer and longer subpaths of P, naming the 'optimal' contexts $C^*(1), C^*(2), \cdots, C^*(l)$ and their values $c^*(1), c^*(2), \cdots, c^*(l)$. Let the value of the context which corresponds to the subpath of length s be named c_0^s. Now we declare $C^*(s) = C^*(s-1)$ if

$$\ell(n_{0|c_0^s}, n_{1|c_0^s}|\delta) + \ell(n_{0|c^*(s)} - n_{0|c_0^s}, n_{1|c^*(s)} - n_{1|c_0^s}|\delta) > \ell(n_{0|c^*(s)}, n_{1|c^*(s)}|\delta) \qquad (9)$$

and $C^*(s) = C_0^s$ otherwise. In the end we set $C_0^{k-1} = C^*(l)$.

The idea is to look further down in the path than the first place a father beats the son. If the statistics of some node S further down P is sufficiently different we should use that node as the coding node rather than the so far best. By constructing a brother to S by collapsing the tree we are not seriously burdening the sons model by model costs, which is what makes the algorithm work. The main advantage of this context version is that we do not suffer badly from arranging the context poorly. If we have a 'noise' context bit in the path P the original father-son scheme stumbles far too long at the father node. A constant bias cannot solve this problem. For all three schemes 'noise' context bits seriously reduce performance because statistics are weakened. In our context version updating the statistics tree requires little time whereas finding the coding node is the time critical part.

After this paper went to review we have abandoned the baseline algorithm mainly in order to increase speed. The elements in the improved algorithm (FPAC-F) which amount to a 10-fold increase in speed are 1) Checking for best context only once in a while. 2) Local adaptivity by deminishing counts. 3) Typical prediction. 4)

Image	JBIG				FPAC-B			
	Def. 3-line		Adaptive		1-norm		2-norm	
	bytes	C_r	bytes	C_r	bits	time	bits	time
s01a400	13850	120.66	13724	121.77	98108	1:53:12	98830	1:52:47
s02a400	15523	107.66	15493	107.87	115960	1:48:19	115283	1:48:54
s03a400	146766	11.39	146774	11.39	1033176	1:53:42	1036842	1:53:52
s04a400	129707	6.06	58153	13.52	515455	45:58	520204	45:56
s04b400	159412	4.93	56651	13.88	470134	46:47	713211	46:53
s04d400	120418	6.53	48069	16.36	327222	45:30	329117	45:42
s05a400	34576	48.33	34206	48.86	240805	1:50:26	242707	1:50:37
s06a400	256091	6.53	212874	7.85	1238323	1:48:34	1270465	1:49:00
s07a400	24067	69.44	23076	72.42	170874	1:20:55	173429	1:51:04
s08a400	6125	190.59	6133	190.34	36573	1:14:45	37141	1:15:32
s09a400	74429	1.76	73075	1.79	585543	6:11	567031	6:15
s10a400	13861	216.31	14078	212.98	87721	3:20:00	84410	3:19:45
c01a200	14828	34.61	15033	34.14	113899	33:44	114685	33:45
c02a200	8675	59.16	8677	59.15	60930	32:55	59297	33:06
c03a200	22094	23.23	22187	23.13	160428	34:08	161789	34:07
c04a200	54477	9.42	55048	9.32	388871	33:42	397442	33:37
c05a200	25988	19.75	26168	19.61	190032	33:52	193250	33:51
c06a200	12662	40.53	12633	40.63	88687	32:59	88274	33:10
c07a200	56474	9.09	56179	9.14	418549	33:48	418601	33:48
c08a200	14369	35.72	14525	35.33	107010	33:47	106225	33:50
a80c	695790	5.37	519853	7.19	4996276	4:05:41	5055356	4:07:20

Table 3: JBIG and Context results. Search area: $(h, w) = (16, 16)$. JBIG – Adaptive: The adaptive template pixel is determined as the best 10th template pixel in a 10th order free template tree coding where the 9 first template pixels are chosen as the fixed template pixels of the 3-line JBIG template. Context: coding with different default orderings of the causal pels. Time:(h:m:s)

A new growth rule for the context tree. For clustered dot halftonings we create a pixel ordering using the knowledge of V_1 (and V_2). With this pixel ordering FPAC-F (or FPAC-B) can compete in compression performance with the adaptive template compression scheme.

6.1 Comments on the Results

The test images which are used in this paper are mainly the Stockholm(JBIG) test set and the (ambigous) CCITT test set; a80c (6144 · 4864 pels) is a test image for the graphics arts in Scandinavia. The images contain scanned text, line art, and halftonings. For the clustered dot halftonings V_1 and V_2 span the halftoning screen. The simple halftonings are s04a400 ($V_1 \simeq (4,0)$, $V_2 \simeq (0,4)$), s04b400 ($V_1 \simeq (5,0)$, $V_2 \simeq (0,5)$), and s04d400 ($V_1 \simeq (3,0)$, $V_2 \simeq (0,3)$). The troublesome halftonings are the mixture image s06a400 ($V_1 \simeq (4,0)$, $V_2 \simeq (0,4)$), the error diffusion s09a400, and the clustered dot halftoning a80c ($V_1 \simeq (-11,5.5)$, $V_2 \simeq (5.5,11)$). Except for JBIG where code is generated using a QM-coder the listed code lengths are ideal (calculated). Using the arithmetic coder of [7] the actual length of the generated

code is usually something like 100 bits more than the ideal code length. Arithmetic encoding/decoding is very much faster than other parts of the algorithms (~2 seconds for a CCITT image). All algorithms are implemented in C on a HP 755 computer.

7 Conlusion

We have considered three general purpose schemes of predictive coding. Free tree coding produces superior compression results for all types of images, but the technique is too slow for practical purposes. Free template coding gives substantial improvements over JBIG for halftonings and moderate improvements for images of dense printing. Two techniques for speeding up free template tree encoding have been presented, decoding is almost JBIG-fast. A general purpose one-pass codec of good compression performance has been established by a new version of algorithm context – the default pixel ordering should be by 1-norm.

References

[1] R. Arps, T. Friedman, and R. Pasco. Optimizing Models for Data Compression using the MDL Principle and Stochastic Complexity. In *Proc. of '1990 Picture Coding Symposium'*, Cambridge, MA, Mar. 1990. Session 5.6.

[2] S. Forchhammer and U. Skands. Adaptive Compression of Bi-level halftone images using the JBIG arithmetic coder. Technical Report IT-135, Institute of Telecommunication, Technical University of Denmark, Febr. 1993.

[3] JBIG. Progressive Bi-level Image Compression. ISO/IEC International Standard 11544, 1993.

[4] R. Nohre. *Some Topics in Descriptive Complexity.* PhD thesis, Linköping University, 1993. Linköping Studies in Science and Technology. Diss. No. 330.

[5] J. Rissanen. A Universal Data Compression System. *IEEE Tr. Inform. Theory*, IT - 29(5), Sept. 1983.

[6] J. Rissanen. Complexity of Strings in the Class of Markov Sources. *IEEE Tr. Inform. Theory*, IT - 32(4), July 1986.

[7] J. Rissanen. *Stochastic Complexity in Statistical Inquiry.* World Scientific, 1989.

[8] J. Rissanen. Fisher Information and Stochastic Complexity. Accepted to Tr. Inform. Theory, 1994.

[9] J. Rissanen. Noise Separation and MDL Modeling of Chaotic Processes. In P. Grassberger and J.-P. Nadal, editors, *From Statistical Physics to Statistical Inference and Back.* Kluwer, London, 1994.

[10] Y. Shtarkov. Switching Discrete Sources and its Universal Encoding. *Problemy Peredachi Informatsii, 28(3):95-111*, July-Sept. 1992. Eng. transl. pp. 282-296.

[11] F. Willems, Y. Shtarkov, and T. Tjalkens. Context Tree Weighting: A Sequential Universal Source Coding Procedure for FSMX Sources. In *IEEE Symp. Information Theory*, page 59, 1993.

[12] J. W. Woods. *Subband Image Coding.* Kluwer, 1991.