Tree Coding of Bilevel Images

Martins, Bo; Forchhammer, Søren

Published in:
I E E E Transactions on Image Processing

Link to article, DOI:
10.1109/83.663496

Publication date:
1998

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

Tree Coding of Bilevel Images

Bo Martins and Søren Forchhammer

***Abstract*—Presently, sequential tree coders are the best general purpose bilevel image coders and the best coders of halftoned images. The current ISO standard, Joint Bilevel Image Experts Group (JBIG), is a good example. A sequential tree coder encodes the data by feeding estimates of conditional probabilities to an arithmetic coder. The conditional probabilities are estimated from co-occurrence statistics of past pixels, the statistics are stored in a tree. By organizing code length calculations properly, a vast number of possible models (trees) reflecting different pixel orderings can be investigated within reasonable time prior to generating the code. A number of general-purpose coders are constructed according to this principle. Rissanen's one-pass algorithm, *context*, is presented in two modified versions. The baseline is proven to be a universal coder. The faster version, which is one order of magnitude slower than JBIG, obtains excellent and highly robust compression performance. A multipass free tree coding scheme produces superior compression results for all test images. A multipass free template coding scheme produces significantly better results than JBIG for difficult images such as halftones. By utilizing randomized subsampling in the template selection, the speed becomes acceptable for practical image coding.**

***Index Terms*— Bilevel images, context, halftone, image compression, JBIG.**

I. Introduction

BILEVEL image compression is useful in transmission and archival applications. Facsimile transmission is still a very important application because information is presented to the receiver in the layout the sender chooses without the sender having to support a broad range of receiver platforms. For the same reason, distribution by computer nets of documents of mixed text and halftones as bitmaps may also increase in the future. High resolution text (400–600 dpi) and high-quality halftones may be transmitted without increased bandwidth provided more advanced algorithms than the present ones are put into use. Especially, if instead of being scanned, the documents are computer generated.

In this paper, we construct algorithms for compression of primarily bilevel image material. The new algorithms may also, with simple modifications, be used as stand-alone coders of continuous-tone images with few (2–5) b/pixel and as general purpose entropy/universal coders. We consider methods based on sequential prediction of conditional probabilities that are coded using arithmetic coding [1]. Two very important examples of coders that use finite context modeling are JBIG [2] and the *context algorithm* [3]–[6]. Where baseline JBIG operates with a specific model class defined by a ten-pixel template (its default appearance is depicted in Fig. 1), the context algorithm can be used as a variable-size template algorithm. The past used for conditioning may be organized in a tree. Important aspects of tree coders are the choice and ordering of past prediction pixels and how many to use, i.e., the depth of each prediction. For one of the new algorithms, the best compression on standard test images is the sole aim. We construct a number of algorithms which are much less complex, but still achieve considerably better compression results than JBIG. Two of the new algorithms are modified versions of the context algorithm. They are tailored for bilevel image compression rather than universal coding and trade some generality of the context algorithm for advantages in terms of speed, compression performance, and memory usage. The baseline version of the two is universal for a class of finite context (finitely generated) stationary ergodic sources.

In Section II, we present a method for fast calculation of code length useful for evaluating different models. The context algorithm is treated in Section III. The new versions are presented. A fast subsampling based method for choosing and ordering context pixels is presented in Section IV. Aiming at the highest compression, a generalization of template coding to free tree coding is presented in Section V. Results are presented in Section VI.

II. Finite Context Coding

We consider tree coding of a bilevel image, x^T, where the pixels are coded sequentially in raster scan order. In this paper, x_t denotes the tth pixel relative to raster scan order. The unknown symbol at time t, x_{t+1}, is also denoted u. Stochastic variables are written in capitals, e.g., U and C. We adopt the usual shorthand for strings and apply it to any ordered set of symbols, thus, $c_1 \cdots c_k$, is denoted c^k. We use $p(\cdot|\cdot)$ to denote a conditional probability and $\hat{p}(\cdot|\cdot)$ to denote the corresponding estimate. The arguments specify the conditional probability in question.

In bilevel image coding the probability of the next symbol U is conditioned on its context. The context may be the value of a number of pixels in a fixed spatial relationship to U. The selection of this set is a key problem. Based on the results for finitely generated one-dimensional (1-D) sources [3], the problem is frequently divided into an ordering of the pixels, thereby defining a context string, c^t, and a selection of a prefix of this string as the coding context. Symbol probabilities of different contexts as well as symbols generated in a given context are usually treated as being independent. Thus, the model becomes a collection of independent Bernoulli sources, often arranged in a tree structure. By the assumptions of independence, it becomes a simple matter to assign conditional

Manuscript received July 4, 1996; revised February 20, 1997. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Robert Forchheimer.

The authors are with the Department of Telecommunications, Technical University of Denmark, DK-2800 Lyngby, Denmark (e-mail: bm@tele.dtu.dk; sf@tele.dtu.dk).

Publisher Item Identifier S 1057-7149(98)02459-2.

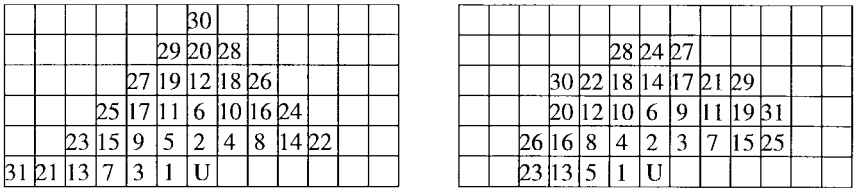

Fig. 1. Default orderings of the past with maximum depth, $l = 31$. The ten first pixels of smallest 2-norm constitute the 3-line JBIG template with default positioning of the adaptive pixel.

probabilities to a new event: Having determined the coding context, the counts of that context alone form the basis of the probability estimate of U being zero. If n_0 zeroes and n_1 ones were generated in context, c, we use the (sequential) estimator:

$$\hat{p}(0|c) = \frac{n_0 + \delta}{n_0 + n_1 + 2\delta}. \tag{1}$$

In principle, this is the JBIG [2] estimator. The value of δ is optimized to 0.45 in [2] for a large number of bilevel images. $\delta = 0.5$ minimizes the stochastic complexity relative to the class of all prior distributions [7]. The estimator is optimal [8] if the events generated in context c are independent and if the prior probability of $U = 0$ initially is beta distributed with the nuisance parameters $\delta_0 = \delta_1 = \delta$.

A. Fast Bernoulli Code Length Calculation

In coding schemes where we investigate different tree models, we often wish to calculate the accumulated code length of the events directly from the node counts. Let $\ell(n_0, n_1|\delta)$ denote the code length of a binary string with n_0 zeroes and n_1 ones coded sequentially according to (1), as follows:

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

To make a fast calculation of $\ell(n_0, n_1|\delta)$ we use a table, $T_2(n_0, n_1)$, to look up the value for small counts: $n_0 < N \wedge n_1 < N$. (The constant N is 100 in our applications.) For larger values of n_0 and/or n_1, we use Stirling's formula to approximate the value as shown in (3), at the bottom of the page, where $\xi_1 = -\log[\sqrt{2\pi}\Gamma(2\delta)/\Gamma^2(\delta)]$ and $\xi_2 = -\log[\Gamma(2\delta)/\Gamma(\delta)]$ are constants. $T_1(n)$ is a table for $\ell(n, 0|\delta)$. The logarithms are base 2. $\Gamma(\cdot)$ denotes the gamma function.

III. Context Algorithm Revisited

The context algorithm, first introduced in [3] and later refined in [4]–[6], provides the means to select between a number of context models for the data, x^T. For bilevel images, the context string c^t is defined by a time-invariant ordering of pixels and the coding context c is picked as a time-varying prefix of the context string. Pixel ordering is deferred to Section IV. Organizing the reversed context strings, i.e., the suffixes, in a tree, the context algorithm specifies the context to be used at each time instance in two (interlaced) stages. The first stage grows the tree. The second selects the context.

Node c of the context tree stores the counts $n_{u|c}$, $u = \{0, 1\}$. The count $n_{u|c}$ denotes the number of times that U took on value u in context c. Having selected the coding context, say c^k, the current pixel U is encoded with arithmetic coding using $\hat{p}(0|c^k)$ as the conditional probability for $U = 0$. The probability estimate is computed from the counts in the context tree using (1). The context c^0 denotes the zero-order context.

The context algorithm has been presented in slightly different variations. Below the context selection rules are reviewed. We formulate them for a binary alphabet.

Traversing the tree path, P, formed by the $|P|$ first bits in c^t, the context algorithm selects the coding node for U by a series of father-son code length comparisons. The algorithm

$$\ell(n_0, n_1|\delta) \simeq \begin{cases} -\left(n_0 + \delta - \frac{1}{2}\right)\log(n_0 + \delta) - \left(n_1 + \delta - \frac{1}{2}\right) \\ \quad \cdot \log(n_1 + \delta) + \left(n_0 + n_1 + 2\delta - \frac{1}{2}\right) \\ \quad \cdot \log(n_0 + n_1 + 2\delta) + \xi_1 & \text{if } n_0 \geq N \text{ and } n_1 \geq N \\ T_1(n_1) + \left(n_0 + \delta - \frac{1}{2}\right)\log\frac{n_0+n_1+2\delta}{n_0+\delta} \\ \quad +(n_1 + \delta)[\log(n_0 + n_1 + 2\delta) - 1] + \xi_2 & \text{if } n_0 \geq N \text{ and } n_1 < N \\ T_1(n_0) + \left(n_1 + \delta - \frac{1}{2}\right)\log\frac{n_0+n_1+2\delta}{n_1+\delta} \\ \quad +(n_0 + \delta)[\log(n_0 + n_1 + 2\delta) - 1] + \xi_2 & \text{if } n_0 < N \text{ and } n_1 \geq N \\ T_2(n_0, n_1) & \text{if } n_0 < N \text{ and } n_1 < N \end{cases} \tag{3}$$

stops the first time a father is better than his sons. In [4] and [5] the direction of traversal is from root to leaf. In [3] and [6], the direction is toward the root in the sense that the deepest node is chosen.

In [4]–[6], the basis of the father-son comparison is an efficiency difference—an accumulation of the difference in code length at the father node c^{s-1} and the son c^s of the instances occurring at the son c^s. For the instance u the contribution is

$$\Delta L = -\log \hat{p}(u|c^{s-1}) + \log \hat{p}(u|c^s). \tag{4}$$

In [5] and [6], the efficiency difference may be stored in the son node c^s. In [4], the efficiency difference of the father compared with each son were added (and could reasonably be stored in the father node c^{s-1}). This way the father is compared to both sons at once. When deciding whether to prefer c^{s-1} or c^s as the coding node for the current, yet unknown, symbol, the sign of the efficiency difference determines the matter (*positive* means that the sons are better than the father). Thus, the coding model is found by the predictive minimum description length principle (PMDL).

We may view the above context selection principle of [4] in a way that is only based on the counts $n_{u|c}$, thereby avoiding the need for storing efficiency differences in the tree. Furthermore, it paves the way to a generalization of father-son to ancestor-descendant comparison.

Let the current full path in the context tree be denoted P and the corresponding maximal size context be denoted $c^{|P|}$. The tree is traversed root to leaf (the depth s of the son node is initially one). Using code lengths (2), the efficiency difference of the father and the two sons leads to the rule that if

$$\ell(n_{0|c^s}, n_{1|c^s}|\delta) + \ell(n_{0|c^{s-1}} - n_{0|c^s}, n_{1|c^{s-1}} - n_{1|c^s}|\delta) \geq \ell(n_{0|c^{s-1}}, n_{1|c^{s-1}}|\delta) \tag{5}$$

then, for the events that were recorded in the context tree, the sons did not do better than their father, and we terminate the context selection declaring the coding context to be $c^{|P|*} = c^{s-1}$. If (5) proved false, we disregard c^{s-1} as the coding context, increment s, and repeat the evaluation of (5). If the recursion did not terminate before depth $|P|$, we define $c^{|P|*} = c^{|P|}$.

We should mention that the choice of growth rule for the context tree has a small effect on the value of the efficiency difference, as the sons are not built at the same time as the father node. As long as we make sure that both father and sons in some sense account for the same events, the precise details are not important. The growth rule that we shall employ for the context tree is described in Section III-B.

An interesting experimental observation described in [5] and [6] is that compression results on grey-scale images are greatly improved if the efficiency difference is initialized by a positive amount, Q, so that there is a bias toward using the sons. Furthermore, in [6] the efficiency difference is clamped to a maximum absolute value.

A. A Baseline Full Path Algorithm Context (FPAC-B)

The above algorithm compares father and son on the context path. We generalize this comparison to the case where an ancestor node is compared to a descendant. Our bid for a modified version of the context algorithm tailored for bilevel image coding is a $|P|$-step algorithm to perform the selection of one of the nodes in the path. We compute the coding context as the PMDL-optimal context in longer and longer subpaths of P, naming the PMDL-optimal contexts c^{1*}, c^{2*}, $\cdots$, $c^{|P|*}$, where $c^{i*} \in \{c^0, c^1, \cdots, c^i\}$. Define initially c^{0*} as c^0, i.e., the root. Initially, let $s = 1$ and compute the code length difference between symbols generated in the ancestor context $c^{(s-1)*}$ on one side and, on the other side, symbols being generated in the descendant context c^s and the complement to context c^s with respect to $c^{(s-1)*}$, i.e., context strings for which $c^{(s-1)*}$ but not c^s is a prefix. If

$$\ell(n_{0|c^s}, n_{1|c^s}|\delta) + \ell(n_{0|c^{(s-1)*}} - n_{0|c^s}, n_{1|c^{(s-1)*}} - n_{1|c^s}|\delta) \geq \ell(n_{0|c^{(s-1)*}}, n_{1|c^{(s-1)*}}|\delta) \tag{6}$$

we declare $c^{s*} = c^{(s-1)*}$, as the descendant and its complement with respect to the ancestor did not do better than the ancestor for the events that were recorded in the context tree. If (6) proved false, we declare $c^{s*} = c^s$. In any event we increment s and repeat the evaluation of (6). After $|P|$ steps we have the desired context, $c^{|P|*}$. We use the fast calculation of (3) to compute the Bernoulli code lengths.

As in [6], we choose to bound $|P|$ by some maximal value, l. This for one thing has the effect that we need not worry about the additional context selection requirements in [3] that has the effect of excluding nodes of unreliable statistics. The context tree may be grown as in [3], [5], or [6], but we use growth by *copy and attenuate* (see Section III-B).

The motivation behind the new context selection is the observation that the conditional probability $p_{u|c}$ in an ancestor node is the weighted sum of the conditional probabilities in a subtree of descendants, and although at a given time the difference between the empirical distributions, $n_{u|c}/n_c$, in an ancestor and a descendant may be quite significant, it is possible that none of the father-son pairs along the path have a significantly different empirical distribution.

The modified context algorithm based on (6) we refer to as the full-path context algorithm (FPAC). The baseline version (FPAC-B) is given above. A faster version is presented next.

B. A Fast Full Path Algorithm Context (FPAC-F)

FPAC may be improved in speed (and compression performance) by introducing four different refinements:

1) occasional checking for the best context;
2) typical prediction as in JBIG;
3) renormalizations of counts (introducing local adaptivity);
4) tree growth by copy and attenuate (see below).

All the refinements serve to increase the speed. We can hope for a compression improvement over FPAC-B due to the local adaptivity. The typical prediction may also yield improvement for some data.

1) *Occasional Checking*. If the source is stationary all probabilities inferred by the node counts converge as t approaches infinity. Therefore, in a particular leaf context, $c^{|P|}$, for t larger than some threshold, FPAC-B always comes out with the same PMDL-optimal node $c^{|P|*}$. For this reason, we can skip the search for $c^{|P|*}$ most of the time using the formerly found PMDL-optimal node $c^{|P|*}$ as the coding node for path $c^{|P|}$. All that is required is that we at each leaf node in the tree store a pointer to that node on its path that was found PMDL-optimal at the last evaluation. A suitable scheme of when to do the search also needs to be defined. Checking occurs when the leaf counts exceed a (dynamic) threshold, $D(c^{|P|})$, and when a pair of sons is appended to the tree.
2) *Typical Prediction (optional)*. To increase encoding and decoding speed of, e.g., scanned text we adopt the (baseline) typical prediction of JBIG [2]. Typical prediction applies to an entire image line. The definition of a typical line is a line that replicates the former line. In our codec the event, *line m is typical/nontypical* is modeled as a first order Markov process in m, using the usual predictor with δ equal to 0.45.
3) *Renormalizations*. Real life data are rarely stationary [2], [9], [10], and algorithms are modified accordingly. We implement local adaptivity by scaling the recorded frequency counts. The procedure consists of occasionally subtracting leaf events from some leaf and all its ancestors. For the sake of simplicity, this renormalization step is combined with the procedure of occasional checking for the best context. A parameter k_1 controls how often to do the full-path search.

FPAC-F does the following for each pixel (on a nontypical line).

Climb the tree to the leaf, $c^{|P|}$
Use for the coding node, $c^{|P|*}$, the node pointed to by the leaf, $c^{|P|}$
If $n_{0|c^{|P|}} + n_{1|c^{|P|}} = D(c^{|P|})$
  if the tree should not grow
    Perform the full path search of (6).
    Save in the leaf, $c^{|P|}$, a pointer to $c^{|P|*}$.
    $D(c^{|P|}) := D(c^{|P|}) + k_1$
    for $i = 0, 1$
      $N_i \triangleq \lfloor 0.5 n_{i|c^{|P|}} \rfloor$
    for $i = 0, 1$ and $k = 0, \cdots, |P|$
      $n_{i|c^k} := n_{i|c^k} - N_i$
  else append sons to $c^{|P|}$ by the growth procedure.
Update the context tree. (7)

A path does not grow if $|P| = l$ or if we have reached the maximal number of nodes, N_{St}, in the context tree (i.e., reached the statistics memory limit). We do not delete nodes. The restrictions on the growth of the tree combined with scaling have a side effect of local adaptivity which is sometimes useful. By (7), the counts in the tree grow as $\sqrt{t}$ instead of t (see [8]).

4) *Copy and Attenuate (optional)*. Generating a new pair of context tree leaves costs time and memory. Therefore, as in [6] we parameterize the growth rule of the context tree so that a node c^{k-1} bears sons when $n_{c^{k-1}}$ equals some threshold T_{ca}. Instead of initializing the counts of the new-born sons by zero, we exploit the assumed dependence between the father node and the sons by initializing the sons with the expected counts as follows:

$$n_{i|c^{k-1}0} \triangleq \left\lfloor \tfrac{1}{2} \cdot n_{i|c^{k-1}} + \tfrac{1}{2} \right\rfloor$$
$$n_{i|c^{k-1}1} \triangleq n_{i|c^{k-1}} - n_{i|c^{k-1}0}. \quad (8)$$

Disregarding the truncation, the initializations correspond to a prior Beta distribution $[Be(n_{0|c^{k-1}a} + \delta, n_{1|c^{k-1}a} + \delta)]$ for the zero-probabilities in context $c^k a$.

1) Universal Coding: In the previous sections, the context algorithm was modified specifically so as to obtain better compression of bilevel images. The theoretical impact of the modifications with respect to universality for binary sources is investigated here. In the theorem below, we prove that FPAC-B is a universal code for an important subclass of tree sources.

We consider a tree source over the alphabet $\{0, 1\}$ with given branch labelings where for given l, the (unknown) conditional probabilities of each size l path c^l satisfy:

$$p(u|c^k) = p(u|c^{k+1}) \quad \text{for} \quad k \geq k(c^l) \quad (9)$$

where the 2^l numbers, $k(c^l) \leq l$, denote unknown constants. Thus, the smallest unique subset, $\mathcal{C}$, of the contexts $\{c^{k(0^l)}, \cdots, c^{k(1^l)}\}$ describes a complete subtree.

Furthermore, we require the ergodic theorem to hold true in each node, c, such that with probability 1 the maximum likelihood estimate of $p(u|c)$, formed by the frequency counts in that node, divided by the true conditional probability, $p(u|c)$, goes to 1 as t approaches infinity.

In the following theorem, we shall assume sufficient memory.

Theorem 1: Let x be an infinite string generated by a binary tree source with restriction on the maximal depth as defined by (9) and satisfying ergodicity of the nodes. Using FPAC-B, for almost all samples x

$$p[c^{|P|*}(t) \in \mathcal{C}] \to 1 \text{ as } t \to \infty \quad (10)$$

and the code length

$$-\frac{1}{t} \log \hat{p}(x^t) \to H(U|\mathcal{C}) \text{ as } t \to \infty. \quad (11)$$

The proof of Theorem 1 is given in the Appendix.

The requirement about multiple ergodicity is (probably) more restrictive than merely requiring the source to be ergodic, but we propose Theorem 1 to hold true also in the latter case.

The maximum depth, l, imposed on context paths makes the algorithm less general and the proof easier than, e.g., for the context algorithm in [3]. In practice, memory is limited and some fixed limit reasonable. Following the theorem, a given node allowance, N_{St}, will bound l to $\lfloor \log N_{St} \rfloor - 1$. In practice it is better to pick a larger l and slowly use up the memory as the context tree grows.

FPAC-B or FPAC-F are not immediately applicable for the encoding of a multialphabet source, but they can be used if

the multialphabet problem is binarized. Compression may be achieved by forming a separate context tree for each bit in the binary decomposition. In encoding the mth most significant bit in the binary decomposition of the tth sample, the context bits would be among the $m-1$ more significant decomposition bits at time t and all the decomposition bits at earlier times. If for a given binary decomposition of a multialphabet source, the components of the decomposed source obey the condition of (9) and are coded using FPAC-B, then the per symbol code length approaches the entropy of the source as t approaches infinity. Modifications of the context algorithm for grey-scale images was presented in [6].

IV. Free Template Coding

Template coding may be perceived as a simplification of codes like the context algorithm. Template coding is more practical because template decoding is faster and much simpler than variable-depth decoding. The compression and encoding speed varies greatly with the effort put into finding the template. It is, therefore, of practical interest to find fast schemes for producing very good templates. The baseline JBIG is a ten-pixel template coder with one free template pixel. The free pixel is essential for efficient compression of periodic halftones. Additional free template pixels may capture the halftone period even better and increase compression. In free template coding, the entire template is freed. The identity of the free template is communicated to the decoder prior to coding the data. For a template of size k, we have to transmit k and the coordinates of the template pixels. We shall not consider this negligible cost in the following.

Template coding with multiple free pixels has been reported in [11] and [12]. Multiple encodings using different templates or simulated encodings using approximations to code length (e.g., conditional entropy) as a selection criterion were used to construct the final template. Both references reported significant improvements in code length compared to coding with a fixed template. In the following section, we present an algorithm for constructing a free template which is based on simulated encodings using precise code lengths as the selection criterion.

A. Greedy Construction of the Free Template

We grow the template by a greedy approach. Once a template pixel is selected, it stays in the template. The kth context bit, C_k, may be chosen among the pixels of the search area of Fig. 2. The $N_a = h \cdot (2w+1) + w$ candidates are denoted $A_1 \cdots A_{N_a}$.

The code length that would result from picking A_j as C_k given the already chosen template pixels C^{k-1} may be calculated as follows:

$$L(x^T) = \sum_{c^{k-1}} \sum_{a_j=0}^{1} \ell(n_{0|c^{k-1}a_j},\, n_{1|c^{k-1}a_j}|\delta). \tag{12}$$

Based on (12), the best pixel (or no pixel) is picked.

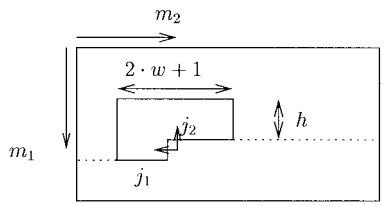

Fig. 2. Search area for context pixels. Definition of absolute and relative coordinates. U has relative coordinates $(j_1, j_2) = (0, 0)$ and absolute coordinates (m_1, m_2).

B. Complexity Reduction

To evaluate (12) for all the N_a candidates we only need a single pass of the data, updating a table of dimensions $2^{k+1} \cdot N_a$ as we go along. In computing (12) we use the fast calculation (3).

Because the memory requirement is exponential in k we choose the number of candidates, N_a, to be inversely proportional to 2^k for k larger than some value k_{sh}. When decreasing the set of candidate pixels, we throw away those pixels that have the largest 1-norm distance to U (see Fig. 1).

The time for building a kth order template is denoted t_s. We have

$$t_s \simeq \sum_{q=1}^{k} T(N_a K_1 + K_2 \cdot q) \tag{13}$$

where K_1 and K_2 are constants with respect to image size T and N_a. The term $K_2 \cdot q$ comes from calculating the context entry c^q of the statistics table. The term $N_a K_1$ comes from updating the counters $(n_{0|c0},\, n_{0|c1},\, n_{1|c0},\, n_{1|c1})$ for each candidate in the search area.

1) Subsampling: The time, t_s, for growing the template can be reduced if we base the choice of the kth context pixel on a smaller number of pixels, $n(k)$, than the total number, T. One way to implement this is to cut out what is supposedly a representative part of the image and use that to grow the template. This procedure is extremely sensitive to instationarities. As a new idea we choose to pick the $n(k)$ pixels by subsampling. We jump from pixel to pixel but encode them (i.e., emulate coding of them) in their correct context. A general description of subsampling in two dimensions is given in [13, ch. 2]. In subsampling, points on the input lattice (m_1, m_2) are associated with points on a subsampling lattice (u_1, u_2). We modify hexagonal subsampling as described in [13] introducing a subsampling factor k_s. Furthermore, we introduce (pseudo)random offsets $(\Delta m_1, \Delta m_2)$ so that the points on the input lattice do not appear in a regular grid, which may interfere with the periodic grid of a halftoned image. Our scaled, randomized hexagonal subsampling takes the form

$$\begin{bmatrix} m_1 \\ m_2 \end{bmatrix} = k_s \cdot \begin{bmatrix} 2 & 1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + \left\lfloor \frac{k_s}{\left\lfloor \frac{5k_s}{4} \right\rfloor} \right\rfloor + \begin{bmatrix} \Delta m_1(u_1, u_2) \\ \Delta m_2(u_1, u_2) \end{bmatrix}. \tag{14}$$

There are $4 \cdot k_s^2$ as many pixels in the input lattice as in the subsampling lattice. The offsets $(\Delta m_1, \Delta m_2)$ are determined by a simple random generator picking one of the $4 \cdot k_s^2$ points associated with each sampling center.

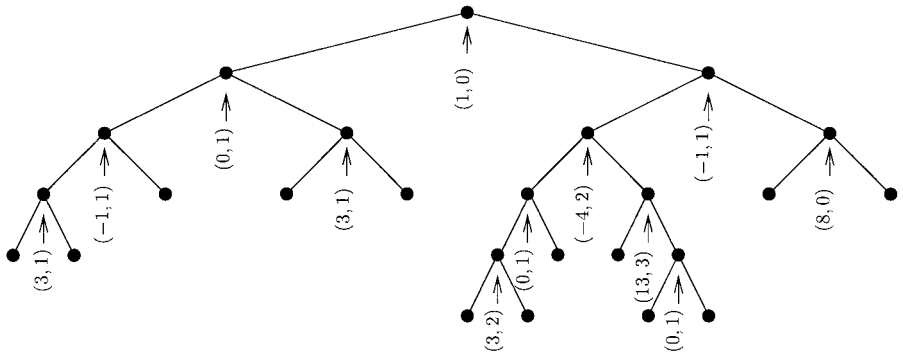

Fig. 3. Free tree for test image c04a200 ($b = 4000$ b). The cost of coding the data is 493 328 b. The cost of coding the structure of the tree is 25 b. To code the 12 split pixels, which are stored in the internal nodes of the tree, we use $12 \cdot 11$ b. The code string for the tree structure is 1 111 000 100 111 100 010 100 100.

TABLE I
A SUBSAMPLING SCHEME, $k_s(k) = \lfloor k_0 \prod_{k=1}^{k} s_f(k) \rfloor$. SUBSAMPLING TAKES PLACE FOR $k_s(k) > 0$

k	1	2	3	4	5	6	7	8	9	>10
s_f	1	1	1	1	$2^{0.25}$	$2^{0.25}$	$2^{0.25}$	0.80	0.75	$2^{0.5}$

The subsampling factor k_s should be chosen so that $n(k)$ is just large enough to allow us to pick a good template pixel. In particular, k_s should be chosen so that the growth of the template is not terminated prematurely. For both reasons, we need to decrement k_s with the template size, k. The subsampling scheme should reflect the expected form of $H(U|C^{k-1}) - H(U|C^k)$, a function that goes rapidly to 0 for growing k. For large k, we attempt to keep the variance of the estimate of $H(U|C^k)$ independent of k. For a specific node, c^{k-1}, and its one son, c^k, the maximum likelihood estimates, $\hat{p}(u|c^{k-1})$ and $\hat{p}(u|c^k)$, are asymptotically normal. If $p(C_k = 0|c^{k-1}) = \frac{1}{2}$ we shall need twice the number of events at depth k because we treat nodes $c^{k-1}0$ and $c^{k-1}1$ as independent. Hence, for large k we use $k_s(k)/k_s(k-1) = \sqrt{2}$. For smaller k, i.e., larger $H(U|C^{k-1}) - H(U|C^k)$, we can relax the equal-variance requirement. By some experimentation we choose the subsampling scheme of Table I. The constant, k_0, of the subsampling scheme is picked experimentally as $k_0 = \max(\lfloor \sqrt{T \cdot 2^{-18}} + \frac{1}{2} \rfloor, 5)$. The max function ensures that a coding scheme where we select only few context pixels will not be dramatically slowed down by the template selection procedure.

C. Hybrid Coding

Template coding may be perceived as coding with the leaves of a perfectly balanced tree. If the number of template pixels k is large, most leaves will contain insufficient statistics for sharp prediction. To improve compression an algorithm like FPAC-B or FPAC-F may be used to determine the coding depth. With variable depth coding, it becomes less critical to find all the template pixels by greedy searches.

We construct a hybrid coder where the ordering of pixels is determined in three steps.

1) Let the first r pixels be chosen by default (1-norm).
2) Let the next $k - r$ context pixels be found in $k - r$ searches.
3) Let the last $l - k$ pixels be chosen by default.

The search procedure for the $k - r$ pixels is identical to the search procedure by which we grow a free template. With the hybrid coder, we first communicate the pixel ordering and then run FPAC-B or FPAC-F. 1-norm distance is used as default ordering throughout the paper. In a few places comparison with 2-norm (Fig. 1) is presented.

The variable depth of hybrid coding will usually improve compression also when all the pixels are picked by searching. This is because the resulting template is a compromise between unnecessary initialization cost in some parts of the corresponding balanced tree and forfeited opportunities to reduce conditional entropy in other parts of the tree.

V. FREE TREE CODING

Aiming at the best compression, we use a coding method that is more strongly adapted to the data than coding with a free template or the context algorithm. The context of U is still defined by the color of a number of context pixels in a fixed spatial relationship to U, but now the ith context pixel depends on the values of the $i - 1$ first context pixels, c^{i-1}. The organization of the context is stored in the nodes of a context tree, which also contain frequency counts that grow during coding. We call the pixel C_i, whose value defines c_i in the particular path c^{i-1}, the split pixel of node c^{i-1}. The coding depth of the free tree does not change as in the context algorithm. We always code with a leaf. The free tree is found on the encoding side and transmitted to the decoder before transmitting the data. An (unrealistically small) example of a free tree is given in Fig. 3. The coordinates (j_1, j_2) of the split pixels are relative to the position of U (Fig. 2).

Free tree coding has been used by Nohre [14] for coding grey-scale images. The multialphabet problem was binarized as outlined in Section III-B1 and the individual binary decisions coded with a free tree. Nohre also gave an algorithm for constructing a free tree for bilevel coding.

A. Construction of the Free Tree—Greedy Build-Up

The problem in free tree coding is how to construct the tree. In the bilevel case, Nohre used a very small number (24) of candidates for split pixels, N_a. The small number enabled him to construct a balanced tree of depth N_a and rearrange the tree following the concept that an optimal tree cannot consist of

nonoptimal subtrees. This procedure does not take into account the cost of transmitting the tree. For most bilevel images, N_a should be large ($\sim 10^3$), as pixels of good predictability may lie far from U. We choose a large N_a and consequently, due to complexity, limit ourselves to a series of greedy searches when constructing the tree. Because of the large number of potential split pixels, the cost of coding the tree is potentially large and we must incorporate the tree cost in the procedure by which we grow the tree.

Assume that the free tree is defined as far as to the $(i-1)$th context bit in some path of the tree. We wish to select the best context pixel, C_i, given the specific context c^{i-1}. Pixel C_i may be chosen among the candidates in the area depicted in Fig. 2. The code length that would result from picking a particular candidate pixel A as C_i may be calculated as follows:

$$L(x|c^{i-1}, A) = \sum_{a=0}^{1} \ell(n_{0|c^{i-1}a}, n_{1|c^{i-1}a}|\delta). \qquad (15)$$

The notation $x|c^{i-1}$ means that we only consider the events U for which the size $i-1$ context equals the specific context c^{i-1}. Equation (15) gives us the means to compare different candidates for C_i in context c^{i-1}. Again, we pick the best pixel (or no pixel), but this time in each node of the tree.

A split pixel should only be defined if by doing so the incremental tree cost is smaller than the reduction in the code length of the data given the larger tree. The free tree is unbalanced, so besides the need to code the identity of the split pixels, we need to code the tree structure itself. For the latter job, we adopt the procedure of [14]. We create a binary string y to describe the anonymous tree structure where y_k equals zero if node no. k is a leaf. The nodes of the tree are visited one by one, depth first, starting at the left branches. Consider the example of Fig. 3—the nodes are visited as follows (left branch is zero and right branch is one): root $\rightarrow$ 0 $\rightarrow$ 00 $\rightarrow$ 000 $\rightarrow$ 0000 $\rightarrow$ 0001 $\rightarrow$ 001 $\rightarrow$ 01 $\rightarrow$ 010 $\rightarrow$ 011 $\rightarrow$ 1 $\rightarrow$ 10 $\rightarrow$ 100 $\rightarrow$ 1000 $\rightarrow$ 10 000 $\rightarrow$ 100 001 $\rightarrow$ 1001 $\rightarrow$ 101 $\rightarrow$ 1010 $\rightarrow$ 1011 $\rightarrow$ 10 110 $\rightarrow$ 10 111 $\rightarrow$ 11 $\rightarrow$ 110 $\rightarrow$ 111.

If the number of leaves in the tree is denoted by $z+1$, then the total number of nodes is $2z+1$ and the number of internal nodes is z. Using 1 b for each node, we code y with $2z+1$ bits or approximately 2 bits per internal node. The identity of a split pixel may be indexed with $\log \lceil |w+h(2w+1)| \rceil$ bits. A suitable prefix code [8] realizes this cost. For $h = w = 2^j$, the prefix code codes a split pixel with $\simeq 2j+1$ b. The total cost of changing a leaf into an internal node (appending sons to it) is, therefore, approximately $\simeq 2j+3$ bits.

By using the growth rule that $L(x|c^{i-1}) - L(x|c^{i-1}, A)$ must be larger than the incremental tree coding cost, $2j+3$ b, in order to grow the tree adding A, the tree will not be too large. As the cost of expanding the tree may surely be smaller than $2j+3$ b, the tree might be too small. A bit of experimentation, simulating (optimistically) reduced tree coding costs does not affect the code length of the data much, so what may be gained by a more clever tree coding scheme is basically a reduction of the tree coding costs, which constitute only about 8% of the total code length.

In this paper, we parameterize the growth decision using an incremental tree cost b (defaulting to $2j+3$ bits). The parameterization allows a trade-off between compression and tree construction time. Most of the compression is usually obtained with the first few branches of the big coding tree. The example of Fig. 3 illustrates this point: a free tree with as few as 13 leaves can compress test image $c04a200$ by a factor of 8.32.

Above, we considered free tree coding as a two-step procedure where we transmit the tree first and the data next. Empirical results in [8] show that a free tree may be constructed truly predictively in such a way that the compression performance on large and difficult images will be better than what can be obtained with FPAC-B and FPAC-F. The good results do not extend to all images, because initially the one-pass free tree is too small to give sharp prediction.

VI. Compression and Complexity Results

The test images used in this paper are mainly the Stockholm (JBIG) test set [15] and the (ambiguous) CCITT test set [16] supplemented by a80c [17], a halftone test image for the graphic arts in Scandinavia. The Stockholm images are marked by initial letter *s* and the CCITT images by initial letter *c*. The images contain scanned text, line art, and halftones. The troublesome halftones are the mixture image s06a400, the error diffusion s09a400, and the clustered dot periodic halftone a80c. The main groups are text and line drawings (TL) and halftones (H). Text and line art are further divided in computer generated (C) and scanned (S). The latter being split into little or medium amount of printing (l) and dense printing (d). The halftoned images are marked whether they are periodic (P) or not, and whether they are dither (D) or not. We make these distinctions because the compression and the comparisons differ by the nature of the material.

A. Compression Results

We distinguish between truly predictive algorithms with default pixel ordering (fast one-pass algorithms, Table II) and multipass algorithms where the pixel ordering is established fully or in part prior to coding the data (Table III). It is a drawback for the multipass algorithms that they require enough memory to buffer the entire image and introduce a delay. The JBIG recommendation [2] suggests how to position the adaptive template pixel on the fly, based on a simple correlation between U and each candidate pixel, A, over the first 2000 pixels of an image stripe (a horizontal slice of the image). By the sparse and uneven choice of representative pixels and by disregarding the influence of the nine fixed template pixels in the selection of A, there is a risk of picking a nonoptimal adaptive pixel. JBIG compression results for the test images (except a80c) are given in [18]. Considerably better compression results are obtained if we use the free template algorithm to determine the position of the adaptive template pixel given the nine fixed template pixels of the three-line template. The improvement is 20% on halftones, 10% overall compared to [18]. JBIG with this optimized pixel selection is our reference for the new multipass algorithms.

TABLE II
ONE-PASS CODING COMPARISON OF JBIG AND FPAC-F. TYPICAL PREDICTION IS USED FOR BOTH

			JBIG		FPAC-F	
Ordering			default, 3-line template		default, 1-norm	
Coding depth			Fixed, image independent		Adaptive, ($N_{St} = 87381$,) $l = 24$, ($k_1 = 10$) ($T_{ca} = 10$)	
Image	Rows	Cols	bytes	t_c (s)	bytes	t_c (s)
s08a400	3040	3072	6198	5.8	4952	51.7
s10a400	5856	4096	13785	14.8	10504	127.5
$\sum$TL,C			19983	20.6	15456	179.2
s01a400	4352	3072	13860	5.8	12012	44.9
s02a400	4352	3072	15553	6.6	14488	54.3
s05a400	4352	3072	34576	8.4	29384	80.9
s07a400	4352	3072	24132	8.8	21556	85.3
c01a200	2376	1728	14809	2.1	13996	23.1
c02a200	2376	1728	8714	2.9	7780	29.1
c03a200	2376	1728	22107	3.0	19948	36.5
c05a200	2376	1728	25988	3.1	23624	39.2
c06a200	2376	1728	12681	2.9	11180	31.0
c07a200	2376	1728	56477	3.2	51544	53.3
c08a200	2376	1728	14373	3.0	12948	31.5
$\sum$TL,S,l			243270	49.8	218460	509.1
s03a400	4352	3072	146774	11.4	128072	141.6
c04a200	2376	1728	54523	3.0	48364	47.5
$\sum$TL,S,d			201297	14.4	176436	189.1
$\sum$TL			464550	64.2	410352	698.2
s04a400	2048	3072	130213	5.7	63088	67.0
s04b400	2048	3072	158817	5.8	56452	69.3
s04c400	2048	3072	129965	5.1	39264	63.4
s04d400	2048	3072	120452	5.6	39652	62.0
$\sum$HP,D			539447	22.2	198456	261.7
a80c	6144	4864	695987	27.2	610944	318.9
$\sum$HP			1235434	49.4	809400	580.6
s06a400	4352	3072	256145	11.7	156864	151.7
s09a400	1024	1024	74414	1.2	73492	39.0
$\sum$H			1565993	62.3	1034396	771.3
$\sum$TL+H			2030543	126.5	1444748	1469.5

Relative improvements in this paper are given as the increase in compression factor. The compression factor is defined as uncompressed data size divided by code length.

The arithmetic coder of JBIG is a QM-coder, for all other results the arithmetic coder is the one described in [8] and [1]. With the latter coder the difference between the calculated code length and the file count is approximately 100 b for most images, a negligible amount. The images are zero-padded when being coded. For all algorithms, the parameters applied are all specified in the respective table and figure texts.

1) One-Pass Compression Results: FPAC-F (Section III-B) gives slightly (~2%) better results on the test images than FPAC-B (Section III-A). Both algorithms outperform default template JBIG for all images. Table II gives a comparison of fixed template JBIG and FPAC-F. Overall, FPAC-F provides a 41% higher compression factor than JBIG. The gain comes primarily from a better compression of the halftones (51% totally), whereas the improvement is 13% on text and line art.

The default ordering of pixels was chosen to be 1-norm rather than 2-norm due to the performance on periodic, dithered halftones (172% better than JBIG) and on text. Assuming a halftone period of 5 or 6 and ordering by 2-norm, the periodicity will not be captured for any reasonable value of the maximal context tree depth, l (see Fig. 1).

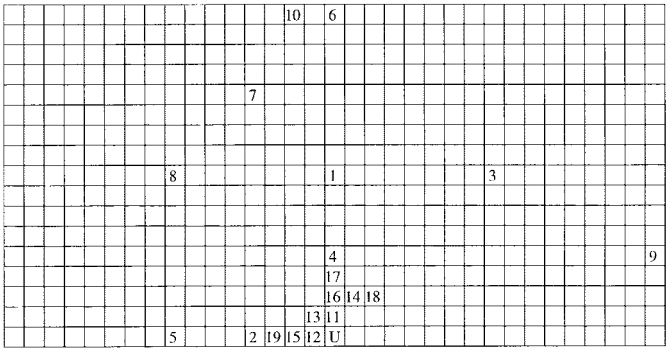

Fig. 4. Template for test image s04a400 found by the free template algorithm. Initial search area: $(h, w) = (16, 16)$, limitation of search area (1-norm) for $k > k_{sh} = 12$. Memory usage is 16 MB.

A test was conducted to evaluate the context selection based on ancestor-descendant comparison as in FPAC against basing the selection on comparison of the father and its two sons as in the context algorithm [4]. For this purpose, FPAC-B was compared with a coder that is identical to FPAC-B in every respect except that it uses the father-son context selection rule of (5). With this father-son selection rule [4], the tree is climbed root-to-leaf and the coding node is the first father that is better than his son. This is a hesitant climb rule. To improve it, we also imitate the initialization of [5], [6] by adding Q bits to the right-hand side of (5) to bias the result to disfavor the father model. Fig. 5 shows the result on the test set as a function of Q. If Q exceeds the maximal model cost difference, $\simeq \frac{1}{2} \log T$, between the father model and the sons model, nothing is left of the PMDL idea—the coder will choose the leaf as the coding context. Because FPAC-B uses bounds on the context tree, a leaf coder can perform quite well as long as the bounding parameters are neither very small nor very large. On the test images, FPAC-B gives the best results. Using a reasonable choice of bias ($Q = 5$), the father-son selection outperforms JBIG, whereas the unbiased father-son context selection [4] here performs worse than JBIG. As seen in Fig. 5, even with the best choice of Q the father-son algorithm is not better than a much less complex coder where a leaf is preferred deterministically ($Q = \infty$). Without bounds on the tree size as in the actual context algorithm, leaf coding would not give good results and the bias Q should be chosen differently. Other versions of the context algorithm may give better results than the father-son results reported here.

In Fig. 5, FPAC-B is 6% better than the leaf coder, and is 14% better on text and line art. For small values of l and N_{St}, FPAC-B performance drops somewhat and the leaf coder performance drops only slightly due to improved coding on text and line art, so that FPAC-B is only 2% better overall. Another (faster) low-complexity coder of interest is an l-pixel template coder. Fig. 6 displays the results. With $l = 16$, the memory usage is still moderate and the template coder is considerably better than JBIG.

2) Multipass Compression Results:

The free tree algorithm (Section V) produces excellent compression results (Table III) for all image types, 46%

TABLE III
MULTIPASS RESULTS. FREE TEMPLATE AND HYBRID CODING: LIMITATION OF SEARCH AREA (1-NORM) FOR $k > k_{sh}$. HYBRID CODING: DEFAULT ORDERING: 1-NORM. t_s DENOTES THE PART OF THE CODING TIME, t_c, THAT WAS USED TO PICK THE $k - r$ FREE TEMPLATE PIXELS, WHERE $k \leq k_{\max}$. FPAC-F WAS RUN WITH $l = 24$, ($k_1 = 10$, $T_{ca} = 0$). JBIG: FREE TEMPLATE SELECTION OF THE ADAPTIVE TEMPLATE (NOT INCLUDED IN THE TIMINGS). TYPICAL PREDICTION IS APPLIED EXCEPT FOR THE FREE TREE ($h = w = 16$ FOR ALL RESULTS)

	JBIG		**Free template + hybrid coding**					**Free tree**
Ordering	$r = 9$ fixed 1 adaptive		$r = 0$, max. 20 adaptive $k_{sh} = 12$ (~ 16 Mbytes)		$r = 2$, max. 4 adaptive $k_{sh} = 6$ ($\sim \frac{1}{4}$ Mbytes)			($b = 11$)
Subsampling	No		No		Yes			No
Coding depth	Fixed, image independent		Fixed, image dependent	Adaptive, FPAC-F ($N_{St} = 171072$)	Adaptive, FPAC-F ($N_{St} = 87381$)			Leaf
Image	bytes	t_c (s)	bytes	bytes	bytes	t_c (s)	t_s (s)	bytes
s08a400	6206	5.9	4808	4704	5000	67.5	13.3	3807
s10a400	14018	14.7	10728	9716	10032	148.4	13.0	8198
$\sum$TL,C	20224	20.6	15536	14420	15032	215.9	26.3	12005
s01a400	13731	5.8	11936	11384	12084	53.5	6.0	10287
s02a400	15485	6.6	14388	13928	14332	68.8	12.2	12718
s05a400	34256	8.5	29308	28352	29596	105.2	18.6	26976
s07a400	23086	8.8	21572	20948	21556	112.9	20.1	19180
c01a200	15022	2.1	14568	13888	13844	36.1	5.6	12326
c02a200	8696	3.0	7608	7372	7796	38.0	6.5	6875
c03a200	22207	3.1	20364	19804	20028	53.7	10.3	17659
c05a200	26187	3.1	23816	22956	23532	56.2	10.6	20468
c06a200	12618	2.9	11284	10932	11176	48.1	9.6	10025
c07a200	56168	3.2	53144	51900	51777	62.8	10.6	48065
c08a200	14539	3.0	13076	12604	12920	49.5	10.3	11471
$\sum$TL,S,l	241995	50.1	221064	214068	218641	684.8	120.4	196050
s03a400	146782	11.5	131292	126832	128240	171.3	28.4	117018
c04a200	55120	2.9	47204	44828	46848	60.9	9.8	37019
$\sum$TL,S,d	201902	14.4	178496	171660	175088	232.2	38.2	154037
$\sum$TL	464121	64.5	415096	400148	408761	1132.9	184.9	362092
s04a400	58150	5.4	41216	41152	43160	94.7	21.7	38773
s04b400	56588	5.4	37856	37672	37464	101.2	22.1	34581
s04c400	52545	4.8	35112	34920	34740	90.6	21.8	32306
s04d400	48082	5.3	34644	34480	34928	91.0	21.4	33141
$\sum$HP,D	215365	20.9	148828	148224	150292	377.5	87.0	138801
a80c	519718	27.1	382088	374776	402120	335.6	25.9	316205
$\sum$HP	735083	48.0	530916	523000	552412	713.1	112.9	455006
s06a400	212934	11.5	154504	151552	154596	181.0	27.1	131852
s09a400	73086	1.2	69156	67352	72480	39.1	3.9	68010
$\sum$H	1021103	60.7	754576	741904	779488	933.2	143.9	654868
$\sum$TL+H	1485224	125.2	1169672	1142052	1188249	2066.1	328.8	1016960

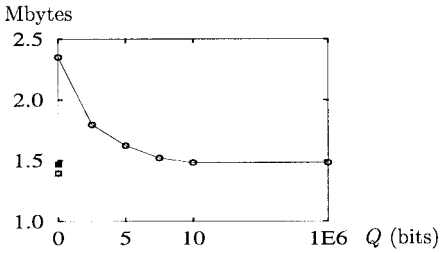

Fig. 5. Comparison between FPAC-B (nonfilled square) and "FPAC-B" using a context selection rule similar to that of the context algorithm [4] with bias Q (circle). Default pixel ordering (1-norm). Parameters for both coders are $l = 31$, $N_{St} = 349\,525$, $T_{ca} = 0$. The pure PMDL context selection [4] corresponds to $Q = 0$. Bias $Q = \infty$ corresponds to a leaf coder. The filled square displays the FPAC-B total with $l = 24$, $N_{St} = 87\,381$, $T_{ca} = 0$.

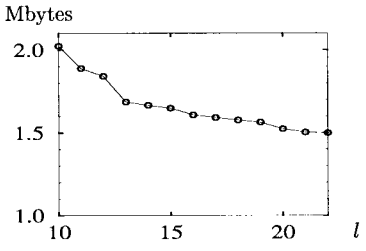

Fig. 6. Overall compression results with l-pixel template coding with 1-norm ranking. Typical prediction is on. A simple form of JBIG local adaptivity is applied.

better than optimized JBIG totally (and 100% better than default JBIG). Again, most is gained on halftones (55%). The compression results for the halftones are, to our knowledge, better than anything in the literature. For stochastic halftones, the context tree should be large but not necessarily very flexible. Our otherwise second best algorithm—the hybrid algorithm (Section IV-C) where the free template search determines the pixel ordering up to some order k—seems to be a better choice for these images. Our sample in the test set, s09a400, is coded best with the hybrid algorithm where the default pixel ordering is by 2-norm (the natural choice for stochastic images). With the parameter setting $N_{St} = 87\,381$,

$r = 0$, $k_{\max} = 20$, $T_{ca} = 0$, $k_1 = 2$, and no subsampling, the code length becomes 65 968 bytes—the best result on this image. Finding only few context pixels by searching, the encoding complexity of the hybrid algorithm is reduced very much. With maximum 4 adaptive pixels and a template size of 16, the loss in compression by this partly free template algorithm compared to the hybrid algorithm is 7% overall (6% if we apply a simple form of JBIG adaptivity to the template coder).

If the full template is found by searching (Fig. 4), we may code with leaves (i.e., use template coding) at almost no cost in compression performance. This is free template coding (Section IV). Decoding memory is saved by this choice and decoding speed is increased.

The free template algorithm and the hybrid algorithm of moderate complexity (Table III) are approximately 25% better than optimized JBIG overall. In particular, the halftones are coded better. All the multipass algorithms perform better on periodic, clustered dot halftones as used in the graphic arts if we use an increased search area, say, $(h, w) = (32, 32)$. With the larger search area the free tree compresses a80c to 311 469 bytes. The results on the text images show that the margin to JBIG is highly increased with computer-generated material. To investigate the performance of our algorithms on computer generated halftones, we created a number of high-resolution clustered dot halftones of the Lena image with the threshold halftoning algorithm described in [17]. The hybrid algorithm with $r = 2$, $k_{\max} - r = 4$, $l = 24$, $h = 32$, $w = 32$ compressed these images 80% better than optimized JBIG. The free tree with $h = 32$, $w = 32$, $b = 13$ compressed them 144% better than optimized JBIG.

We have focused on halftones, where the most seems to be gained. Another reason is that the more specialized soft pattern matching (SPM) techniques [19] as a preprocessing offers improved compression for text. SPM plays the role of enabling context pixels to be placed around that same spot in a matching letter as U covers in the current letter.

B. Complexity

Timing results for the algorithms of moderate complexity are given in Tables II and III. t_c denotes the time for compressing an image. The timing results are obtained with algorithms implemented in C (using the UNIX timex command) on an HP 9000, series 755 computer. With default pixel ordering FPAC-F is approximately 12 times slower than JBIG in the given software implementations. The hybrid algorithm with maximum 4 adaptive pixels is 16 times slower than default template JBIG in our implementations. The throughput of this algorithm on the chosen platform is 100 kpixels/s. Speed can be improved by replacing FPAC-F by template coding. With maximum 4 adaptive pixels and a template size of 16 the algorithm is four times slower than default template JBIG (in the implementations we have). For FPAC, the maximal context tree depth, l, significantly influences compression time, because the tree must be climbed for most if not all pixels. In our implementation of FPAC-B and FPAC-F, each context tree node holds six integers of 4 bytes each, so that $N_{St} = 87\,381$ corresponds to a memory requirement of 2 megabytes. The memory requirement may easily be lowered with the same number of nodes but at some cost in compression time. Small values of l and N_{St} were chosen in Tables II and III.

VII. Conclusion

We have considered a number of general purpose schemes of predictive coding for bilevel image compression.

Two modified versions of the context algorithm, FPAC-B and FPAC-F, have been presented. FPAC-B was proven universal on the important class of tree sources (defined by the chosen pixel ordering) for which the maximal depth of the tree is not larger than a predefined value, l. Both algorithms give good results and robust performance on the test images. FPAC-F is much faster than other versions of the context algorithm. With default pixel ordering the compression factor of FPAC-F is 41% better than JBIG over a large test set. The largest gain is for dithered periodic halftones (172% better than JBIG).

Choosing a good ordering of the context pixels has great influence on compression, especially for periodic images. The context pixel selection may be performed using greedy selection and multipass coding. Our best multipass algorithm (the free tree algorithm) choosing the ordering most minutely, compresses periodic halftones 78% better than FPAC-F with default pixel ordering (42% overall). Free tree coding produces superior compression results for all types of images (except, perhaps, stochastic halftones), but the encoding is too slow for other purposes than bench marking. Free template coding gives substantial improvements over optimized JBIG for pure halftones and mixture images and moderate improvements for images of dense printing. For the optimized JBIG, the JBIG adaptive template was chosen with the free template algorithm, thereby improving the performance on periodic halftones substantially. Basing the greedy template selection on image pixels in a randomized, hexagonal grid and gradually decreasing the search area for the template pixels, the template selection becomes feasible for software implementation on a general purpose computer. Free template decoding speed is of the same order of magnitude as JBIG decoding speed. Using the free template algorithm to (partly) define an ordering of pixels and letting FPAC-F determine the coding depth, the compression performance is increased a little at the cost of complexity and of increased decoding and encoding time. Stochastic halftoned images were coded best by this technique (hybrid coding). By using fewer minutely determined template pixels in the ordering, the algorithm becomes an attractive compromise of compression performance and complexity. The compression factor is totally 25% better than that of optimized JBIG. In the given software implementations, the hybrid algorithm is one order of magnitude slower on the encoding side than default template JBIG. By replacing FPAC-F by template coding, speed is greatly improved at a moderate loss in compression performance.

Both one-pass and multipass algorithms perform comparatively better with high-quality images. For computer generated images (compared to scanned images), the percentual gain over JBIG is doubled or tripled.

APPENDIX

Proof of Theorem: The proof of Theorem 1 is quite analogous to the proof in [3]—it falls in two parts: In Part 1, we prove that for $t \to \infty$ with probability 1, the context c^{k+s} (where s is a positive integer) will be preferred to c^k by FPAC-B if $p(u|c^{k+s}) \neq p(u|c^k)$, which means that we will choose a coding node which at least has depth $k(c^l)$. As we have limited depth, this accounts for FPAC-B being a universal coder. In Part 2, we prove that for $t \to \infty$ with probability 1 the context $c^{k(c^l)+s}$ (where s is a positive integer) will not be used as the coding node. This together with Part 1 proves the remainder of the theorem.

To prove the theorem, we need a lemma to state that the effect of choosing a particular value of δ in a Bernoulli model of almost any binary string has an effect on its code length, which is of magnitude $O(1)$ for its length approaching infinity. For $t \to \infty$, we will with probability 1 have $n_0 > N \wedge n_1 > N$ [see (3)] as long as all conditional probabilities, $p(u|c)$, are greater than zero. To include degenerate sources in the proof is trivial but will decrease its legibility. The code length $\ell(n_0, n_1|\delta)$ will thus be approximated by $\hat{\ell}(n_0, n_1|\delta)$ where

$$\begin{aligned}\hat{\ell}(n_0, n_1|\delta) = &(n_0 + n_1 + 2\delta - \tfrac{1}{2}) \log(n_0 + n_1 + 2\delta) \\ &- (n_0 + \delta - \tfrac{1}{2}) \log(n_0 + \delta) \\ &- (n_1 + \delta - \tfrac{1}{2}) \log(n_1 + \delta) + \xi_1 \end{aligned} \tag{16}$$

where $\xi_1 = -\log[\sqrt{2\pi}\Gamma(2\delta)/\Gamma^2(\delta)]$. By the Taylor expansion, $\ln(1+x) = x + O(x^2)$, we get the following lemma.

Lemma A.1:

$$\begin{aligned}\hat{\ell}(n_0, n_1|\delta) = &n_0 \log \frac{n}{n_0} + n_1 \log \frac{n}{n_1} + \frac{1}{2} \log n \\ &+ \xi_1 - (\delta - \frac{1}{2}) \log \frac{n_0 n_1}{n^2} + O\left(\frac{1}{n}\right) \end{aligned} \tag{17}$$

$$= nH_p\left(\frac{n_0}{n}\right) + \frac{1}{2} \log n + O(1) \tag{18}$$

where $n = n_0 + n_1$ and $H_p(q)$ denotes the binary entropy function $-q \log q - (1-q) \log (1-q)$.

Define the shorthand $c = c^{(s-1)*}$. To ease notation further, define a stochastic variable, Y, such that $c^{(s-1)*}y = c^{(s-1)*}0 = c^s$ and $c^{(s-1)*}y = c^{(s-1)*}1$ denotes the compliment of c^s with respect to $c^{(s-1)*}$. Let $n_{\cdot|\cdot}$ refer to $n_{u|cy}$. In this notation, $n_{i|c1} = n_{i|c^{(s-1)*}} - n_{i|c^s}$ is a shorthand of the middle code length in (6). Let $n_{\cdot\cdot}$ refer to n_{cy}, and let $n_c = n_{c0} + n_{c1}$, as follows.

Part 1:

$$\begin{aligned}\Delta L = &\hat{\ell}(n_{0|c0}, n_{1|c0}|\delta) + \hat{\ell}(n_{0|c1}, n_{1|c1}|\delta) \\ &- \hat{\ell}(n_{0|c}, n_{1|c}|\delta) \end{aligned} \tag{19}$$

$$\begin{aligned}= &n_{c0} H_p\left(\frac{n_{0|c0}}{n_{c0}}\right) + n_{c1} H_p\left(\frac{n_{0|c1}}{n_{c1}}\right) \\ &- n_c H_p\left(\frac{n_{c0}}{n_c}\right) + O(\log n_c) \end{aligned} \tag{20}$$

$$\begin{aligned}= &n_c[p(Y=0|c) H(U|c, Y=0) \\ &+ p(Y=1|c) H(U|c, Y=1) - H(U|c) + \epsilon_{n_c}] \\ &\text{with prob. 1 for } t \to \infty. \end{aligned} \tag{21}$$

In the last equation, we used the ergodic property, assumed valid for all contexts. The expression ϵ_{n_c} is a function that goes to zero as n_c approaches infinity. The constant $p(Y = 0|c)H(U|c, Y=0) + p(Y=1|c)H(U|c, Y=1) - H(U|c)$ is nonpositive and zero only when $p(u|cy) = p(u|c)$ for all values of u and y. Hence, $\Delta L < 0$ with probability 1 for $t \to \infty$.

Part 2: Define $\hat{p}(y|c) = n_{cy}/n_c$ and $\hat{p}(u|cy) = n_{u|cy}/n_{cy}$ and $\hat{p}(u|c) = n_{u|c}/n_c$. For $s > k(c^l)$ we have

$$\begin{aligned}\Delta L = &\hat{\ell}(n_{0|c0}, n_{1|c0}|\delta) + \hat{\ell}(n_{0|c1}, n_{1|c1}|\delta) \\ &- \hat{\ell}(n_{0|c}, n_{1|c}|\delta) \end{aligned} \tag{22}$$

$$\begin{aligned}= &n_c\left[\hat{p}(Y=0|c) H_p\left(\frac{n_{0|c0}}{n_{c0}}\right) + \hat{p}(Y=1|c) \right. \\ &\left. \cdot H_p\left(\frac{n_{0|c1}}{n_{c1}}\right) - H_p\left(\frac{n_{0|c}}{n_c}\right)\right] + \frac{1}{2} \log n_{c0} \\ &+ \frac{1}{2} \log n_{c1} - \tfrac{1}{2} \log n_c + O(1) \end{aligned} \tag{23}$$

$$\begin{aligned}= &-n_c \sum_{y=0,1} \hat{p}(y|c) \sum_{u=0,1} \hat{p}(u|cy) \log \frac{\hat{p}(u|cy)}{\hat{p}(u|c)} \\ &+ \frac{1}{2} \log \left[n_{c0}\left(1 - \frac{n_{c0}}{n_c}\right)\right] + O(1). \end{aligned} \tag{24}$$

Defining $\theta(t, c, y) \triangleq \hat{p}(U=0|c) - \hat{p}(U=0|cy)$ we get $[\hat{p}(U=0|cy)/\hat{p}(U=0|c)] = 1 - [\theta(t, c, y)/\hat{p}(U=0|c)]$ and $[\hat{p}(U=1|cy)/\hat{p}(U=1|c)] = 1 + [\theta(t, c, y)/\hat{p}(U=1|c)]$. By the Taylor expansion, $\ln(1+x) = x - \frac{1}{2}x^2 + O(x^3)$, we get

$$\begin{aligned}\Delta L = &-n_c \sum_{y=0,1} \hat{p}(y|c) \frac{\ln 2}{2} \\ &\cdot \{[\hat{p}(0|c)^{-1} + \hat{p}(1|c)^{-1}]\theta^2(t, c, y) + O[\theta^3(t, c, y)]\} \\ &+ \frac{1}{2} \log \left[n_{c0}\left(1 - \frac{n_{c0}}{n_c}\right)\right] + O(1). \end{aligned} \tag{25}$$

As the maximal depth of the tree is bounded by a constant, l, and as the node counts in the tree are bounded from below by a function that is proportional to t we can repeat the remaining steps of the proof in [3] with the result $\Delta L > 0$ with probability 1 for $t \to \infty$.

The use of copy and attenuate initialization of counters only offsets the counters by a constant and does not change the proof.

REFERENCES

[1] J. Rissanen, *Stochastic Complexity in Statistical Inquiry.* Singapore: World Scientific, 1989.

[2] JBIG, "Progressive bi-level image compression," *ISO/IEC Int. Standard 11544*, 1993.

[3] J. Rissanen, "A universal data compression system," *IEEE Trans. Inform. Theory*, vol. IT-29, pp. 656–664, Sept. 1983.

[4] ———, "Complexity of strings in the class of Markov sources," *IEEE Trans. Inform. Theory*, vol. IT-32, pp. 526–532, July 1986.

[5] ———, "Noise separation and MDL modeling of chaotic processes," in *From Statistical Physics to Statistical Inference and Back*, P. Grassberger and J.-P. Nadal, Eds. Boston, MA: Kluwer, 1994.

[6] J. Rissanen, M. Weinberger, and R. Arps, "Applications of universal context modeling to lossless compression of grey-scale images," *IEEE Trans. Image Processing*, vol. 5, pp. 575–586, Apr. 1996.

[7] J. Rissanen, "Fisher information and stochastic complexity," *IEEE Trans. Inform. Theory*, vol. 42, pp. 40–47, Jan. 1996.

[8] B. Martins, "Lossless compression of digital images," Ph.D. dissertation, Tech. Univ. Denmark, Lyngby, Feb. 1996.
[9] P. G. Howard, "The design and analysis of efficient lossless data compression systems," Ph.D. dissertation, Dept. Comput. Sci., Brown Univ., Providence, RI, June 1993.
[10] R. N. Williams, *Adaptive Data Compression.* Boston, MA: Kluwer, 1991.
[11] S. Forchhammer and U. Skands, "Adaptive compression of bi-level halftone images using the JBIG arithmetic coder," Tech. Rep. TR IT-135, Tech. Univ. Denmark, Lyngby, Feb. 1993.
[12] R. B. Arps, T. D. Friedman, and R. C. Pasco, "Optimizing models for data compression using the MDL principle and stochastic complexity," in *Proc. 1990 Picture Coding Symp.*, Cambridge, MA, Mar. 1990, Session 5.6.
[13] J. W. Woods, *Subband Image Coding.* Boston, MA: Kluwer, 1991.
[14] R. Nohre, "Some topics in descriptive complexity," Ph.D. dissertation, Linköping Studies Sci. Technol., Linköping Univ., Linköping, Sweden, 1993.
[15] R. B. Arps and T. K. Truong, "Comparison of international standards for lossless image compression," *Proc. IEEE*, vol. 82, pp. 889–899, June 1994.
[16] R. Hunter and A. H. Robinson, "International digital facsimile standard," *Proc. IEEE*, vol. 68, pp. 854–867, July 1980.
[17] S. Forchhammer and K. S. Jensen, "Data compression of scanned halftone images," *IEEE Trans. Commun.*, vol. 42, pp. 1881–1893, Feb./Apr. 1994.
[18] W. B. Pennebaker and J. L. Mitchell, *JPEG Still Image Data Compression Standard.* New York: Van Nostrand Reinhold, 1993.
[19] P. G. Howard, "Lossless and lossy compression of text images by soft pattern matching," in *Proc. Data Compression Conf.*, 1996, pp. 210–219.

Bo Martins was born in 1966 in Copenhagen, Denmark. He received the M.S. and Ph.D. degrees in electrical engineering from the Technical University of Denmark (DTU) in 1990 and 1996, respectively.

He is currently an Assistant Professor at the Department of Telecommunication at DTU. His interests include data compression, halftoning, and recognition. He is currently involved in the standardization of JBIG-2, an emerging international standard for compression of bilevel images.

Søren Forchhammer was born in 1959 in Copenhagen, Denmark. He received the M.S. degree in engineering and the Ph.D. degree from the Technical University of Denmark (DTU), Lyngby, in 1984 and 1988, respectively.

Currently, he is an Associate Professor at the Department of Telecommunication, DTU, where he has been employed since 1988. His main interests include data compression, image communications, and source coding.