Lossless/lossy compression of bilevel images

Martins, Bo, Forchhammer, Soren

Bo Martins, Soren Forchhammer, "Lossless/lossy compression of bilevel images," Proc. SPIE 3018, Color Imaging: Device-Independent Color, Color Hard Copy, and Graphic Arts II, (4 April 1997); doi: 10.1117/12.271615

SPIE.

Event: Electronic Imaging '97, 1997, San Jose, CA, United States

Lossless/Lossy Compression of Bi-level Images

Bo Martins and Søren Forchhammer
Dept. of Telecommunication, 371, Technical University of Denmark
e-mail: bm@tele.dtu.dk and sf@tele.dtu.dk

Abstract

We present a general method for lossless/lossy coding of bi-level images. The compression and decompression method is analoguous to JBIG, a current international standard for bi-level image compression, and is based on arithmetic coding and a template to determine the coding state. Loss is introduced in a preprocess on the encoding side by flipping pixels in a controlled manner. The method is primarily aimed at halftoned images as a supplement to the specialized soft pattern matching techniques which work better for text. The new algorithm also works well on documents of mixed contents e.g. halftoning and text without any segmentation of the image. The decoding is usually slower than JBIG due to a more wide-spread template. A decoding output of more than 1 Mpixels per second can be obtained in software implementations. We present a greedy 'rate-distortion' algorithm for flipping as well as less complex algorithms intended for relatively fast encoding and moderate latency. In the less complex algorithms, flipping and encoding is carried out in the same pass. The potential risk of flipping avalanches is minimized by conditioning flipping on the sign and magnitude of the local grey-scale error computed by a forgetful error diffusion algorithm. Template based refinement coding is applied for a lossy-to-lossless refinement step. The (de)coding method is proposed as part of JBIG-2, an emerging international standard for lossless/lossy compression of bi-level images.

1 Introduction

We present a general and robust method for lossless/lossy compression of bi-level images. The encoder uses arithmetic coding and a template which may be the one specified in the current international standard, JBIG. If so the decoder is JBIG. A preprocess on the encoder side introduces loss by flipping pixels in a controlled manner. The method can target the lossy image for a given not-too-large distortion. The decoder is the same in the lossy and the lossless case. An important issue is controlling the pixel errors. The (de)coding method is proposed as part of JBIG-2, an emerging international standard for lossless/lossy compression of bi-level images.
Soft Pattern Matching (SPM) [1] is effective for lossy/lossless coding of bi-level images as text images. SPM is not used for e.g. bi-level halftone images. Therefore, we focus on halftones, especially periodic halftones and images of mixed contents. At 2-3 times the compression of JBIG-1 periodic halftones are perceptually lossless.

2 Template Coding

We consider k-order template coding of a bi-level image, x^T, where the pixels are coded sequentially in raster scan order applying arithmetic coding to a sequence of conditional probabilities. The unknown pixel at time t, is denoted U, its value is denoted u, and its context c. If prior to time t, n_0 zeroes and n_1 ones were generated in context, $c = c^k$, we use the (sequential) estimator:

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

This expression is the basis of the JBIG [2] estimator (the estimator is implemented as a locally adaptive probability state machine using $\delta = 0.45$).
Following [3] we can calculate the accumulated code length of the events directly from the frequency counts in the contexts regardless of the order in which the values appeared. 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). In [3] the code length is approximated very

accurately by 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:

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

where $\xi_1 = -\log\frac{\sqrt{2\pi}\Gamma(2\delta)}{\Gamma^2(\delta)}$ and $\xi_2 = -\log\frac{\Gamma(2\delta)}{\Gamma(\delta)}$ are constants. $T_1(n)$ is a table for $\ell(n, 0|\delta)$. The logarithms are base 2. $\Gamma(\cdot)$ denotes the gamma function.

3 Rate

From the previous section it is clear that we may compute the code length of the image from the statistics at time T when we use the probability estimator (1). The (lossless) code length, $L(x^T)$, is:

$$L(x^T) = \sum_{c^k} \ell(n_{0|c^k}, n_{1|c^k}) \tag{3}$$

The marginal effect of flipping some pixel U with original value u in context, c^k, may also be computed from the statistics at time T. This is composed of the direct effect of coding another value and the effect of changing the context for all the pixels for which the flipped pixel appears in the template. Let ΔL_0 denote the 'direct' effect of flipping U, i.e. generating one less u and one more $\bar{u}$ in context c^k (we use the notation $\bar{u} = 1 - u$). U is itself a context pixel for the k pixels which result from mirroring the k template pixels in U. In Fig. 1, we have depicted the template with which we code JBIG test image s04a400 along with its mirror template. Let ΔL_q denote the impact on the code length of the events generated in contexts, v_q and w_q, when flipping U. v_q is the original context for the mirror pixel of context pixel no. q. w_q is the context for the mirror pixel of context pixel no. q if U is flipped. We have $v_1 = uc_2^k$, $v_k = c_1^{k-1}u$, and $v_q = c_1^{q-1}uc_{q+1}^k$ for $1 < q < k$. And we have $w_1 = \bar{u}c_2^k$, $w_k = c_1^{k-1}\bar{u}$, and $w_q = c_1^{q-1}\bar{u}c_{q+1}^k$ for $1 < q < k$. The marginal effect on the total code length, $L(x^T)$, by flipping U is:

$$\Delta L(x^T) = \sum_{q=0}^{k} \Delta L_q \tag{4}$$

The 'direct' gain from flipping is:

$$\Delta L_0(x^T) = \log\frac{n_{u|c^k} - 1 + \delta}{n_{0|c^k} + n_{1|c^k} - 1 + 2\delta} - \log\frac{n_{\bar{u}|c^k} + \delta}{n_{0|c^k} + n_{1|c^k} - 1 + 2\delta} \tag{5}$$

For $q > 0$ we have:

$$\Delta L_q(x^T) = \log\frac{n_{u'|v_q} - 1 + \delta}{n_{0|v_q} + n_{1|v_q} - 1 + 2\delta} - \log\frac{n_{u'|w_q} + \delta}{n_{0|w_q} + n_{1|w_q} + 2\delta} \tag{6}$$

In (4) $\Delta L(x^T)$ expresses the number of bits it costs to flip U. Hence, a negative value of $\Delta L(x^T)$ makes flipping profitable. If we choose to flip U then we update the involved statistics, $c^k, v_1, \cdots, v_k, w_1, \cdots, w_k$ accordingly. The (lossless) code length of the altered image is $L(x^T) + \Delta L(x^T)$. (We may note that when evaluating (6) we should use an intermediate update of counters for the rare event that flipping a pixel will affect the same context more than once.)

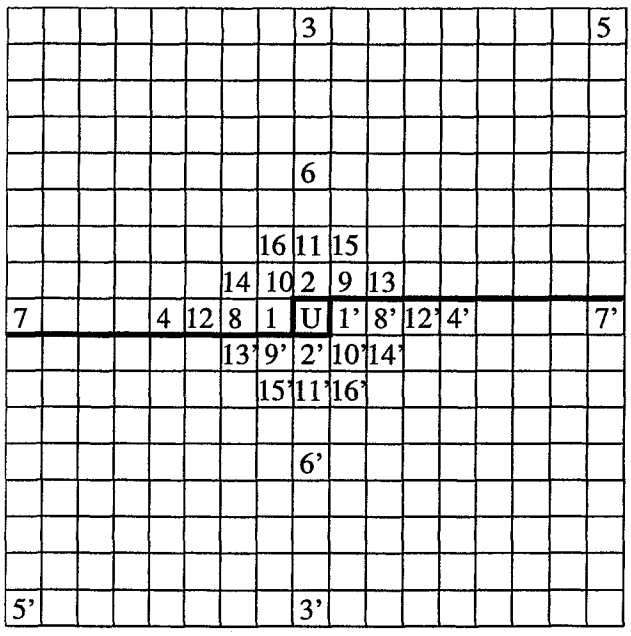

Figure 1: Preparing lossy template coding with template pixels 1–16. The current pixel, U, becomes template pixel no. q when the template origin is at position q'.

4 Distortion Measures and Control

Bit-flipping affects the visual quality of the lossy image in different ways depending on the type of image. For scanned text legibility is the most important aspect of quality and smoothness of the contours of the letters the second most important aspect. Bit-flipping that increases smoothness without affecting legibility is not undesirable. For text images soft pattern matching (SPM) [1] techniques produces excellent compression at perceptually lossless quality. Our primary goal is therefore to handle halftones and mixed documents. For halftones it is important to preserve the perceptual grey-scale value over both small and large areas and to preserve sharp edges. For these images bit-flipping is a gentle way of decreasing quality.

For simplicity we use the Hamming distance, i.e. the number of bit errors, as distortion. In the rate-distortion inspired flipping strategy below other distortion measures, d, could be used. We also introduce distortion control by disallowing flipping according to some rules. In terms of distortion measures this means that we assign infinite distortion in these instances.

We apply 2 ways of controlling distortion which are both related to image blocks of m_b rows and n_b columns. The first method aims at preserving image details by setting a maximum on the total number of pixels that may be flipped in a block. The second method aims at reducing the error of the (percepted) grey-scale over a larger area of an image. In its pure form it is an error diffusion of the flipping-errors. The way it works is that flippings that tend to reduce the grey-scale error are preferred to flippings that tend to increase it. The error diffusion in its pure form is rather too restrictive and we modify it by a combination of occasionally scaling down the diffused error and of allowing flippings that increase the grey-scale error as long as the diffused (and possibly scaled) grey-scale error is not too large.

5 Introducing Loss

We only consider flipping the pixels one at a time. First we present a more complex multi-pass rate-distortion inspired algorithm. Thereafter we focus on fast, low-latency lossy compression.

5.1 Rate-Distortion Flipping

This method provides a continuous 'rate-distortion' trade-off up to a maximum distortion (the r-d curve is not optimal). Lossy compression consists of 5 steps:
1) Finding the adaptive template pixels. 2) Collecting image statistics for all contexts. 3) Listing flip candidates. 4) Evaluating flip candidates and possibly flipping them. 5) Losslessly encoding the altered bitmap.
Step 1 will be described in Sect. 7. Step 2 was described in Sect. 2. The probability estimator (1) is used. The code length is given by the values of the counts independently of the order occurences appear. Steps 3 and 4 may have to be repeated if a low rate is desired. It should be noted that the complexity of the decoder is not affected by the complexity of the encoder.
Steps 3 and 4 specify the flipping of pixels, thus defining the lossy image. The process may be stopped at a specified target rate $L*$ or distortion $d*$ or at a specified slope of rate-distortion. Based on the counts we may calculate a marginal code length $l = -\log \hat{p}(u|c)$ for an individual pixel U in context, c, based on the counts for the whole image. We may also calculate the marginal change in rate for the image, $\Delta L(x^T)$, from (4) (5) (6). As in [4], we aim at optimizing the rate-distortion. This implies operating with a given slope point on the r-d curve. Therefore, we threshold $\Delta L(x^T)/\Delta d$ as the essential part of the decision whether we should flip a fixel. Here, Δd is the incremental change in distortion that will result from flipping the pixel. We slide down the 'rate-distortion' curve decreasing the numerical value of the slope threshold, τ. We do not obtain rate-distortion optimality as we consider one pixel at a time and are constrained to use the decoding algorithm chosen once and for all.
A more detailed version of Steps 3 and 4 is

- Calculate the lossless coding rate $L(x^T)$ and set distortion $d = 0$
  If $L(x^T) \leq L^*$ STOP.
- 3) List flip candidates:
  For each pixel U of the image
    If marginal code length $l \geq \tau_l$ and marginal rate $\Delta L(x^T) < \tau_L$ then
      Add $(U, \Delta L(x^T))$ to the flip candidate list
- 4) Evaluate flip candidates:
  For threshold $\tau = \tau_{\min}, \tau = \tau + \Delta\tau, \tau \leq \tau_L$
    For pixels U of the candidate list with (formerly computed) $\Delta L(x^T) < \tau$
      Recompute $\Delta L(x^T)$
      If $\Delta L(x^T)/\Delta d < \min(\tau_f, \tau)$ then
        Flip pixel U and update counter statistics
        $L(x^T) = L(x^T) + \Delta L(x^T)$
        $d = d + \Delta d$
        If $L(x^T) \leq L*$ or $d \geq d*$ STOP.
- Repeat Steps 3 and 4 a maximum of N_i times

The algorithm above stops at the specified rate L^* or distortion d^* if achievable. If we desire a specific slope of the 'rate-distortion' curve it is specified by τ_f and the test for L^* and d^* becomes void.
In our experiments, the distortion measure Δd is 1 bit per error pixel.
Although each pixel may be subject to the calculation of (4) we may gain a large factor in speed by observing that the differential code lengths, ΔL_q, must be expected to have unequal influence on ΔL. It is unlikely that it will be profitable to flip a pixel for which $\Delta L_0 > 0$, i.e. a pixel that is a more probable symbol (MPS). For highly compressible images, such as text or periodic halftones less probable symbol (LPS) pixels are rare. For example the JBIG test image s04a400 has less than 2% LPS'es. It is a fast procedure to determine whether a pixel is an MPS or an LPS. Therefore, we choose the threshold $\tau_l = 1$ bit, which means that the candidate list consists of (a subset of) LPS pixels. Especially, for highly compressable images this reduces the candidate list to be considered.
In Step 3 we may safely use a maximal list size (much) smaller than the number of pixels in the image as we intend to flip only a small fraction of the pixels of the image. If we do that it may happen that the list is not long enough. In that case we decrement τ_L and repeat Step 3. We use a list size of 1/32 times the total number of pixels of the image. For the secondary parameters, we use the (initial) value: ($\tau_{\min}$,$\Delta\tau$,τ_L,τ_f) = (-15 nats,1 nat,+5 bits,-1 kbyte per % distortion).
Storing the values of $L(x^T)$ and d as the algorithm proceeds gives the 'rate-distortion' curve before actually coding the image in Step 5. Using $N_i = 5$ for two test images we get the curves depicted in Fig. 4.

5.2 Flipping Utilizing Forgetful Error diffusion

For faster compression and to decrease latency we may prefer simpler coding schemes. The next two algorithms require two passes of the image. In the first pass the statistics are collected, so that we can perform the flipping calculations. In the second pass we flip.
In this method when flipping, we process the m_b by n_b blocks in raster scan order. Within a block we compute for every flip candidate (if any) the corresponding $\Delta L(x^T)$. At the end of the block we flip the flip candidate with the smallest value of $\Delta L(x^T)$ if the distortion rules allow it. If we flip one pixel in a block we have a strong wish to make the block grey-scale neutral by flipping a pixel of the opposite color. One possibility is, in a second pass of the block, to consider all pixels of the opposite color of the previously flipped. If any of these has a negative value of $\Delta L(x^T)$ a second pixel is flipped.
Lossy compression consists of 3 steps:
1) Finding the adaptive template pixels. 2) Collecting image statistics for all contexts. 3) Proceeding blockwise, flip pixels. Diffuse the introduced grey-scale error to neighbouring blocks. Proceeding in raster scan order, encode the image (with latency m_b lines).

- 3.1) Flip pixels in block (i, j):
  for $a = 0$; $a < a_{\max}$; $a = a + 1$

  List flip candidates of block (i, j):
  For each pixel U of the block

  If marginal code length $l \geq \tau_l(a)$ and marginal rate $\Delta L(x^T) < \tau_L(a)$ then
  Add $(U, \Delta L(x^T))$ to the flip candidate list.

  Evaluate flipping:
  If the list is not empty

  Pick the flip pixel U^* of the list as the one with the smallest value of $\Delta L(x^T)$. If the current grey-scale error of block (i, j), $g_{i,j}$, is larger than τ_s only candidates for which $u = 1$ are considered. If the $g_{i,j} < -\tau_s$ only candidates for which $u = 0$ are considered.
  If there is a flip pixel, U^*

  Let the original value of U^* be denoted u^*.
  Flip U^* and update counter statistics
  $L(x^T) = L(x^T) + \Delta L(x^T)$
  $d = d + \Delta d$
  Update the block grey-scale error:
  if $u^* = 1$ then $g_{i,j} = g_{i,j} - 1$
  else $g_{i,j} = g_{i,j} + 1$

  If no pixel was flipped exit Step 3.1

- 3.2) Diffuse the grey-scale error at block level:

  If there was not flipped in the block

  $g_{i,j} = g_{i,j} \cdot \tau_s$

  Diffuse $g_{i,j}$ to the non-causal neighbouring blocks using e.g. Floyd-Steinberg weights.

The choice of the quality controlling parameters τ_g and τ_s is critical for the lossy quality. We use the values $\tau_g = 0.25$ and $\tau_s = 0.5$. If we use Floyd-Steinberg weights and if the template is no higher than m_b and does not extend farther away to either side from the current pixel than n_b this choice of parameters prevents the worst form of flipping avalanches. We compromise slightly on speed using $a_{\max} = 2$, hoping to pair flips. We usually use $(m_b, n_b, \tau_l(0), \tau_L(0), \tau_l(1), \tau_L(1)) = (8, 8, 1, 0, 1, 0)$. Thus, in both passes we only consider flipping pixels that are LPS'es.
To increase compression the individual blocks and/or the entire image may be subjected to bit-flipping multiple times and the parameters τ_g and/or τ_s may be increased.

5.3 *Safe* flipping

The simplest form of flipping is that of scanning the image in raster scan order, processing each pixel instantly in the second pass, flipping it if $\Delta L(x^T)$ in (4) is negative (or, in general, less than some negative threshold). This method has the unfortunate side effect that flipping avalanches may occur. Such avalanches appear as artefacts and should therefore not be tolerated.
The simplest way to avoid avalanches is a rule that forbids the flipping of a pixel that has a flipped pixel in its context. With this control mechanism latency is reduced further compared to 'error-diffusion'-flipping. As we process each pixel only once, speed is usually improved, also. The only disadvantage about *safe*-flipping is that it may be too restrictive - that we will not flip enough pixels to achieve a much lower rate than the lossless rate.

6 Refinement Coding

Refinement coding gives a lossless coding using the lossy image. We use a refinement coder that generates the lossless image with a template coder that has template pixels in the causal part of the lossless image and template pixels in the non-causal part of the lossy image (see Fig. 3). The reason we do not wish to use template pixels from the causal part of the lossy image is that refinement becomes particularly neat: we can refine the image by writing in the lossy image. Using this technique and the template of Fig. 3 refinement coding can be made extremely fast.
As an option, refinement coding can consist in coding an error location bitmap which when XORed with the lossy bitmap produces a refined bitmap. This way lossy/lossless coding including refinement may be performed based on JBIG-1.

7 Template Selection

The template selection has great influence on compression [3] [5] and may also have a big impact on decoding speed. We propose that the lossy-lossless coder may use a 16 pixel template of which 4 pixels are adaptive. The default location of the adaptive template pixels and the location of the 12 fixed pixels is proposed as shown in Fig. 2.

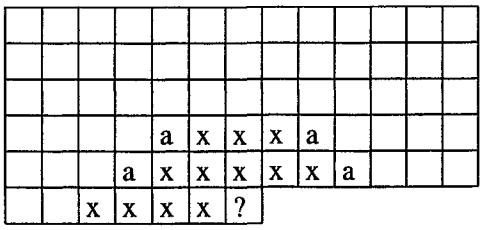

Figure 2: Proposed 3-line template for lossless and lossy coding. The pixels marked a are default positions for the 4 adaptive template pixels.

The JBIG-1 10 pixel template (9 default pixels and 1 free pixel) is used as an option and for comparison.
The refinement coder may use a 12 pixel template out of which 2 pixels are adaptive. The default location of the adaptive template pixels and the location of the 10 fixed pixels is proposed as shown in Fig. 3.
We pick the lossy/lossles template greedily as in [3] starting with the two causal four-neighbours and picking a number of adaptive template pixels (usually 2, 3, or 4). Eventually we supplement up to k pixels picking the remaining pixels by default.
The adaptive pixels are picked by a fast method utilizing randomized subsampling [3]. The decoding speed is largely determined by the appearence of the adaptive and the fixed template. When the template pixels on every line appear in one run the number of pixel lines involved determine the decoding speed. If we wish to select a size 16 template with (max.) 4 adaptive pixels and a 3-line default template, we start out including the two four-neighbours in the template, grow it greedily till it has size 6. It may occur that 1 or more of the chosen pixels were in the default template so that we are free to pick one or more free pixels if we so choose. Our rule is to pick another free pixel if exactly 1 of the free pixels were in the default template, thus requiring a strong periodicity in order to choose another free pixel. Once we have found the free pixels we supplement the template by the default template pixels untill we have 16 different template pixels. The search for free pixels may terminate before the template has size 6. In that

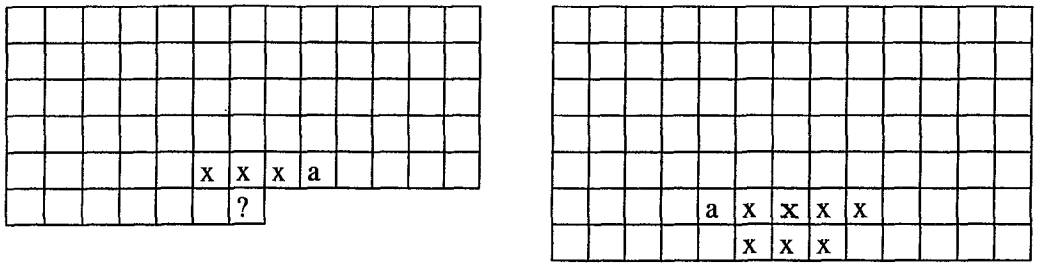

Figure 3: 3-line template for refinement coding. The pixels marked a are default positions for the 2 adaptive template pixels. The pixel marked in boldface in the lossy image is aligned with the current pixel, ?.

case we use fewer free pixels and more default pixels. The resulting template for test image s04a400 is depicted as pixels 1–16 on Fig. 1.

The refinement template is picked similarly to the lossy/lossless template. When picking the free template pixels the initial template consists of the pixel in the lossy image that is aligned with the current pixel.

As described in [5] and [3] the selection of the free template pixels becomes slower as the template grows. To increase speed we therefore reduce the set of candidates for the free pixels. We cannot do this by default without risking hurting compression, but we do have a very good indication of which pixels may be good template pixel if we do the greedy search once. When we find the first free pixel we proceed as described above, but at the same time we order the remaining candidates with respect to how well they performed. In the following searches we only collect statistics for the first n_a candidates on this list. In our experiments we reduce the initial candidate set of 544 pixels to 64. This also has the effect that the memory requirement for finding the template pixels is lowered accordingly. Even if we search for 5 free pixels the memory requirement does not exceed the 65 kbytes which we need anyhow for QM coding. It should be noted that we should still include near-by pixels in the template prior to searching for the first pixel so that we do not risk not finding the period in documents of mixed text and halftone.

8 Results

We have compressed images in the JBIG-2 test set with the new algorithm. The entropy coder is a QM-coder for all results.

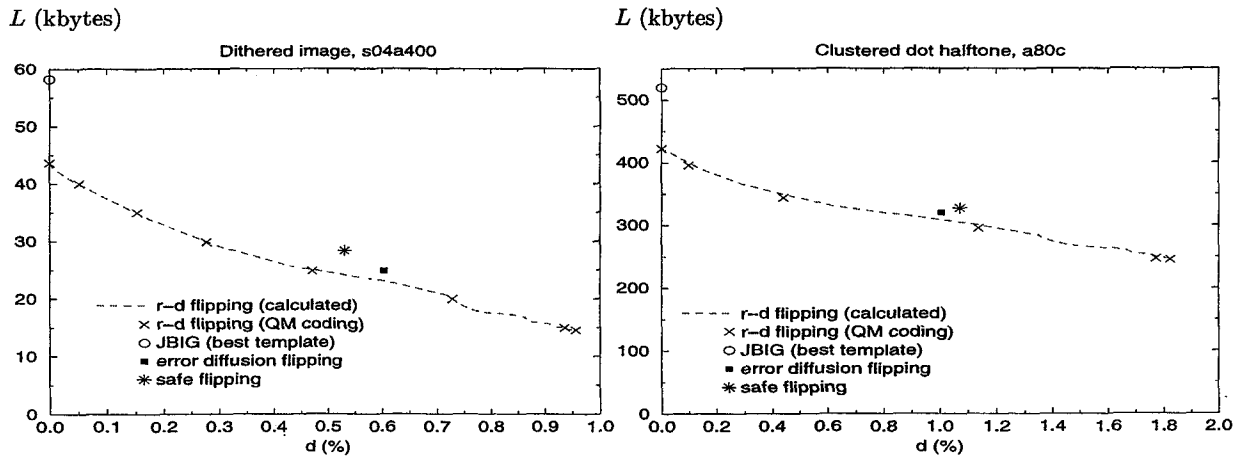

Figure 4: Compression and distortion results for a dither and a clustered dot halftone image. The 'rate-distortion' curve comes for free during 'r-d'-flipping.

In Tables 1, 2, and 3 we list results for the (for our purpose) most interesting ones. The images listed are the halftones

	Lossless			Lossy						Refinement		
Image	L(bytes)	$t_c(s)$	$t_d(s)$	L(bytes)	$d(\%)$	$t_c(s)$	L(bytes)	$d(\%)$	$t_c(s)$	L(bytes)	$t_c(s)$	$t_d(s)$
f10_200	53366	3.6	1.7	36615	0.9	12.8	32748	1.1	23.7	27702	6.1	2.2
f10_240	76031	5.3	2.5	53832	0.8	18.8	48181	1.1	35.1	41086	8.3	3.0
f10_300	99537	7.7	3.8	70717	0.7	26.0	63928	1.0	50.2	56858	11.7	5.2
f15_200	92964	3.0	1.5	72072	1.1	22.7	58397	2.3	43.0	46343	4.2	1.7
f16_800	158549	11.0	6.4	120520	0.6	38.6	115334	0.7	77.5	62875	15.1	8.3
s06a400	165378	12.0	5.8	111708	1.0	41.9	99376	1.3	80.7	89153	17.1	7.6
Mix,Halft.	645825	42.6	21.7	465464	-	160.8	417964	-	310.2	324017	62.5	28.0
s04a400	43609	6.7	2.6	20471	0.7	16.9	14571	1.0	29.4	37825	9.4	3.4
s04b400	39508	5.6	2.6	18997	0.6	14.6	15118	0.8	25.9	32818	9.4	3.4
s04c400	35995	6.7	2.5	17483	0.5	15.2	12638	0.7	27.1	30731	9.4	3.6
s04d400	35008	6.3	2.5	17820	0.5	14.9	13350	0.7	26.0	28926	9.4	3.4
a80c	422228	21.6	13.0	279902	1.4	114.3	245834	1.8	229.9	276289	25.9	16.3
Per. Halft.	576348	46.9	23.2	354673	-	175.9	301511	-	338.3	406589	63.5	30.1
s09a400	73652	1.3	0.7	67008	0.9	13.2	57771	3.7	40.0	27211	1.5	0.7
All Halft.	**1295825**	**90.8**	**45.6**	**887145**	-	**349.9**	**777246**	-	**688.5**	**757817**	**127.5**	**58.8**
f06_200	11637	3.9	1.6	8547	0.1	6.7	8091	0.2	11.6	4368	5.8	2.2
f06_300	17489	7.1	3.6	13026	0.1	12.0	12525	0.1	21.9	6446	11.4	4.9
f06_400	24061	9.9	6.3	17943	0.1	18.1	17214	0.1	34.5	8625	16.3	8.7
Line/Text	53187	20.9	11.5	39516	-	36.8	37830	-	68.0	19439	33.5	15.8
f02_200	7663	4.0	1.6	6063	0.1	6.0	5790	0.1	10.7	2678	5.7	2.1
f02_300	11707	7.1	3.5	9153	0.1	11.2	8752	0.1	20.0	3725	11.0	5.0
f02_400	15414	10.7	6.3	12173	0.0	17.2	11633	0.1	33.4	5224	16.1	8.8
f08_200	13004	4.1	1.6	10508	0.1	7.2	10105	0.2	12.8	4186	6.3	2.2
f08_300	18589	8.0	3.7	14770	0.1	13.4	14261	0.1	23.9	6069	12.3	5.0
f08_400	25217	11.2	6.5	19987	0.1	19.5	19402	0.1	37.9	8023	16.0	8.7
Hand wr.	91594	45.1	23.2	72654	-	74.5	69943	-	138.7	29905	67.4	31.8

Table 1: Compression and timing results for the rd-flipping algorithm. Lossy results are given after $N_i = 1$ pass and after $N_i = 5$ passes. Refinement coding of the lossy image after 5 passes.

and mixture images (text+halftone and/or mixed halftones). Also a number of images that do not primarily consist of typed text are included. The subjective quality of the lossy reconstructions is good or perceptually lossless for most images. For the error diffusion s09a400 the visual quality deteriorates rapidly when flipping. We do not consider bit-flipping a good method for lossy compression of error diffused images as the decrease in quality is too high compared to the relatively small gain in compression rate.

The algorithms work with any template. They can be taylored to JBIG (i.e. JBIG-1) in which case existing encoding and decoding software/hardware can be used. With this choice, the algorithm requires only a preprocessing unit on the encoder side. For 'All Halftones', lossy JBIG-1 yields 22% higher compression than lossless JBIG-1 using 'error-diffusion' flipping (Table 3). In this experiment we used $\tau_l(1) = \infty$ in order to maximize compression. The single adaptive pixel was found as described in Sect. 7. We did not look for a second pixel if the first pixel was in the default JBIG-1 template.

By allowing a larger template and a few more adaptive template pixels compression performance is increased at higher quality. For periodic halftones the lossy compression rate is more than twice what can be obtained with lossless JBIG. Using 'error-diffusion' flipping ($\tau_l(1) = 0$) we can obtain 33% better lossy compression of halftoned images at better quality than lossy JBIG. Better quality for the same rate or lower rate for the same quality is achieved with 'rate-distortion' flipping.

We achieve considerably better lossless compression on the halftones using the larger template and more adaptive pixels. With max. 4 adaptive pixels we achieve 26% better lossless compression than lossless JBIG on the halftones. (With 2 adaptive pixels we achieve 20% better lossless compression than lossless JBIG.)

Using the rate-distortion based flipping followed by refinement 1536 kbytes is used to code 'All Halftones'. This is 18% more than what is used by lossless coding in one step (Table 1).

	Lossy (err.diff)				Refinement (err. diff)			Lossy (safe)		
Image	L(bytes)	d(%)	$t_c(s)$	$t_d(s)$	Lbytes	$t_c(s)$	$t_d(s)$	Lbytes	d(%)	$t_c(s)$
f10_200	43882	0.5	8.9	1.8	16554	5.8	2.2	43909	0.6	7.2
f10_240	62608	0.5	12.7	2.4	23108	8.8	3.1	62686	0.6	10.5
f10_300	82220	0.5	18.5	3.7	32637	11.6	5.0	82411	0.6	15.6
f15_200	79801	0.9	10.3	1.5	23874	4.2	1.6	78352	1.3	6.8
f16_800	132412	0.4	28.5	6.4	45343	15.0	8.2	134148	0.5	23.9
s06a400	129091	0.7	29.4	5.7	58324	16.7	7.6	130455	0.8	24.2
Mix,Halft.	530014	-	108.3	21.5	199840	62.1	27.7	531961	-	88.2
s04a400	24790	0.6	13.0	2.5	28568	9.4	3.6	28345	0.5	11.9
s04b400	23837	0.5	11.4	2.5	24012	9.3	3.5	25979	0.5	10.5
s04c400	22685	0.4	12.3	2.5	20667	9.7	3.6	22944	0.4	11.6
s04d400	24112	0.4	11.8	2.5	19334	9.5	3.6	23655	0.4	11.2
a80c	319901	1.0	67.9	12.7	187274	26.2	16.2	327099	1.1	60.7
Per. Halft.	415325	-	116.4	22.7	279855	64.1	30.5	428022	-	105.9
s09a400	66583	1.8	6.6	0.7	16787	1.6	0.6	67345	2.2	3.8
All Halft.	**1011922**	-	**231.3**	**44.9**	**496482**	**127.8**	**58.8**	**1027328**	-	**197.9**
f06_200	9222	0.1	6.5	1.7	3432	6.1	2.2	9280	0.1	6.7
f06_300	13845	0.1	13.1	3.7	5155	11.8	5.0	13934	0.1	13.3
f06_400	18884	0.1	19.4	6.3	7181	17.5	8.9	18971	0.1	20.6
Line/Text	41951	-	39.0	11.7	15768	35.4	16.1	42185	-	40.6
f02_200	6335	0.1	6.3	1.6	2197	6.3	2.3	6389	0.1	6.7
f02_300	9789	0.0	12.6	3.6	3251	11.1	4.9	9784	0.1	13.2
f02_400	12580	0.0	20.2	6.4	4509	16.2	8.7	12720	0.1	21.4
f08_200	10983	0.1	6.9	1.6	3448	6.5	2.2	11057	0.2	6.9
f08_300	16060	0.1	13.5	3.7	5465	12.1	4.9	16014	0.1	14.2
f08_400	20887	0.1	21.1	6.4	7568	17.2	8.8	20931	0.1	22.0
Hand wr.	76634	-	80.6	23.3	26438	69.4	31.8	76895	-	84.4

Table 2: Compression and timing results for the error diffusion flipping algorithm and for the safe-flipping algorithm.

8.1 Complexity

The timing results for our implementations in C of the new new algorithms are stated in Tables 1 and 2. Encoding time is denoted t_c, decoding time is denoted t_d. The platform is a HP workstation series 9000, model C160, using the HP-UX 10.20 operating system.

Decoding speed is roughly 2Mpixels per second regardless of whether the image is lossy or lossless. Due to the implementation refinement decoding appears slightly slower, but is potentially faster.

Lossless encoding is always slower than decoding because the template must be found on the encoding side. Due to the new template selection speed-up an encoding speed of appr. 1 Mpixels per second is achieved. Lossy encoding is slower than lossless encoding. For 'All halftones' the 'r-d'-flipping algorithm with 1 flipping pass is twice as slow as error diffusion flipping and 4 times as slow as lossless encoding. 'r-d'-flipping with 5 flipping passes is twice as slow as 'r-d'-flipping with 1 flipping pass. 'r-d'-flipping is slower than the 'error diffusion'-flipping because the former has more computations of $\Delta L(x^T)$. In the current form the 'r-d'-algorithm is much slower in the instances where the flip candidate list runs full and must be remade. This problem, however, can be dealt with using some standard buffer control technique so that τ_L is a function of the remaining free space on the list. It should be noted that the list only ran full for those test images that were not well-suited for the flipping algorithm: s09a400 (an error diffused image) and f15_200 (containing a sub-image that is apparently dithered with white noise). *safe*-flipping is only a little faster than 'error-diffusion'-flipping in our implementations, so we prefer 'error-diffusion'-flipping for the low-latency algorithm.

9 Conclusion

We have presented a general algorithm for lossy/lossless compression. Loss is introduced by flipping pixels before applying lossless template coding. A number of strategies for controlling the flipping of pixels was introduced. The best compression results (for a given quality) was obtained by a rate-distortion inspired algorithm. The algorithm

	Lossless	Lossy	
Image	L(bytes)	L(bytes)	d(%)
f10_200	63649	55771	0.5
f10_240	103029	90759	0.6
f10_300	145772	131078	0.5
f15_200	113072	102016	1.2
f16_800	160908	137683	0.4
s06a400	218077	177265	0.9
Mix,Halft.	804507	694572	-
s04a400	58153	34138	0.6
s04b400	56651	33411	0.5
s04c400	52545	33476	0.4
s04d400	48069	32459	0.4
a80c	535422	438579	1.1
Per. Halft.	750840	572063	-
s09a400	74429	67908	2.4
All Halft.	**1629776**	**1334543**	-
f06_200	11992	9501	0.1
f06_300	18286	14313	0.1
f06_400	25015	19516	0.1
Line/Text	55293	43330	-
f02_200	8040	6534	0.1
f02_300	12051	9679	0.1
f02_400	16764	13462	0.0
f08_200	13309	11087	0.2
f08_300	19865	16123	0.1
f08_400	27380	22230	0.1
Hand wr.	97409	79115	-

Table 3: Compression results for JBIG. Lossy JBIG using the error diffusion flipping algorithm.

outputs an image with a specific rate or a specific distortion up to some maximum distortion. Faster algorithms with less latency needing extra distortion control were presented. Also for these algorithms a 'rate-distortion' trade-off defines the lossy image. The new algorithms yield significant improvement in compression for low distortion. Halftones (excepting error diffusion type images) and mixed documents are dealt with efficiently. The algorithms offer an effective supplement to the text image oriented SPM-technique.

References

[1] Paul Glor Howard, "Lossless and Lossy Compression of Text Images by Soft Pattern Matching," in *Proc. of Data Compression Conference*, 1996, pp. 210–219.

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

[3] B. Martins and S. Forchhammer, "Tree Coding of Bi-level Images," Submitted to *IEEE Trans. Image Proc.*: June 96, 1996.

[4] K. Ramachandran and Martin Vetterli, "Rate-distortion optimal fast thresholding with complete JPEG/MPEG decoder compatibility," *IEEE Trans. Image Proc.*, vol. 3, no. 5, pp. 700–704, Sept. 1994.

[5] B. Martins and S. Forchhammer, "Bi-level Compression with Tree Coding," in *Proc. of Data Compression Conference*, 1996, pp. 270–279.

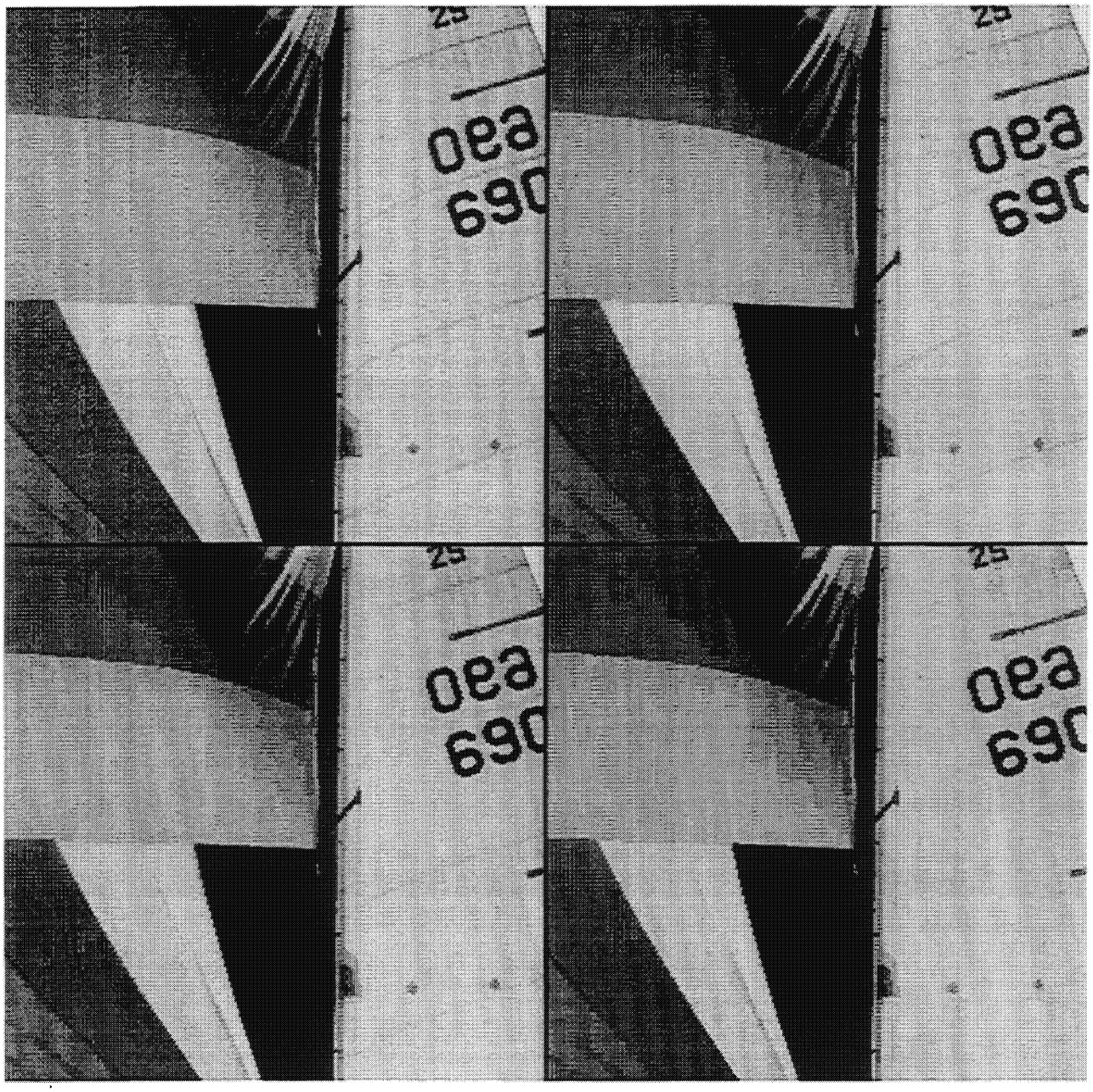

Figure 5: Detail of dithered image s04a400. Upper left: original, $L = 43609$ bytes. Upper right: 'error diffusion' flipping, $(L, d) = (24790 \text{ bytes}, 0.6\%)$. Lower left: 'r-d' flipping (after 1 pass), $(L, d) = (20471 \text{ bytes}, 0.7\%)$. Lower right: 'r-d' flipping (after 5 passes), $(L, d) = (14571 \text{ bytes}, 1.0\%)$

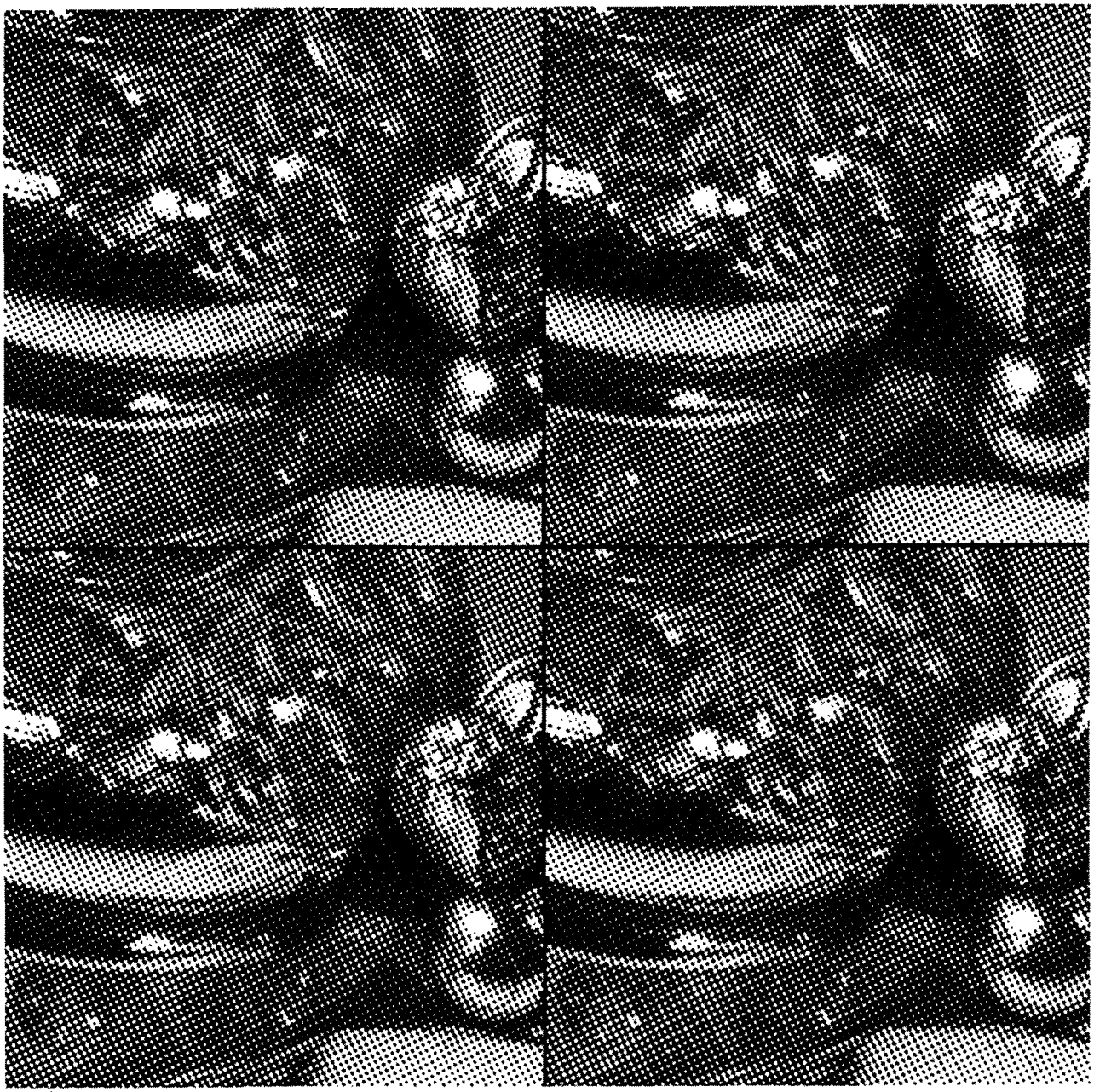

Figure 6: Detail of clustered dot halftoned image a80c. Upper left: original.$L = 422228$ bytes. Upper right: 'error diffusion' flipping, $(L, d) = (319901 \text{ bytes}, 1.0\%)$. Lower left: 'r-d' flipping (after 1 pass), $(L, d) = (279902 \text{ bytes}, 1.4\%)$. Lower right: 'r-d' flipping (after 5 passes), $(L, d) = (245834 \text{ bytes}, 1.8\%)$