Quadtree Based JBIG Compression

B. Fowler   R. Arps[*]   A. El Gamal   D. Yang

ISL, Stanford University, Stanford, CA 94305-4055
{fowler,arps,abbas,dyang}@isl.stanford.edu

Abstract

A JBIG compliant, quadtree based, lossless image compression algorithm is described. In terms of the number of arithmetic coding operations required to code an image, this algorithm is significantly faster than previous JBIG algorithm variations. Based on this criterion, our algorithm achieves an average speed increase of more than 9 times with only a 5% decrease in compression when tested on the eight CCITT bi-level test images and compared against the basic non-progressive JBIG algorithm. The fastest JBIG variation that we know of, using "PRES" resolution reduction and progressive buildup, achieved an average speed increase of less than 6 times with a 7% decrease in compression, under the same conditions.

1   Introduction

In facsimile applications it is desirable to integrate a bilevel image sensor with lossless compression on the same chip. Such integration would lower power consumption, improve reliability, and reduce system cost. To reap these benefits, however, the selection of the compression algorithm must take into consideration the implementation tradeoffs introduced by integration. On the one hand, integration enhances the possibility of parallelism which, if properly exploited, can speed up compression. On the other hand, the compression circuitry cannot be too complex because of limitations on the available chip area. Moreover, most of the chip area on a bilevel image sensor must be occupied by photodetectors, leaving only the edges for digital logic.

The first compression algorithm we investigated for integration with an *area* image sensor is based on the well known quadtree data structure [1]. The quadtree algorithm we employeed involves performing logical OR operations on blocks of pixel data to create pixels in successive layers of a resolution pyramid. This is a form of fixed to variable length coding that exploits background skipping. Such simple logical operations can be easily performed during the "read out" from a sensor without any impact on access time. As a result, this algorithm is well suited for integration with a sensor. It exploits the parallelism available on a chip without requiring complex

---

[*]IBM Almaden Research Center, San Jose, CA 95120-6099 (arps@almaden.ibm.com).

circuitry. Unfortunately, the quadtree algorithm does not provide adequate compression ratios, providing typically less than a factor of 2, compared to ratios of 20 to 100 for state of the art algorithms such as the JBIG standard [2]. To achieve maximal compression we considered the *basic, non-progressive* JBIG. This algorithm does not, however, take full advantage of the parallelism afforded by integration since only local image information is used during the coding process.

To get the best of both worlds we decided to use adaptive arithmetic coding with conditional prediction, as used in JBIG, following quadtree compression. We then realized that such a combination can be made compliant with the JBIG standard! The quadtree data structure can be viewed as a *resolution reduction* pyramid in a *progressive* JBIG version. The quadtree algorithm behaves like a combination of a resolution reduction method and the *deterministic prediction* [3] option in a progressive JBIG algorithm.

Deterministic prediction skips over pixels in coding that can be completely predicted at a decoder, given only the already reconstructed pixels of the progressive resolution pyramid. Such a quadtree (QT) algorithm can be completely realized within the constraints of the JBIG user-specifiable resolution reduction method, along with its corresponding constrained table for deterministic prediction. Note that the deterministic prediction implied by the quadtree algorithm does not completely exploit all of the deterministic prediction potential presented by its resolution reduction scheme. This will be explained further in Section 3.

The paper is organized as follows. The next section provides a brief review of the basic JBIG algorithm, hierarchical resolution reduction of binary images, and typical and deterministic pixel prediction. Our JBIG compliant quadtree (QT) algorithm is described in Section 3. In Section 4 we use a set of eight CCITT images to compare our algorithm to other versions of JBIG algorithms on the basis of compression and speed. We show that our algorithm is uniformly faster in terms of arithmetic coding operations and achieves comparable compression.

2   Algorithmic Aspects of JBIG

JBIG compression algorithms can be categorized into two classes: non-progressive and progressive, where in the latter case a hierarchy of resolution information about the image is available for coding. Information on the basic non-progressive JBIG to which we also compare our QT JBIG algorithm can be found in [2].

A block diagram of a progressive JBIG binary image compression encoder is given in Figure 1. The algorithm begins by transforming the original image into a pyramid of hierarchical, resolution reduced images. Each resolution reduced image is then encoded, beginning with the lowest resolution image. Typical and deterministic prediction [3] of pixels, as will be explained, is used where possible, so that such pixels can be skipped in the arithmetic coding step – thus achieving better compression along with the desired compression speed up. The unskipped pixels are arithmetically coded as described in [2].

Hierarchical resolution reduction (RR) of binary images transforms a single image

Binary Image **Pixel - Skipping Prediction** **Compressed Image Code**

Resolution Reduction → Typical Prediction → Deter-ministic Prediction → Cond-itional Predictive Model → Adaptive Arithmetic Coding →

Matched Tables (PRES, QT etc.)

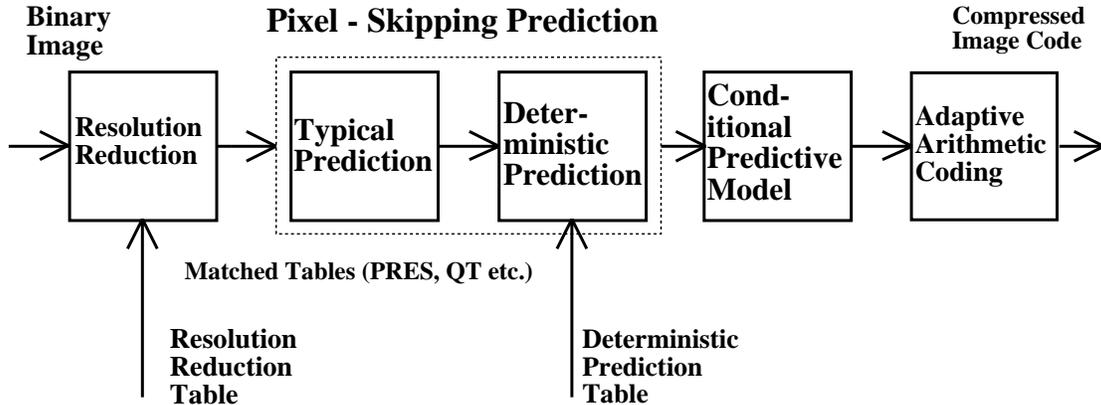**Resolution Reduction Table**

Deterministic Prediction Table

Figure 1: Block Diagram of JBIG Compression Encoder

into a pyramid of progressively smaller images. In keeping with the JBIG standard, we assume that resolutions are reduced symmetrically by factors of two. Each *layer* of the resolution reduction pyramid is created using information from the previous higher resolution layer and possibly the present layer. Each *pixel* in the next lower resolution layer replaces four pixels in the higher resolution layer. Typically, the same method is used to create each successive layer of the resolution reduction pyramid.

The PRES resolution reduction method suggested in JBIG is more complicated than the QT method. Its purpose is to generate high quality, i.e. visually appealing, lower resolution images. To determine the value of a resolution reduced pixel, the PRES method uses three pixels from the same layer and nine pixels from the previous higher resolution layer. A complete description of this method is given in [2]. In contrast to this, the quadtree reduction method uses only four higher resolution pixels.

Figure 2 illustrates results from the PRES and QT methods on a 256×256 pixel image, assuming five resolution reductions for both. The resolution reduced images in this figure, as well as in Figure 3, are shown with pixels representative of their resolution. Note that with QT reduction, successive lower resolution images rapidly become more black resulting in a faster loss of quality when compared with PRES reduction. Total illegibility of the QT RR images occurs about one resolution reduction sooner than that of the PRES RR images. Coding time, measured in terms of the number of arithmetically encoded pixels, is our design objective rather than the quality of resolution reduced images as was done previously.

Pixel prediction exploits the fact that hierarchical resolution reduction layers, being multiple representations of the same image, contain highly related and sometimes redundant information. Such prediction can be used to skip over and thus reduce the number of pixels to be arithmetically coded, thereby increasing compression speed.

Typical Prediction (TPB or TPD). There are two methods for typical prediction: one tailored for progressive (TPD) and the other for non-progressive (TPB) JBIG [2].

To determine if a 2×2 *quad* of higher resolution pixels is typical or not, TPD [2] uses a 3×3 neighborhood of lower resolution pixels. A quad is defined to be typical if all the pixels in it and all the pixels in the nine nearest lower resolution neighbors
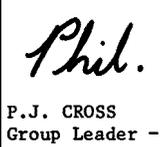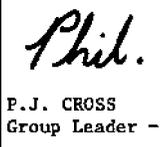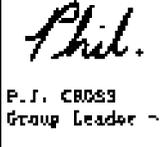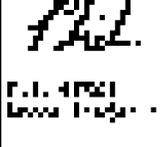
dpi	PRES	% Black	QT	% Black
200		11.18		11.18
100		11.06		13.85
50		11.67		18.43
25		13.09		25.49
12.5		16.02		41.02
6.25		18.75		57.81
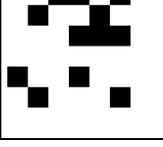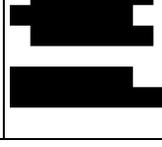
Figure 2: Resolution Reduced Images using the PRES or QT Methods.

have the same value. For example, a quad is non-typical if the nine nearest neighbors have the same value but one or more of its four higher resolution pixels differ from that value. A higher resolution line-pair of pixels is non-typical, if it contains *any* non-typical quads. If a *line-pair* of pixels is typical, then all of its typical quads can be skipped during arithmetic encoding.

When TPD is specified during encoding and decoding, a flag bit is used to indicate typical behavior for each line pair in each image of the resolution reduction pyramid, except that the lowest resolution layer is encoded using TPB. Although these flag bits typically have a very skewed distribution, i.e. they are easy to compress, they do reduce the compression ratio. As such, TPD trades the coding of a number of predictable pixels for the coding of a few flag bits. The second column in Figure 3 illustrates the pixels skipped using TPD for the PRES RR images of Figure 2. In contrast to Figure 2, the black pixels in Figure 3 are pseudo-images that represent the arithmetically encoded pixels.

TPB uses the previous scan line to predict all the pixels in the next scan line of image data. A flag bit is encoded at the beginning of each line to indicate whether "typical" prediction held true for that line.

Deterministic Prediction (DP). DP [2] uses information from lower resolution pixels to *exactly* determine the value of the next higher resolution pixels. Since the DP rules are based on inversion of the RR method, these methods must be matched. Any higher resolution pixels that can be exactly determined are again not arithmetically encoded. For example, assume that the RR method used determines each lower resolution pixel by finding the logical "AND" of the four associated higher resolution pixels. Then, if a lower resolution pixel has a value of one, its corresponding higher resolution pixels also have values of one and therefore can be predicted. The third column in Figure 3 illustrates the arithmetically encoded pixels using DP for the PRES RR images. Notice how few of the arithmetically encoded pixels are skipped with DP, that those skipped are concentrated near the edges and that they complement the pixels skipped using TPD. Although few in number, DP skipped pixels contribute more heavily to compression gains. The edge pixels skipped by DP, contain more information than the background pixels skipped using TPD.

Typical and Deterministic Prediction (TPD/DP). When TPD and DP are combined, they are commutative and can essentially be performed in parallel, i.e. any pixel that is typically predicted is not arithmetically encoded and any pixel that is deterministically predicted is also not arithmetically coded. The fourth column of Figure 3 illustrates the arithmetically encoded pixels using TPD/DP for the PRES RR images.

3 JBIG Compliant Quadtree Algorithm

The QT algorithm can be implemented using the JBIG standard with a user defined RR table, and a matched DP table. Therefore to completely specify QT it is sufficient

to describe the RR and the DP methods.

The QT RR method uses a logical OR of four higher resolution pixels to determine each lower resolution pixel, i.e. if all four higher resolution pixels are zero then the corresponding lower resolution pixel is zero, otherwise it is a one.

The arithmetic coder following the QT DP method skips the coding of higher resolution pixels if their corresponding lower resolution pixel is zero. Therefore, any time a pixel with a value of zero is encountered during arithmetic encoding all corresponding higher resolution pixels need not be coded since all their values are zero. This is true by construction because of the QT RR method.

The QT DP method does not fully exploit the information in the RR pyramid. For example, if three higher resolution pixels in a given quad are zero but the associated lower resolution pixel is one then the remaining pixel is known to be one. This addition to the RR method would reduce the total number of pixels that are arithmetically encoded, but would slow down a hardware realization because of the required additional pixel "reads". Since this type of prediction increases the speed by less than 1% and decreases the compression by 0.5%, the cost of additional pixel reads outweighs the speed benefits obtained.

Figure 3 illustrates the operation of TPD and DP with QT RR using the QT images from Figure 2. Note the horizontal stripes in the D5 QT TPD column, which are caused by non-typical line pairs in the hierarchical image. One can visualize the superior pixel-skipping achieved using QT, by comparing the D5 PRES TPD/DP colum with rightmost, D5 QT TPD/DP, column of Figure 3.

4   Relative Speed and Compression of QT

The algorithms' speed and compression are compared. We define algorithm speed to be inversely proportional to the number of pixels presented to the arithmetic coder. To benchmark our fastest JBIG compliant QT compression algorithm (D5 QT-TPD/DP) we compare it to the basic non-progressive JBIG algorithm D0 known to achieve the best compression, and to the suggested progressive JBIG algorithm using PRES (D5 PRES-TPD/DP) which previously achieved the best speed up known from skipping arithmetically coded pixels. This comparison is done using a set of eight CCITT test images. Each image has 1728×2376 pixels, but because of software limitations the y dimension was reduced to 2304 by removing the last 72 lines. This results in 1728×2304 images containing 3,981,312 pixels. The removed lines in each image compress greatly since they are of uniform color and thus have relatively little impact on the results.

Figure 4 compares the relative speed of compression achieved by the D5 QT-TPD/DP, D5 PRES-TPD/DP, D0-TPB and D0 using D0 as a reference. Note that our QT algorithm is in all cases faster than the other three algorithms. On average, it is 1.6 times faster that the PRES algorithm, 6.8 times faster than the D0-TPB algorithm, and 9.4 times faster than the D0 algorithm used as a reference. The compression ratios of the four algorithms are compared in Figure 5. The D0-TPB algorithm achieves an average compression similar to that of the reference D0 algo-

Figure 3: Arithmetically Coded Pixels for PRES and QT Reduction using DP, TPD or TPD/DP Prediction. The RR is the same as that in the previous figure.

rithm. Our algorithm achieves an average of 2% better compression than the PRES algorithm and 5% less compression than the D0 algorithm. Note that our algorithm also consistently achieves better compression than the PRES algorithm (D5 PRES-TPD/DP). Tables 1 and 2 contain the results used to generate the figures.

5   Conclusion

We described a JBIG compliant quadtree based compression algorithm, intended for integration with a bilevel area image sensor on the same chip. We demonstrated that our algorithm is significantly faster than basic non-progressive JBIG, and progressive JBIG using PRES, while achieving comparable compression. We believe that our algorithm may also lend itself to faster operation even when implemented in software.

Another form of parallelism utilizes multiple adders to increase the speed of arithmetic coding [4]. A hardware architecture displaying this type of parallelism was recently presented by Feygin, *et al.* [5]. Note that multiple adders occupy more silicon real estate than simple quadtree logic even when the quadtree logic must be replicated for every scan line of an area image sensor. Furthermore, this parallel arithmetic coding technique exploits the same contiguous image background regions as a quadtree front-end. We suspect that parallel arithmetic coding techniques will fail to increase coding speed if background pixels are already skipped over by a quadtree front-end. Since the two methods appear to be mutually exclusive, we chose to concentrate on the quadtree parallelism.

Comparison of our results with those of Feygin, *et al.* [5] is not straightforward, since they investigated speeding up ABIC [6] rather than JBIG and used a slightly different scan of the CCITT test images. Although ABIC is a direct precursor of the basic JBIG non-progressive algorithm, it has a smaller nearest neighbor model and is based on the "Q" [7] rather than the "QM" adaptive arithmetic coder [2]. Nevertheless, the results are quite consistent with our observation that both approaches exploit image background regions. Their reported average speed up, 7 to 13 times depending on complexity, is similar to our results. Moreover, the speed up per CCITT test image is also quite similar.

We note that the CCITT test images used are representative of typical business documents, but do not include "digital halftones". We do not expect speed up performance increases for our algorithm or other algorithms mentioned here when applied to halftone images.

References

[1] T. Markas et al., "Quad tree structures for image compression applications," *INFORMATION PROCESSING & MANAGEMENT*, vol. 28, no. 6, 1992.

[2] International Telegraph and Telephone Consultative Committee (CCITT), *Progressive Bi-level Image Compression*, Recommendation T.82, February 1993

[3] Sheinwald et al., "Deterministic prediction in progressive coding," *IEEE Transactions on Information Theory*, vol. 39, no. 2, March 1993.

[4] W.B. Pennebaker and J.L. Mitchell, *JPEG Still Image Data Compression Standard,* Van Nostrand Reinhold, New York, 1993.

[5] G. Feygin, P.G. Gulak, and P. Chow, "Architectural Advances in the VLSI Implementation of Arithmetic Coders for Binary Image Compression," in *Proc. 1994 Data Compression Conference*, Snowbird, Utah, March 1994.

[6] R.B. Arps, T.K. Truong, D.J. Lu, R.C. Pasco, and T.D. Friedman, "A Multipurpose VLSI Chip for Adaptive Data Compression of Bilevel Images," *IBM J. of Research and Development*, vol. 32, no. 6, Nov. 1988.

[7] W.B. Pennebaker, J.L. Mitchell, G.G. Langdon, and R.B. Arps, "An Overview of the Basic Principles of the Q-Coder Adaptive Binary Arithmetic Coder," *IBM J. of Research and Development*, vol. 32, no. 6, Nov. 1988.
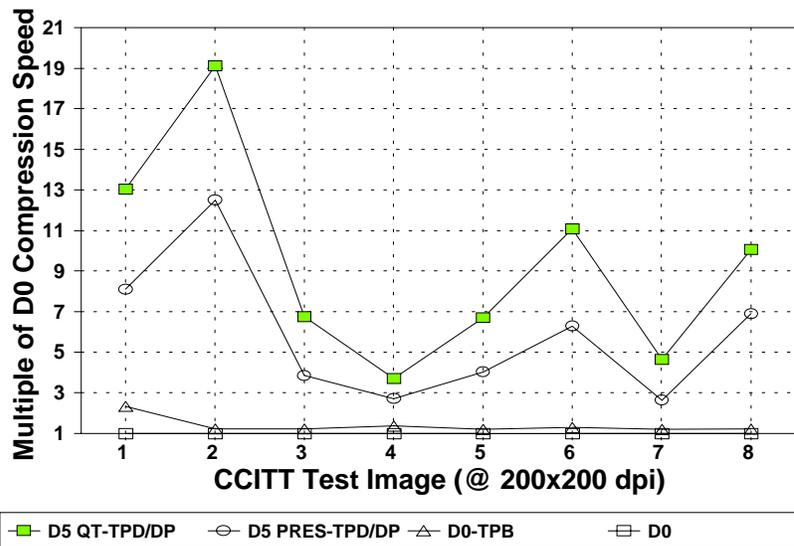
Figure 4: Comparison of JBIG Compression Speeds vs. Non-progressive JBIG

Comparison of JBIG Compression Ratios
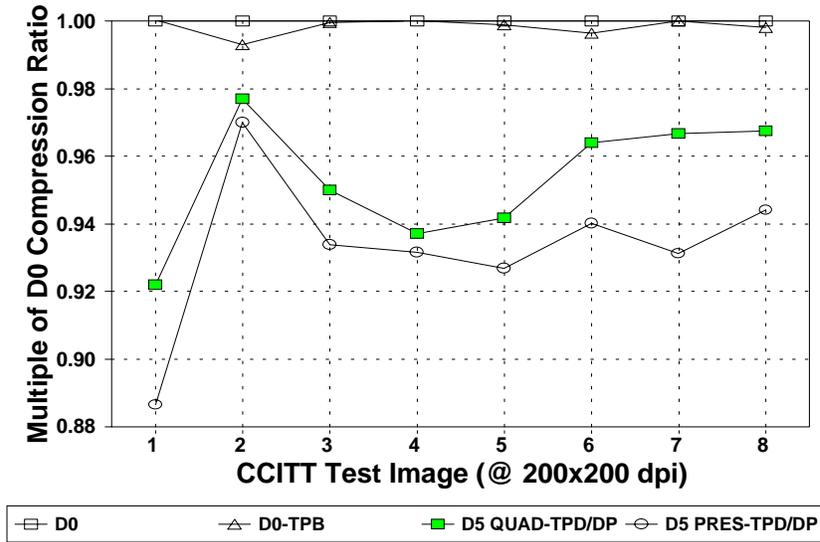### Pixel Skipping Modes vs. Basic D0 Mode

Figure 5: Comparison of JBIG Compression Ratios vs. Non-progressive JBIG

	D0	D0 TPB	D5 PRES TPD	D5 PRES DP	D5 PRES TPD/DP	D5 QT DP	D5 QT TPD	D5 QT TPD/DP
Arithmetically Coded Pixel Count For CCITT Images 1–8								
1	3981312	1703808	604472	4021440	490978	320616	1173044	305348
2	3981312	3252096	380464	4035178	318575	325348	851048	208152
3	3981312	3227904	1274300	4065718	1031557	637812	2171528	589192
4	3981312	2897856	1784552	4112969	1469132	1089676	2667868	1074352
5	3981312	3352320	1213000	4063504	987180	623752	2268912	594128
6	3981312	3077568	775636	4039549	633469	399960	1766076	359480
7	3981312	3326400	1876992	4087863	1514283	856240	3876300	854680
8	3981312	3305664	693096	4512112	576888	2312636	1642388	395876

Table 1: Results for Non-progressive JBIG, and QT or PRES Progressive JBIG

	D0	D0 TPB	D5 PRES TPD	D5 PRES DP	D5 PRES TPD/DP	D5 QT DP	D5 QT TPD	D5 QT TPD/DP
Compressed Image Size (Bytes) For CCITT Images 1–8								
1	14655	14650	17447	16526	16533	15787	16496	15895
2	8456	8515	9128	8734	8718	8563	9179	8655
3	21907	21916	24633	23432	23460	22893	23723	23061
4	53925	53921	60347	57949	57890	57320	58346	57546
5	25792	25823	29216	27805	27828	27194	28160	27389
6	12520	12566	13989	13294	13317	12850	13584	12987
7	56210	56211	64124	60280	60362	58062	59101	58146
8	14197	14223	15673	15038	15037	14625	15299	14674

Table 2: Results for Non-progressive JBIG, and QT or PRES Progressive JBIG