Context Selection for Lossless Compression of Bi-Level Images

Evaristo Ramalho and Eduardo Peixoto

***Abstract*— Bi-level images are used in many fields to represent binary masks for object and region of interest coding, enhancement layers of color images and point cloud occupancy maps as geometry slices. Compression of bi-level images is usually performed using context adaptive arithmetic coding. In this paper, we propose a low complexity context selection algorithm that chooses, for each image being encoded, the most useful contexts for encoding that particular image. This information is then transmitted in the bitstream at the cost of 16 bits. Results show that the proposed algorithm outperforms the typical, fixed contexts for natural images and, by a larger margin, for point cloud geometry slices.**

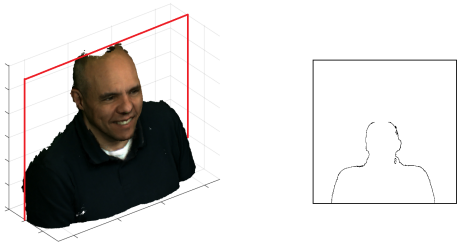
***Keywords*— Context Adaptive Binary Arithmetic Coding, Lossless Compression, JBIG.**

I. Introduction

A bi-level image, often called a binary image, is a 2D array of pixel values that can only take two values, usually depicted as black and white. Compression of this kind of images has been used in several applications for a long time. In the past, a widely spread application was document transmission using facsimile (fax) machines, which is one the motivations for the JBIG standard [1]. Nowadays, a bi-level image can be used to represent objects in image segmentation, for instance, when applying compression using region of interest techniques [2]. Pinho [3] uses bi-level image compression to encode the contour maps of a binary image, while Abdat et al. [4] proposed a scheme that represents a binary image with a Gray code, then encodes each bit plane separately as a binary image. Recently, some state-of-the-art encoders of point cloud geometry [5], [6] represent the point cloud occupancy map as an array of bi-level images. An example of this representation is shown in Fig. 1, where these images are referred as *point cloud slices*. Using this representation and a lossless bi-level image encoder, it is able to outperform other encoders that use an octree representation of these occupancy maps.

State-of-the-art compression of bi-level images is achieved by context adaptive arithmetic coding [7], which is used in both JBIG [8] and JBIG2 [9] [10] standards. In the basic form of this technique, the image pixels are encoded in raster-scan order, and some of the previously encoded pixels in the causal region of the image are used to form a context for the current pixel being encoded. This context is used to select a specific probability table that will be used to encode that pixel. After encoding this pixel, its value is used to update the probability table, giving the technique the "adaptive" name.

Evaristo Ramalho is with the Programa de Pós-Graduação em Engenharia Elétrica (PPGEE) at Universidade de Brasília (UnB), e-mail: evaristora28@gmail.com. Eduardo Peixoto is with the Departmento de Engenharia Elétrica at Universidade de Brasília (UnB), e-mail: eduardopeixoto@ieee.org.



(a) Point Cloud Ricardo (b) Point Cloud Slice

Fig. 1: Example of: (a) Point Cloud and (b) Point Cloud Slice for $Y = 350$. The slice represents only the geometry information for a given value $Y = y$. By using all slices $y \in [1, N]$ the geometry information can be transmitted in a lossless fashion.

Since we are dealing with bi-level images, for each new pixel that is considered in this context the size of this probability table doubles. This severely limits the number of pixels that are used in this context. The JBIG standard, for instance, can use up to 16 pixels, which gives 65536 possible contexts. However, it is usual to limit this number to the range $8 - 10$, since more contexts do not mean necessarily a better compression. Nevertheless, typical encoders use a fixed number of contexts at fixed, specified pixel positions.

In this paper we build a lossless bi-level image encoder, based on context adaptive arithmetic coding. However, in our encoder, any of the 16 pixels used as contexts can be added independently, and the information of which pixel positions provide the contexts is transmitted in the bitstream header. We then propose an algorithm to automatically select these context pixels in an efficient manner. Our algorithm is then a two-pass algorithm: it first scans the image to select which contexts are better suited to encode this image, and, after this decision is made, it encodes the image using only the selected contexts. We have made sure that this first pass is efficient, and much faster than actually encoding the image with the arithmetic coder, so that the complexity of the algorithm is kept low. The results show that our algorithm outperforms a similar context adaptive arithmetic coder using fixed contexts by a small margin for natural images, and by a large margin for the more challenging, highly sparse point cloud geometry slices.

II. Related Work

Several schemes were proposed to improve upon the JBIG coder. Fowler et al. [11] use a Quadtree based algorithm to speed-up the encoding with little loss in compression. Martins

et al. [12] propose a multi-pass algorithm that generates a tree, indicating which pixels are actually used as contexts, this approach is based on the idea, later shown by Fränti et al. [13] that not all contexts in JBIG are used equally, usually more than half of the bits are generated from the ten most important contexts, this work also compares the use of static trees, generated using training images, with optimizing this tree for specific images (in which case, the tree itself would need to be transmitted).

Other schemes creates new method to compress bi-level like Malvar [14], that proposed an encoder that uses two rules to compress the image: it computes the context-dependent probability of an image and uses it in a pixel prediction module, and adjusts a Run Length parameter, creating the BLC coder. Zahir et al. [15] propose a Near Minimum Sparse Pattern Coding, a type of block coding, that has low complexity compared to JBIG2 approach, while Zhou et al. [16] proposes a lossless approach with a new Direct Redundancy Elimination and Dynamic Context Mode to compress with arithmetic coding.

III. Proposed Work

In this paper we use our own implementation of a context adaptive binary arithmetic coder. The arithmetic coder itself is a standard integer implementation of an arithmetic coder, not any of its approximations [17]. Moreover, our encoder does not implement any resolution reduction, prediction or templates found in the JBIG standard [8]. The pixels are encoded in raster scan order, and some of the pixels already encoded (i.e., to the left and above the current pixel being encoded) are used to form the context used to encode that pixel. In this paper, we are considering up to 16 bits to form this contexts, at pixel positions shown in Fig. 2, where $\mathbf{p}$ is the current pixel being encoded. Pixels that fall outside the image region are always considered to be $\mathbf{0}$. Whenever we say we are using a specific number of contexts N we will be using the pixel positions in the range $[1, N]$ in this figure. Each context $\mathbf{C}$ has its own probability table, storing the probability for the current pixel within that context. This probability is initialized according to the number of $\mathbf{1}$s and $\mathbf{0}$s found in each context. Let n_0 be the number of zeros and n_1 be the number of ones. Then, the initial probability for that context will be $p(\mathbf{p}=0|\mathbf{C}) = \frac{n_0+1}{n_0+n_1+2}$ and $p(\mathbf{p}=1|\mathbf{C}) = \frac{n_1+1}{n_0+n_1+2}$. These offsets are added to avoid the probability being initialized as zero. For instance, for the four bit context $\mathbf{C}=\mathbf{1101}$ we will have $p(\mathbf{p}=0|\mathbf{C}) = \frac{2}{6} = \frac{1}{3}$ and $p(\mathbf{p}=1|\mathbf{C}) = \frac{4}{6} = \frac{2}{3}$. In typical fashion, this context is adaptive. At the encoder, the current pixel will be encoded using the current probability table. After encoding the symbol, the probability for that context will be updated according to the pixel encoded (i.e., the number of zeros or ones for that context will be updated). At the decoder, it will first decode the pixel, and then apply the same procedure to update the probability table, so that these tables will always be synchronized at both encoder and decoder.

Fig. 3 shows the results of encoding two binary images using this codec varying the number of pixels N considered in the context. We also show the n-th order entropy H_n for

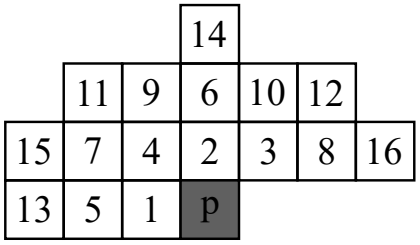
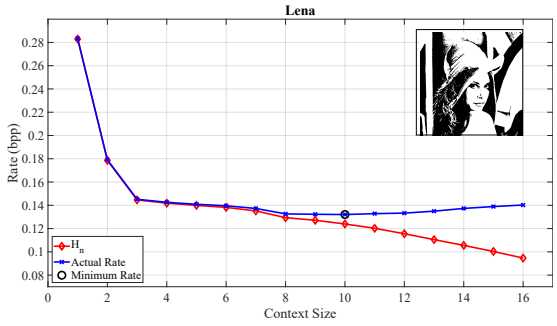


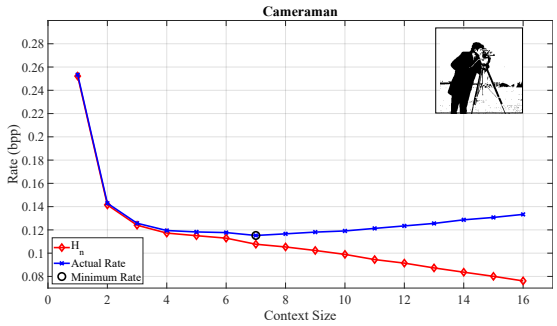
Fig. 2: The 16 contexts used in arithmetic coding.

that particular context. As shown by Shannon's Information Theory, this n-th order entropy approaches the image entropy when $n \to \infty$, and it is the minimum rate achievable by this particular probability model. Also, we have that $H_{n+1} \leq H_n$, i.e., adding more pixels to the context cannot increase the entropy, i.e., it cannot increase the achievable rate.

In Fig. 3 we can see that, initially, the output of our encoder closely follows the n-th order entropy. However, at some point, the output rate of the codec starts to diverge from the n-th order entropy, and it actually starts to *increase*. This is a known issue of context adaptive arithmetic coders [13] - if we have many contexts that occur too rarely, these contexts cannot adapt properly when encoding the image. The usage of these poorly adapted contexts leads to this increase in rate. More importantly, the number of contexts at which this inflection happens (i.e., the number of contexts where the minimum rate is achieved) is different for each image. For image Lena in Fig. 3a this is achieved at $N = 10$, while for image Cameraman in Fig. 3b it is achieved at $N = 7$.



(a) Lena



(b) Cameraman

Fig. 3: Bit-rate and n-th order Entropy in bits per pixel (bpp) for images: (a) Lena and (b) Cameraman.

A. Selecting Contexts

In the experiment shown in Fig. 3 we consider 16 different ways of using the contexts shown in Fig. 2, namely: $\{1\}, \{1,2\}, \{1,2,3\}, ..., \{1,2,3,..,15,16\}$. This is but a fraction of the possible 65536 different compositions that can be used. For instance, the pixel to the left of the current pixel is *always* included in the context, even if it is not correlated to the current image for that particular image.

In the proposed method we are making this context selection completely *independent* - each different pixel may be included in the context regardless of the other pixels. In order to transmit this information to the decoder we have included a small *context vector* in the bitstream header - just 16 bits indicating whether that particular numbered pixel position will be included in the context. With this modification we can select among the 65536 possible contexts. The issue is, now, how to choose between all possibilities. Naturally, one could use an exhaustive search algorithm and test all possibilities, but this would require encoding the image 65536 times, which would be too expensive in terms of computational cost. We have devised two simpler options, which are explained next.

B. Greedy Algorithm

Our first approach is inspired by the *greedy* class of algorithms [18]. In this type of algorithm, we attempt to optimize one choice at a time and, once this choice is made, it is regarded as optimal and we proceed to the next choice - the first choice is never re-visited, so this algorithm does not test all possibilities in the search space. It is not guaranteed to be optimal, though. For our algorithm, referred here as the *Greedy Algorithm*, we first test encoding the image using only *one* pixel as context, testing all 16 positions in Fig. 2. After encoding the image 16 times, we find the pixel position that, when added to the context, yields the minimum rate and add it to our solution - from this point on, it will always be used. Then, we test encoding the image using two pixels as context: the one selected in the first iteration, and one of the other 15 not chosen. We then select the position that yields the minimum rate and continue in this fashion, until, at most, the 16th iteration. A pixel position is only added to the solution if it yields a smaller rate than the previous solution, so the algorithm can stop before the 16th iteration (which it often does). In the worst case, however, it needs to encode the image $16 + 15 + 14 + ... + 2 + 1 = 136$ times to reach a solution. This is very small compared to the exhaustive approach above (0.2%), but it is still a large number, that limits the feasibility of this solution. We have therefore devised a fast algorithm to perform this selection, which is explained in the next section.

C. Fast Algorithm

The proposed algorithm is a two-pass algorithm. In the first pass we perform an analysis of the image. In this stage we specifically avoided costly operations, such as encoding the image or performing any type of arithmetic coding. After the analysis is performed, we make the decision of which pixel positions will be added to the solution. The second pass is where the image is encoded using this solution. This way, we avoid encoding the image twice, keeping the complexity on par with the typical algorithm.

In the first pass the algorithm visits all pixel positions gathering the complete probability table as if all 16 pixels would be used in the context. It stores, for each different context, the number of times a pixel **0** and a pixel **1** was found for that specific context. With this information the algorithm can compute the n-th order entropy for any combination, without needing to visit the image again. For instance, if the context numbers are ordered from the most significant bit (16) to the least significant bit (1), then it could compute the first order entropy of using the 16-th pixel by analyzing the two halves of the table, and it could compute the first order entropy of using the 1-th pixel combining the odd and even rows. By using different combinations with the table, all first order entropies can be computed.

Then we use a modified version of the greedy algorithm but, instead of encoding the image at each step, we just compute n-th order entropies using this table, which is a much less costly operation. The main idea is the same as the *Greedy Algorithm* described in the previous section. We compute the 1st order entropy for each pixel position, and then select the one with the lowest entropy. Then, we compute the second order entropy using the pixel position chosen in the first iteration paired with the other 15 pixels positions. However, since the n-th order entropy H_n is necessarily lower or equal to the $n-1$-th order entropy, the stop condition of the algorithm had to be modified. We use two stop conditions - if any of these is true, then the algorithm stops. Both conditions work by verifying the values H_n and $H_n - 1$, i.e., the value of the minimum entropy in the current and previous iteration, respectively. The verified conditions are:

$$\left|\frac{H_n - H_{n-1}}{H_{n-1}}\right| < \left|\frac{H_{n-1} - H_{n-2}}{H_{n-2}}\right| \quad (1)$$

$$|H_n - H_{n-1}| < T_1 \quad (2)$$

where T_1 is a thresholding parameter. Note that the first condition is triggered when the potential gain of adding a new pixel to the context is fairly small, while the second condition is triggered when the potential gain is smaller than in the previous iteration. Naturally, the second condition is only considered starting from the second iteration. If none of these conditions are satisfied, the algorithm will repeat the process. Normally the first condition is enough to secure a good number of contexts, however, for images with very low rates, the first condition starts to deviate too much from the optimal, so we need the T_1 threshold to ensure that the algorithm will not use too much contexts in these low rate images. Alternatively, the algorithm stops at the 16-th iterations, as at the moment only 16 pixels are being considered for context. In the second pass, the image is encoded with the selected context.

IV. Results and Discussion

In order to assess the efficiency of the proposed algorithm we have encoded some public available images comparing: (i-ix) our arithmetic encoder using fixed size contexts, from 3 to

TABLE I: Bit-rate results for natural images. All rates are in bits per pixel(bpp).

	Encoder using fixed-size context											
Image	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**JBIG**	**Greedy**	**Proposed**
Baboon	0.5029	0.4964	0.4941	0.4894	0.4870	0.4829	0.4812	0.4808	0.4820	0.4898	**0.4776**	**0.4776**
Barbara	0.2718	0.2562	0.2503	0.2441	0.2350	0.2313	0.2285	0.2216	0.2209	0.2165	**0.2153**	0.2281
Boat	0.1339	0.1294	0.1279	0.1278	0.1247	0.1233	0.1241	0.1252	0.1271	0.1242	**0.1225**	0.1238
Cameraman	0.1257	0.1195	0.1182	0.1180	0.1152	0.1166	0.1181	0.1191	0.1213	0.1211	**0.1151**	0.1160
Goldhill	0.2024	0.1962	0.1912	0.1868	0.1847	0.1807	0.1792	0.1801	0.1812	0.1807	**0.1780**	**0.1780**
Lena	0.1394	0.1361	0.1348	0.1335	0.1320	0.1272	0.1271	0.1268	0.1272	0.1355	**0.1255**	0.1320
Mountain	0.4135	0.3985	0.3883	0.3830	0.3798	0.3749	0.3733	0.3718	0.3717	0.3825	**0.3698**	**0.3698**
Peppers	0.1256	0.1185	0.1151	0.1141	0.1128	0.1097	0.1087	0.1085	0.1088	0.1130	**0.1075**	**0.1075**
Sails	0.1958	0.1921	0.1913	0.1893	0.1868	0.1817	0.1805	0.1793	0.1794	0.1838	**0.1761**	0.1763
Watch	0.1281	0.1258	0.1254	0.1224	0.1214	0.1182	0.1152	0.1147	0.1147	0.1133	**0.1121**	0.1171
Zelda	0.1274	0.1248	0.1214	0.1211	0.1203	0.1176	0.1178	0.1186	0.1200	0.1218	**0.1161**	**0.1161**
Average	0.2151	0.2085	0.2053	0.2027	0.2000	0.1967	0.1958	0.1951	0.1958	0.1984	**0.1923**	0.1947

TABLE II: Bit-rate results for point cloud slices. All rates are in bits per occupied voxel (bpov).

	Encoder using fixed-size context										
Image	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**JBIG**	**proposed**
Andrew9	1.955	1.940	1.933	1.930	1.981	1.973	2.050	2.130	2.220	2.752	**1.904**
David9	2.037	2.024	2.015	2.013	2.066	2.008	2.087	2.171	2.264	2.666	**1.935**
Phil9	1.984	1.981	1.970	1.956	1.999	1.970	2.037	2.104	2.182	2.705	**1.911**
Ricardo9	**1.915**	1.921	1.930	1.932	1.988	1.998	2.080	2.166	2.253	3.119	1.916
Sarah9	2.025	2.018	2.015	2.011	2.070	2.033	2.111	2.193	2.287	2.886	**1.939**
Average	1.983	1.977	2.064	2.060	2.122	2.089	2.164	2.244	2.333	2.826	**1.921**

11 (the contexts are added according to Fig. 2); (x) JBIG using the default parameters; (xi) the **Greedy** Algorithm described in Sec. III-B; and (xii) the **Proposed** Fast Algorithm described in Sec. III-C. We do not show the results from contexts 1,2 and $12-16$ due to the high values obtained in all images and point clouds tests. All codecs are lossless, and all bitrates are shown in bits per pixel. The images were first binarized using the MATLAB function *imbinarize*, which uses Otsu's method [19]. Our prototype implementation is available on GitHub [20]. In all our tests we have used the value $T_1 = 0.001$.

Table I shows the bit-rate results for a few popular images. It can be seen that our encoder is competitive with JBIG for all images. Also, it can be seen that using the fixed-sized contexts the optimal number of pixels in the contexts vary, ranging from 7 to 11 in the images tested. The best performing algorithm for all images is the **Greedy** algorithm, which shows that carefully selecting the pixels for the context is indeed a good strategy for improving the efficiency of this technique. The proposed fast algorithm matches the performance of the **Greedy** algorithm for 5 of the 11 images tested, and performs very closely for another 4 images. Only for images Barbara and Lena the proposed fast algorithm performs significantly worse than the **Greedy** algorithm. It is important to notice that all codecs perform closely to the n-th order entropy, and thus performance gains are expected to be small in absolute terms. Therefore, Table III shows the relative performance of the proposed fast algorithm compared to JBIG and using 8 and 10 fixed-size contexts.

When this type of codecs was applied to point cloud geometry slices it was reported that the JBIG standard performed poorly [5]. The main reason is that these slices are much more sparse than the typical binary images. We applied our technique to encode the slices generated by the first frame of five public available point clouds [21], in this case, we do not use the greedy algorithm due to the large amount of time to compress one single slice. These results are presented in absolute terms in Table II and relative terms in Table IV. Following the literature in point cloud compression, the rate is reported in *bits per occupied voxel*, i.e., the number of bits used divided by the number of occupied voxels in the point cloud. In this setup we encode only the point cloud slices that have at least one occupied voxel, and all slices are encoded independently, without any type of transformation - the idea here is to investigate the performance of the arithmetic coder itself. It can be seen from the tables that indeed JBIG performs significantly worse compared to the proposed codecs. Also, it can be seen that the optimal number of fixed-size contexts varies more than for natural images, from 3 to 8. In this test the proposed fast algorithm yields the best performance for all but one of the point clouds tested (Ricardo9).

Finally, our prototype MATLAB implementation is not suitable for extensive complexity tests. Nevertheless, in all our tests, the second pass of the proposed algorithm (the encoding stage) is only marginally more complex (4% slower) than encoding the image with a fixed-size context - the reason is a suboptimal computation of the context numbers, since in

the proposed algorithm they may not be neighbors, and the function to gather the context number for a given pixel is less optimized. Even though it is not optimized, the first stage of the algorithm is performed in under 2 seconds in Windows PC with an Intel Core i7-7500U processor with 8GB of RAM, which indicates the algorithm's low complexity.

TABLE III: Percentage gains in images between the proposed method and other approaches(8,10 and JBIG)

	Gains of proposed over		
Image	**8**	**10**	**JBIG**
Baboon	-1.1%	-0.7%	-2.5%
Barbara	-1.4%	2.9%	5.4%
Boat	0.4%	-1.1%	-0.3%
Cameraman	-0.5%	-2.6%	-4.2%
Goldhill	-1.5%	-1.2%	-1.5%
Lena	3.8%	4.1%	-2.6%
Mountain	-1.4%	-0.5%	-3.3%
Peppers	-2.0%	-0.9%	-4.9%
Sails	-3.0%	-1.7%	-4.1%
Watch	-0.9%	-2.1%	3.4%
Zelda	-1.3%	-2.1%	-4.7%
Average	-0.8%	-0.5%	-1.8%

TABLE IV: Percentage gains in point clouds between the proposed method and other approaches(4,8,10 and JBIG)

	Gains of proposed over			
Image	**4**	**8**	**10**	**JBIG**
Andrew9	-1.9%	-3.5%	-10.6%	-30.8%
David9	-4.4%	-3.6%	-10.9%	-27.4%
Phil9	-3.5%	-3.0%	$-\ 9.2\%$	-29.4%
Ricardo9	-0.2%	-4.1%	-11.5%	-38.6%
Sarah9	-3.9%	-4.6%	-11.6%	-32.8%
Average	-2.8%	-3.8%	-10.8%	-31.8%

V. Conclusion

This paper proposes a two-pass encoder for lossless compression of binary images. The encoding itself is performed in the second pass, using a context adaptive binary arithmetic coder. In the first pass our algorithm independently selects pixels to be included in the context used by the arithmetic coder. Our proposed algorithm is designed with low complexity in mind, avoiding costly operations in the first pass, and thus it is only slightly more complex than standard implementation of arithmetic coding. It also performs very closely to a high complexity Greedy Algorithm to select these contexts. Our tests show that our algorithm outperforms other techniques by a small margin for natural images, but it outperforms these techniques by a large margin for highly sparse point cloud slices. Future work includes adapting the algorithm for point cloud geometry coders using $2D$ and $3D$ contexts.

References

[1] V. Kyrki, "Jbig image compression standard," Apr. 1999.

[2] K. Khursheed, N. Ahmad, M. Imran, and M. O'Nils, "Detecting and coding region of interests in bi-level images for data reduction in wireless visual sensor network," in *2012 IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2012, pp. 705–712.

[3] A. J. Pinho, "A jbig-based approach to the encoding of contour maps," *IEEE Transactions on Image Processing*, vol. 9, no. 5, pp. 936–941, 2000.

[4] M. Abdat and M. G. Bellanger, "Combining gray coding and jbig for lossless image compression," in *Proceedings of 1st International Conference on Image Processing*, vol. 3, 1994, pp. 851–855 vol.3.

[5] R. Rosário and E. Peixoto, "Intra-frame compression of point cloud geometry using boolean decomposition," in *2019 IEEE Visual Communications and Image Processing (VCIP)*, 2019, pp. 1–4.

[6] E. Peixoto, "Intra-frame compression of point cloud geometry using dyadic decomposition," *IEEE Signal Processing Letters*, pp. 1–1, 2020.

[7] Youngjun Yoo, Young Gap Kwon, and A. Ortega, "Embedded image-domain compression using context models," in *Proceedings 1999 International Conference on Image Processing (Cat. 99CH36348)*, vol. 1, 1999, pp. 477–481 vol.1.

[8] ISO/IEC JTC 1 / SC 29 /WG 1 (ITU–T SG8), "Coding of Still Pictures," ISO/IEC, Tech. Rep. N 1359, Jul. 1999.

[9] P. G. Howard, F. Kossentini, B. Martins, S. Forchhammer, and W. J. Rucklidge, "The emerging jbig2 standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, no. 7, pp. 838–848, 1998.

[10] F. Ono, W. Rucklidge, R. Arps, and C. Constantinescu, "Jbig2-the ultimate bi-level image coding standard," in *Proceedings 2000 International Conference on Image Processing (Cat. No.00CH37101)*, vol. 1, 2000, pp. 140–143 vol.1.

[11] B. Fowler, R. Arps, A. El Gamal, and D. Yang, "Quadtree based jbig compression," in *Proceedings DCC '95 Data Compression Conference*, 1995, pp. 102–111.

[12] B. Martins and S. Forchhammer, "Tree coding of bilevel images," *IEEE Transactions on Image Processing*, vol. 7, no. 4, pp. 517–528, 1998.

[13] P. Franti and E. Ageenko, "On the use of context tree for binary image compression," in *Proceedings 1999 International Conference on Image Processing (Cat. 99CH36348)*, vol. 3, 1999, pp. 752–756 vol.3.

[14] H. S. Malvar, "Fast adaptive encoder for bi-level images," in *Proceedings DCC 2001. Data Compression Conference*, Mar. 2001, pp. 253–262.

[15] S. Zahir and M. Naqvi, "A near minimum sparse pattern coding based scheme for binary image compression," in *IEEE International Conference on Image Processing 2005*, vol. 2, 2005, pp. II–289.

[16] L. Zhou and S. Zahir, "A new efficient algorithm for lossless binary image compression," in *2006 Canadian Conference on Electrical and Computer Engineering*, 2006, pp. 1427–1431.

[17] M. W. Hoffman, *Lossless Compression Handbook*. Academic Press, 2003, ch. 17 - Lossless Bilevel Image Compression, pp. 327–350.

[18] T.Cormen and R. C.Leiserson, *Introduction To Algorithms*. The MIT Press, 1990, ch. 17 Greedy Algorithms, pp. 329–355.

[19] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[20] Evaristo Ramalho and Eduardo Peixoto, "Binary Image Coder Implementation." online, published on —October-2020. [Online]. Available: https://github.com/erlds/contextSelectCode

[21] "Microsoft voxelized upper bodies." [Online]. Available: http://plenodb.jpeg.org/pc/microsoft