UC Davis

IDAV Publications

Title

Efficient Error Free Chain Coding of Binary Documents

Permalink

<https://escholarship.org/uc/item/3236d8hs>

Authors

Estes, Robert R.
Algazi, Ralph

Publication Date

1995

Peer reviewed

Efficient error free chain coding of binary documents

Robert R. Estes, Jr.

estes@cipic.ucdavis.edu

V. Ralph Algazi

vralgazi@ucdavis.edu

CIPIC (Center for Image Processing and Integrated Computing)

2345 ASB, University of California, Davis 95616

Abstract

Finite context models improve the performance of chain based encoders to the point that they become attractive, alternative models for binary image compression. The resulting code is within 4% of JBIG at 200 dpi and is 9% more efficient at 400 dpi.

1 Introduction

In this paper, we re-examine chain codes from the viewpoint of error free data compression of binary documents. Using straight-forward context modeling techniques, we can encode the CCITT reference documents, using north-east-south-west (*NESW*) chain codes, at 1.079 bits/chain code symbol (bps). This is 46% less than the 0-order model and 32% less than the differential left-right-straight (*LRS*) version, which requires $\log_2 3$ bps. It is also 17% less than the theoretical upper bound of $\log_2(1 + \sqrt{2})$ derived by [2], which applies to a more constrained version of the chain representation that we use. When combined with an efficient encoding strategy for the beginning of chain (*BOC*) coordinates, we achieve a bit rate that is within 4% of that obtained using the JBIG binary image compression standard at 200 dpi and 9% less at 400 dpi.

In [6], the only previously published work on higher order context modeling of chain codes, the authors presented results obtained using a 2^{nd} order model based on Freeman's 8-way differential chain code representation [4]. They compressed 72 contour maps (of islands and countries) to an average of 1.519 bps. We have examined and extended their models to binary document images, obtaining 6-9% better results.

Given that chain codes can be represented more efficiently by finite context models and, whereas there appears to be little consideration of chain codes as an error free compression strategy in the literature, we have examined and carried out a substantial analysis of the performance of chain codes for the encoding of binary documents. In Section 2, we briefly discuss the various chain coding representations and their application to error free document compression and, since all representations perform similarly, based on simplicity arguments, we chose the *NESW* chain codes for further analysis, which we begin in Section 3. In Section 4, we attack the problem

Figure 1: Chain coding example. 4-connected contours are traced clockwise, originating from the open circles.

of efficiently encoding the *BOC* coordinates, after which, we present results for the complete encoding strategy for binary documents. We then conclude and present future directions and applications for this work.

2 Chain coding fundamentals

In chain coding, we describe a digitized binary object, or shape, by traversing the boundary of the object, recording the moves made at each step. Of the several common representations [2, 3, 4, 6], we chose an edge based representation in which the boundaries between pixels are traced instead of pixels themselves. A primary reason for this choice was the simplicity of the resulting representation and the lack of complications that arise, as compared to the other representations, when encoding fine structures. We will refer to codes designed based on this representation as *NESW* chain codes (for the north, east, south and west moves in its description.)¹

An example of a *NESW* chain code and a 4-directional pixel based code are shown in Figure 1. Comparing these chains, we see that in the edge based technique, backtracking is never required so that, effectively, it is a ternary source while, for the pixel based alternative, no such simplification exists. *NESW* chain sources are often represented as *LRS* codes to exploit this feature. Another important feature of edge based, 4-connected (those which trace 4-connected regions) *NESW* codes is that no explicit termination overhead is required; returning to its *BOC* terminates a chain.

We have chosen an edge based representation due to its simplicity, however, pixel based representations are more commonly used, in which case, backtracking and chain termination become problematic. Typically, researchers approximate the contours, forcing them to be perfectly 4- or 8-connected [2, 5], or break them into subsegments which satisfy these conditions [1], so that simpler representations can be used. Unfortunately, in an error-free environment, we do not have this option and cannot use these approximations.

¹In a thorough evaluation of the compressibility of the various chain code representations, all techniques performed similarly. In fact, the best representation (*LRS*) was only 1.3% better than the *NESW* representation and 3.3% better than the worst representation (the differential 8-way chain code).

	CCITT image number							
	1	2	3	4	5	6	7	8
chains	1526	185	1733	6154	2490	740	6701	567
symbols	96614	54466	168268	377184	180550	106990	350054	103650
length	63.3	294.4	97.1	61.3	72.5	144.6	52.2	182.8

Table 1: NESW chain code statistics. The standard CCITT images are 1728×2376 and were scanned at 200 dpi. The last row is the average chain length.

A complete representation of a binary image in terms of *NESW* chain codes consists of a list of chains, each of which is described by its *BOC* coordinates and a string of symbols from $\{N, E, S, W\}$ which define a closed contour.

3 Efficient chain coding

A straight forward representation of *NESW* chain codes requires 2 bps, but, exploiting the ternary nature of the source, we can easily reduce this to $\log_2 3 \approx 1.585$ bps. In the case of perfectly 4-connected contours, two rights or lefts cannot occur in sequence and an upper bound of $\log_2(1 + \sqrt{2}) \approx 1.272$ bps has been derived [2]. Even though we encode more complex chains than implied by this bound, often we still obtain results which are substantially below it.

Typically, chain code modeling has consisted of the development of the approximations mentioned in Section 2 and/or the use of chain difference codes. These models are definitely effective but, as was illustrated in [6], higher order models can result in substantial additional gains, especially when coupled with an efficient (arithmetic) encoder. Here, we extend those results by considering finite context models of all depths ≤ 8 .

Some basic properties of the CCITT document test set are given in Table 1, with the resulting compression results in Table 2.² Note that these results do not include *BOC* overhead or the initial segments of each chain used to prime the finite context model. Table 2 clearly illustrates the advantage of using higher order models. We have reduced the bit rate from the brute force 2 bps to 1.079 bps (at a context depth of 7). Furthermore, we use 15% less bits than does the second order model.

An important observation is that the entropy results are much smaller than the QM-code results³ for model depths of 6 and above. This is because the cost of the model is beginning to dominate the code, i.e., the overhead of implicitly transmitting the source probability estimates in the code stream is significant. Furthermore, there are not enough samples to obtain accurate probability estimates for some contexts.

²All totals presented in our tables are based on the additive complexity, i.e., we sum the bits required to encode the entire image set.

³The QM-code is the adaptive binary encoder used in both the JBIG and JPEG compression standards.

model depth	CCITT image number								
	1	2	3	4	5	6	7	8	total
0	1.998	1.998	1.997	1.998	2.000	1.999	1.995	1.999	1.998
	2.032	1.663	1.639	2.020	1.825	1.515	2.022	1.639	1.874
1	1.520	1.516	1.459	1.525	1.495	1.405	1.517	1.485	1.499
	1.550	1.402	1.303	1.560	1.424	1.200	1.539	1.334	1.457
2	1.306	1.271	1.176	1.307	1.244	1.052	1.361	1.217	1.269
	1.327	1.227	1.092	1.330	1.195	0.939	1.381	1.078	1.245
3	1.209	1.088	1.033	1.219	1.127	0.898	1.309	1.054	1.164
	1.232	1.088	0.991	1.239	1.105	0.844	1.333	0.982	1.160
4	1.158	0.986	0.947	1.172	1.057	0.798	1.264	0.966	1.100
	1.198	1.021	0.938	1.200	1.059	0.792	1.301	0.943	1.117
5	1.125	0.948	0.914	1.131	1.026	0.765	1.233	0.931	1.065
	1.184	1.015	0.928	1.165	1.045	0.784	1.280	0.937	1.097
6	1.079	0.920	0.882	1.075	0.985	0.730	1.200	0.901	1.024
	1.178	1.037	0.926	1.123	1.031	0.789	1.268	0.945	1.083
7	1.027	0.886	0.846	1.005	0.936	0.696	1.167	0.865	0.976
	1.194	1.075	0.935	1.077	1.032	0.813	1.273	0.970	1.079
8	0.948	0.838	0.797	0.921	0.872	0.645	1.120	0.816	0.914
	1.221	1.127	0.960	1.037	1.045	0.840	1.295	1.009	1.087

Table 2: Bits/symbol required to encode *NESW* chain codes. The upper number in each pair is an entropy result, while the lower was obtained using the QM-code.

For this reason, we use the QM-code complexity in the sequel as a measure of descriptive complexity, since it implicitly transmits the model to the decoder and, therefore, leads to fair comparisons. Work to close the gap between the entropy and QM-code results, by the design of better models, with fewer parameters, is left for future work.

Although not presented here, we also computed similar results for an 8-directional, pixel-based chain difference code, and encoded the second order model with 1.565 bps. The images processed in [6] were encoded with 1.519 bps, suggesting that our images are slightly more complex. At a depth of 5 we obtained our best result, for that representation, of 1.469 bps, which is 6.5% less than the result at depth 2. Furthermore, in terms of an overall compression ratio, the depth 7 *NESW* code outperforms the depth 2 8-directional chain difference code by more than 9%.

Given such an efficient representation of chain code symbols, we now consider the use of chain codes for document image compression. To do so, we must encode the *BOC* coordinates and the initial segments of the chains, which were not included in the results of this section. For the initial segments of each chain, the solution chosen is to keep an extra set of contexts, at all depths less than the depth of the finite context model being used, and then use only the previous symbols from the current chain to determine the corresponding context. The *BOC* encoding problem is more

encoding method	CCITT image number								
	1	2	3	4	5	6	7	8	total
entropy	12.83	15.88	12.65	10.82	12.13	13.88	10.70	14.26	12.12
0-pel	16.06	16.48	16.04	16.01	16.02	16.08	16.00	16.13	16.02
7-pel	10.61	16.39	11.45	9.35	10.95	12.92	10.24	14.29	10.46
<i>baseline</i>	10.49	16.74	11.47	9.39	10.84	13.09	10.57	14.11	10.56
<i>better</i>	7.50	12.97	8.58	6.65	7.83	9.91	8.38	10.92	7.90

Table 3: Encoding the *BOCs*. All results are in bits/*BOC*.

subtle.

4 Encoding the *BOCs*

A straight forward encoding of the *BOC* coordinates requires $\lceil \log_2(1728 \times 2376) \rceil = 22$ bits per chain. This representation for the *BOCs* leads to an overall compression ratio which is 12% worse than the results we present in Section 5. Clearly, attention to the efficient encoding of the *BOCs* is justified.

Intuitively, as the number of contours increases, we expect the cost per *BOC* to decline. This behavior is easily verified by encoding the binary image of *BOC* coordinates. For an image with n pixels and k chains, such a technique would result in $(n/k)H(k/n)$ bits per chain. For $n = 4105728$ and k between 185 and 6700, this value decreases monotonically from 16 to 11 bits/*BOC*. In Table 3, we compute these entropies and also verify that the QM-code can be used to encode these images efficiently. It is interesting to note that the 1 state (0-pel) QM-code does not achieve performance anywhere near the entropy of the model and, surprisingly, always requires 16 bits per contour.⁴ With a 7-pel predictor, however, the results are often better than predicted, implying that, even over the small support of the 7-pel raster scan context, a significant amount of redundancy exists in the *BOC* coordinate data.

What is it about document images that we might exploit? Typically, they consist of lines, spaced at regular intervals in the vertical direction, of characters which are spaced predictably in the horizontal direction. Intuitively, this suggests a line based representation.

In our *baseline* representation, we use a 4 state decomposition of the *BOC* coordinate data, as illustrated in Figure 2(b). We encode the offset to the next row in context *dr*, encode the offset to the first *BOC* in context *dc1*, and then encode each additional offset in the current row in context *dc*. After each *dc* or *dc1* is encoded, we encode an *eol* decision to indicate whether there are any more *BOCs* to encode

⁴The probability estimation mechanism in the QM-encoder generates very bad estimates for this case. The QM-code probability estimation mechanism is designed assuming that there are many intermixed contexts, so that the renormalization events are fairly random. For the single state case, this lack of randomization may be the cause of the poor estimates.

Figure 2: Efficient representation of the *BOC* coordinates.

on this line. All values, except the binary *eol* decision are encoded in unary using a single probability estimate. This model should be able to exploit dependencies in the data over greater distances than the 7-pel technique, but only for *BOCs* which lie on the same row. The results obtained using this representation are also shown in Table 3. Although, this technique is no more efficient than the 7-pel context, it uses only 4 states and significantly reduces the number of binary decisions that must be encoded.

The results discussed above are based on the original *BOC* coordinates which are located at the top-left corner of each contour. An important observation is that we can start the chain at any point along the chain, giving us the flexibility to chose better (more compressible) *BOCs*. One such encoding is shown in Figure 2(c), which encodes all of the *BOCs* on the 7th row and is obviously more compressible. Thus, our goal is to “rotate” the chains so that the *BOCs* occupy as few lines as possible.

We implemented a greedy version of such an algorithm, i.e., the number of chains passing through each row in the image is computed, the row which passes through the most chains (along with the corresponding chains) is removed, the counts are updated, and the process iterated until all chains are accounted for. Each chain is then rotated to use this new *BOC* coordinate. From the *better* row of Table 3, we see that such a technique reduces the *BOC* overhead by an additional 25% over our *baseline* representation. More importantly, we now have a *BOC* representation which costs approximately 1/3 as much as the original, brute force 22 bits/*BOC*.

There are still important systematic redundancies that we have not exploited for the *BOC* source. For state *dc1*, we could take line to line differences to exploit a constant left margin. Given the previous chain on the current line, which has already been decoded, we could use relative addressing techniques to encode the *dc* values. In text based documents, character spacing could be estimated and used to predict the next offset. Other encodings, in addition to unary, could be used. We have attempted to exploit a few of these factors, with little success. Furthermore, with our current encoding, the *BOC* overhead is only 8-12% of the overall bit rate, so that incremental improvements will offer little overall gain.

strategy		CCITT image number								
		1	2	3	4	5	6	7	8	total
raster-scan	7-pel	34.24	58.33	21.75	9.13	18.97	36.77	8.83	33.49	18.83
	10-pel	35.02	60.66	23.39	9.46	19.90	40.99	9.13	36.00	19.73
baseline	0	19.33	43.83	13.88	5.01	11.52	23.88	5.27	23.07	11.30
	1	25.10	51.79	17.32	6.45	14.61	29.95	6.86	28.20	14.41
	2	29.10	59.02	20.39	7.52	17.18	37.46	7.58	34.53	16.70
	3	31.18	65.93	22.18	8.06	18.40	41.26	7.84	37.66	17.81
	4	32.05	70.25	23.26	8.32	19.05	43.42	8.02	39.20	18.42
	5	32.48	70.51	23.45	8.55	19.26	43.63	8.13	39.37	18.70
	6	32.53	69.19	23.44	8.80	19.41	43.40	8.20	38.97	18.88
	7	32.18	66.59	23.18	9.08	19.38	42.18	8.17	38.00	18.90
	8	31.51	63.77	22.66	9.33	19.14	40.85	8.06	36.63	18.75
better	0	19.74	44.21	14.11	5.11	11.76	24.15	5.37	23.32	11.51
	1	25.52	52.39	17.48	6.56	14.83	30.16	6.96	28.44	14.61
	2	29.37	59.50	20.47	7.57	17.28	37.73	7.59	34.56	16.77
	3	31.49	66.47	22.32	8.10	18.55	41.48	7.83	37.85	17.89
	4	32.32	70.46	23.37	8.35	19.22	43.63	7.97	39.28	18.45
	5	32.65	70.70	23.56	8.58	19.41	43.76	8.06	39.38	18.71
	6	32.72	69.21	23.51	8.86	19.54	43.33	8.10	38.95	18.88
	7	32.21	66.75	23.20	9.17	19.42	42.02	8.07	37.96	18.88
	8	31.46	63.72	22.67	9.41	19.13	40.69	7.95	36.51	18.70

Table 4: Efficient chain coding of binary documents. (Results are expressed as compression ratios.)

5 Error free encoding of binary documents

Now that we have an efficient encoding strategy for chains and *BOC* coordinates, we can evaluate our complete binary document compression system and compare it with the raster scan finite context models widely in use. Table 4 summarizes our results, presenting compression ratios for 7- and 10-pel⁵ raster scan contexts and depth 0–8 finite context chain code models, for both our *baseline* and *better* *BOC* encoding strategies.

Comparing the chain code results, we see that the *better* results are not much better than the *baseline* results and, sometimes, are even worse. At depth 0, there is a 2% improvement, but as the depth increases, the gain decreases. This apparent inconsistency occurs because the *BOC* gain is counteracted by the added complexity

⁵The 10-pel context results were obtained using JBIG with the following parameters: $D = 0$, $L_0 = 2376$, $LRLTWO = TPDON = TPBON = DPON = 0$, which gives the best results for most images. Furthermore, the header, trailer and stuff bytes are included, as per the JBIG specification. This adds a .5% overhead to these results.

strategy	Initial symbol							
	1	2	3	4	5	6	7	8
<i>baseline</i>	16	744	1224	1504	1720	1840	1880	1904
<i>better?</i>	1192	2144	1968	2064	1968	2016	2152	2200
cumulative difference	1176	2576	3320	3880	4128	4304	4576	4872

Table 5: Chain initialization overhead.

of encoding the initial segment of the rotated chain. In the *baseline* technique, every chain is initiated at its top-left corner, leading to many constraints on the chain segment, e.g., the first symbol must be *E*, the second can only be an *E* or an *S*, and so on. In the *better* case, we always chose the *BOC* to be the leftmost intersection with the given row, but the initial segment is not as constrained as when the *BOC* is the top-left corner. We were aware of this trade-off but did not expect the two effects to essentially cancel one another, as they do for larger contexts. Furthermore, both effects are proportional to the number of chains in the image and, therefore, we do not expect this behavior to vary with the number of contours in the image. At higher resolutions, with smoother boundaries, our optimization may be beneficial, but in this case the *BOC* overhead is less important to the overall code, since the chains are typically longer.

Details illustrating this trade-off are presented in Table 5 for CCITT image 1. Each column presents the cost of encoding the n^{th} symbol in a chain for both techniques, as well as the cumulative differences between them. The total difference, for a depth 8 model is 4872, whereas, the *better* *BOC* encoding strategy only saves us 4560 bits.

Recall, however, that we are comparing two of the better *BOC* encoding techniques — we are still considerably better off than with 22 bits/*BOC*. Our optimization may be of use when smaller contexts are used and for images with a large number of chains, but for the results presented here, it is of little value, and we are better off using the *baseline* representation and the top-left *BOCs*.

Finally, in comparing the best *baseline* chain code results with best raster scan results (JBIG), we find that, on average, the chain coding technique is 4.3% worse. If we choose the depth that is optimum for each individual image, the corresponding relative inefficiency is 2.7%. The chain code representation is more efficient for 4 of the 8 images: 2, 3, 6 and 8, by as much as 16% for the graphical image CCITT 2, while being as much as 11% worse for the Kanji test image CCITT 7. This suggests that for simple images, i.e., those with smoother boundaries and fewer chains, chain coding is more effective. This is intuitively pleasing, since it would seem that there is more information about the shape of an object along its contour than in a causal raster scan neighborhood. Furthermore, and considerably more important, this suggests that chain coding techniques should be more effective for higher resolution images.

A closer analysis reveals that there is a strong correlation between the average

strategy	Stockholm image number						
	1	2	3	5	7	8	10
<i>NESW</i>	134.9	116.6	11.60	54.8	69.4	82.4	82.9
10-pel	122.4	109.1	11.40	49.0	69.8	191.7	218.4

Table 6: Chain code compression ratios for higher resolution (400 dpi) documents.

chain length and the performance of the chain codes. The images that the chain code performs better on have average chain lengths of > 95 , while the images it does worse on have average lengths < 75 . Although not necessarily the case, this seems to imply that the chain complexity is highly correlated with the average length of the chains. Higher resolution images will have longer chains and, thus, will be encoded more efficiently by chain codes.

In a simple experiment performed to determine the scaling properties of chain codes, we encoded some of the 400 dpi images from the JBIG Stockholm image test set, and obtained the results shown in Table 6. The chain code results use a depth 5 context. The results for documents 1, 2, and 5 support our conclusion that chain codes are more effective at higher resolutions. For these documents we achieve a 9% bit rate reduction with respect to JBIG’s 10-pel raster scan context. Of the remaining documents, image 3 is a very complex image, which has features similar to the CCITT 200 dpi documents, and the remaining three images are line drawings, with images 8 and 10 consisting of single pixel wide lines. Our chain code representation is very inefficient in latter case because it traces both the inside and outside of these lines, essentially encoding them twice. The 10-pel predictor however captures this structure nicely.

6 Discussion

In this paper, we have used higher order finite context models to obtain chain codes which perform, on average, nearly as well as the widely used causal raster scan techniques, and which perform significantly better on the simpler graphical images. This suggests that at higher resolutions, chain codes should be more efficient than raster scan based techniques. This is intuitively pleasing since higher order context models can be interpreted as a method of obtaining a local estimate of the shape of the object, which seems to tell us much more about the object than a causal raster scan neighborhood; with increasing resolution, a raster scan neighborhood degrades to a *smaller* estimate of the boundary, since it does not extend far enough to pick up other intra- and inter-object dependencies.

One important issue which we have overlooked, but will be considering in future work, is the modeling problem. Here, we chose the brute force modeling technique, i.e., to use an increasing number of previous symbols as our context, and, for larger contexts, the modeling complexity begins to dominate the code. To close the gap

between the actual results and the entropy, we need to reduce the number of contexts and, hence, the amount of information implicitly transmitted to the decoder to describe the model. As we increase the depth of conditioning, chain coding begins to look like object recognition techniques, since the modeling task corresponds to extracting the significant boundary features of the chains.

There are many dependencies within chains that might be exploited by better models. Since the number of E s and W s are equal, as well as the number of N s and S s, one could leave out any two consecutive chains of the contour, if its length was known. For small chains, there are very few possible shapes, which might be better encoded via enumeration or vector quantization techniques.

In conclusion, we have shown that at 200 dpi binary documents can be encoded with chain codes almost as efficiently as they are by causal raster scan context models (within 4.3%) and that for 400 dpi binary documents, they can be compressed approximately 10% more. Furthermore, we have presented the most efficient chain codes to date, representing the CCITT images with an average of 1.079 bits/chain code symbol. This result is 6-9% better than the best previously known result [6]. Finally, these results have been obtained without significant modeling effort, which implies that substantial improvements may be possible.

7 Acknowledgment

This research was supported in part by the UC MICRO program, Hewlett Packard, Lockheed and Pacific Bell.

References

- [1] B. B. Chaudhuri and S. Chandrashekhar. Neighboring direction runlength coding: An efficient contour coding scheme. *IEEE Transactions on Systems, Man and Cybernetics*, 20(4):916–21, July 1990.
- [2] Murray Eden and Michel Kocher. On the performance of a contour coding algorithm in the context of image coding: Part I. *Signal Processing*, 8:381–6, 1985.
- [3] H. Freeman. Computer processing of line-drawing images. *Computing Surveys*, 6(1), March 1974.
- [4] H. Freeman. Map and line-drawing processing. In J. C. Simon and R. M. Haralick, editors, *Digital Image Processing*. D. Reidel Publishing Co., 1981.
- [5] Woosung Kim and Rae-Hong Park. Contour coding based on the decomposition of line segments. *Pattern Recognition Letters*, 11:191–5, March 1990.
- [6] Cheng-Chang Lu and James G. Dunham. Highly efficient coding schemes for contour lines based on chain code representations. *IEEE Transactions on Communications*, 39(10):1511–4, October 1991.