Lossless Compression of Map Contours by Context Tree Modeling of Chain Codes

Alexander Akimov, Alexander Kolesnikov, and Pasi Fränti

Department of Computer Science,
University of Joensuu,
P.O. Box 111, 80110 Joensuu, Finland
{akimov, koles, franti}@cs.joensuu.fi

Abstract. We consider lossless compression of digital contours in map images. The problem is attacked by the use of context-based statistical modeling and entropy coding of chain codes. We propose to generate an optimal context tree by first constructing a complete tree up to a predefined depth, and then create the optimal tree by pruning out nodes that do not provide improvement in compression. Experiments show that the proposed method gives lower bit rates than the existing methods for the set of test images.

1 Introduction

Digital maps are usually stored as vector graphics in a database for retrieving the data using spatial location as the search key. The visual outlook of maps representing the same region varies depending on the type of the map (topographic or road map), and on the desired scale (local or regional map). Vector representation is convenient for zooming as the maps can be displayed in any resolution defined by the user. The maps can be converted to raster images for data transmission, distribution via internet, or because of incompatibility of the vector representations of different systems.

In order to increase the efficiency of raster map compression, we consider the variant, when some object, instead of being rasterized, will be described by *chain codes* and compressed separately from rest of map data. This can lead to a more efficient representation of the map and, consequently, to improve of compression. Chain coding is a common approach for representing different rasterized shapes such as line-drawings, planar curves and contours. We consider thin digital curves of one pixel width, extracted from the vector data before rasterization.

The previous works consider different schemes of encoding and chain code representation [3], [10], [11]. For example, the method in [12] uses second order context model based on 8 directional chain codes. Further development of the context-based compression of chain codes was presented in [4]. The authors have improved the performance of the chain codes encoding by increasing the size of finite context models. The problem of encoding of chain codes by *prediction by partial matching* (PPM) algorithm [2] has been considered in [1].

In principle, context based compression can be improved by using a larger number of neighboring symbols in the context. But the increase of the context size leads to the

H. Kalviainen et al. (Eds.): SCIA 2005, LNCS 3540, pp. 312 – 321, 2005.

problem of context dilution, in which the statistics are distributed over too many contexts, and thus, affects the accuracy of the probability estimates.

Context tree provides a more flexible approach for modeling the contexts so that a larger number of neighbor pixels can be taken into account without the context dilution problem [13]. The context tree algorithm was originally introduced in [16], and analyzed in [10]. Practical solutions for the context tree based compression algorithms for grey-scale and bi-level images have been described at [19] and [13] respectively.

In this paper, we use the context tree approach for encoding the chain codes. We provide algorithm for optimal context tree construction. We compare the compression performance of the rasterized map contours when encoded by JBIG [9], and by the optimal context tree chain codes, representing the same contours. The results show that the proposed method provides 25% lower bit rate than JBIG, and is 40% faster because only the contour pixels need to be processed.

The overall scheme of the proposed compression method is as follows:

Step 1: Extract contours from the vector or raster map and convert them into chain codes. Store the start points and the lengths of the chains.

Step 2: Create and store the optimal context tree for the chain codes.

Step 3: By using of context tree modeling and any entropy coding, encode the chain codes.

This scheme is shown in Figure 1. For simplicity, we store the size and the beginning of each chain (BOC) as such without any further compression.

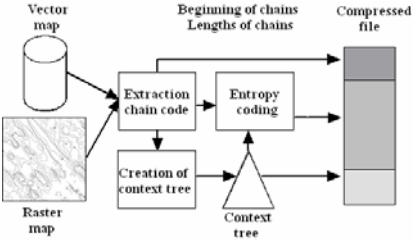


Fig. 1. Overall system diagram of the proposed method

2 Chain Code Representation

Freeman [5], [6] proposed chain coding of digital contours drawings and description. The chain codes represent the digital contour by a sequence of line segments of specified length and direction, see Figure 2. We consider both 8- and 4-directional chain coding schemes; see Figure 3.

The chain code representation is constructed as follows.

Step 1: Select a starting point of the contour. Represent this point by its absolute coordinates in the image.

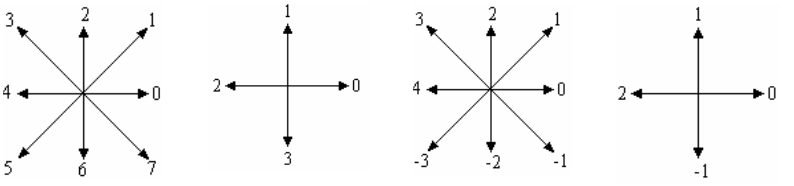


Fig. 2. 8-connected and 4-connected chain codes and their differential chain codes

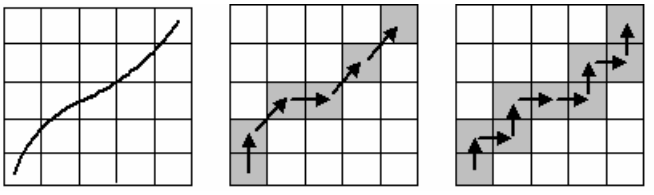


Fig. 3. An example of chain code construction: original curve (left), 8-directional (center) and 4-directional (right)

Step 2: Represent every consecutive point by a chain code showing the transition needed to go from the current point to the next point on the contour.
Step 3: Stop if the next point is the initial point, or the end of the contour. Store the lengths of the contours into the file.

An alternative way for chain code representation are differential chain codes [5]. Each differential chain code k_i is representing by the difference of the current chain code c_i and the preceding chain code c_{i-1} : $k_i = c_i - c_{i-1}$.

The chain codes of contours can be extracted from the map image in two ways. Firstly, if the map is obtained directly from the map database, we can extract contours directly from the vector data. On the other hand, if the map is provided as a color raster image, we can use *color separation* and following by *vectorization* (must be used to extract the contours).

3 Context Tree Modeling

3.1 Finite Context Modeling

We compress the chain codes sequentially according their order in the input data. Consider the current symbol x_i and the string of the M previous symbols $x_{i-1},...,x_{i-M-1}$ denoted as x^{i-1}. In the *context based modeling* the probability of the next symbol x_i is conditioned on its *context* x^{i-1}. The probabilities of the symbols generated in a given context, are treated as independent [17]. Thus, a model becomes a collection of independent sources of random variables. By the assumption of independence, it is easy to assign probabilities to each new symbol generated at the current context. Let us denote the cardinality of the alphabet of the encoded data as N. If

$n_1(x^{i-1}),...,n_N(x^{i-1})$ are the counts of all symbols generated at the given context x^{i-1}, then the conditional probability of the event $x_i = k,\ k \in [1,..,N]$ is:

$$p\left(x_i = k \middle| x^{i-1}\right) = \frac{n_k(x^{i-1})}{\sum_{j=1}^{N} n_j(x^{i-1})} \tag{1}$$

We consider the encoding of the given statistical model by entropy-based encoder. The probability for the entropy-based coder is estimated as:

$$p\left(x_i = k \middle| x^{i-1}\right) = \frac{n_k(x^{i-1}) + \delta}{\sum_{j=1}^{N} n_j(x^{i-1}) + N \cdot \delta} \tag{2}$$

The parameter δ here depends on different arithmetic coders, but it usually equals to $1/N$ [8], [14].

3.2 Context Algorithm Revisited

Context tree is applied for the compression in the same manner as the fixed size context; only the context selection is different. It is made by traversing the context tree from the root to a terminal node, each time selecting the branch according to the corresponding previous symbol value. If the corresponding symbol points to a non existing branch, or the current node is a leaf, then we came to a terminal node, which points to the statistical model that is to be used.

The context tree can be constructed beforehand (*static approach*) or optimized directly for the encoded data (*semi-adaptive approach*). In the second case, the tree structure must be stored in the compressed file. The process of optimal tree construction consists of two main phases: initialization of the context tree, and pruning of the tree.

3.3 Construction of Initial Context Tree

To construct an initial context tree, we process the image to collect statistics for all potential contexts, leaves and internal nodes. Each node stores information of *N* counts for all symbols generated at the current context. The algorithm of the context tree construction is:

Step 1: Create a root of the tree.

Step2: For all *i* = 1 to *n*, traverse the tree along the path defined by the past string x^{i-1}. If some indices of the symbols in x^{i-1} are less than one, then set these symbols to zero. If some node, visited according the correspondent symbol of the string x^{i-1}, does not have a consequent branch (for transition to the next symbol of x^{i-1}), then create the necessary child node and process it. Each new node has *N* counts, which are initially set to zero. In all visited nodes, increase the count of x_i by 1.

This completes the construction of the context tree for all possible contexts. The time complexity of this algorithm is O(n).

3.4 Construction of Optimal Context Tree

The initial context tree needs to be pruned by comparing the parent node and its children nodes for finding the optimal combination of siblings. Let us denote by $c(T)$ the number of bits, required to store the tree structure in the compressed file. For different strategies of the tree construction it will be different:

$$c(T) = \begin{cases} 0, \text{static approach} \\ K, \text{ semiadaptive approach, complete tree} \\ N \cdot K, \text{ semiadaptive approach, incomplete tree,} \end{cases} \tag{3}$$

where K is the cardinality of the tree T. We will denote the set of all terminal nodes as $S(T)$. Let us denote as $n_i(s)$, $s \in S(T)$, the count of the symbol i, encoded by the statistical model, pointed by the terminal node s. By the cost of a terminal node s here we understand the following expression [7], [13]:

$$\tilde{c}\left(n_1(s), n_2(s), \ldots n_N(s)\right) = \begin{cases} 0, \text{ if } n_1(s) = n_1(s) = \ldots = n_N(S) = 0 \\ -\log_2 \dfrac{\prod_{i=1}^{N} \prod_{j=0}^{n_i(s)-1} (j+\delta)}{\prod_{j=0}^{n_0(s)+n_1(s)+\ldots+n_N(s)-1} (j+N\cdot\delta)}, \text{ otherwise.} \end{cases} \tag{4}$$

This definition corresponds algorithmically to the use of a one pass arithmetic coding without the update of the statistical model [8]. By the cost of the context tree T, we will denote the following expression:

$$L(T) = c(T) + \sum_{s \in S(T)} \tilde{c}\left(n_1(s), n_2(s), \ldots, n_N(s)\right) \tag{5}$$

The problem of the tree pruning is to modify the structure of the full context tree so that the expression (5) will be minimized. For solving this problem, we use a bottom-up algorithm [15]. The main principle of this algorithm is that the optimal tree consists of optimal sub-trees.

For any node t from the tree T, let us denote the vector of counts as $\tilde{n}(t) = \left(n_1(t), n_2(t), \ldots, n_N(t)\right)$, the child nodes as t_i, and the node configuration vector as $v = (v_1, \ldots, v_N)$, $v_i \in \{0,1\}$. The vector v defines which of the node branches will remain: if $v_i = 0$, then the ith branch will be deleted from the node. Then the principle of sub optimality for any given sub tree $\hat{T}$, starting from the given node t can be represented as follows: the optimal cost $L_{opt}(\hat{T})$ for any given sub tree $\hat{T} \subseteq T$ can be expressed by the following recursive equation:

$$L_{opt}(\hat{T}) = \begin{cases} 0, & \text{if } \hat{T} \text{ is null} \\ \tilde{c}\left(\tilde{n}(t)\right) + \alpha, & \text{if } \hat{T} \text{ hase no childs} \\ \min\limits_{v}\left\{L_v(\hat{T}, v)\right\}, & \text{otherwise,} \end{cases} \tag{6}$$

where

$$L_v(\hat{T}, v) = \tilde{c}\left(\tilde{n}(t) - v \circ \left(\sum_i \tilde{n}(t_i)\right)\right) + \sum_i \left(v_i \cdot L_{opt}\left(\hat{T}_i\right)\right) + \alpha \tag{7}$$

The tree $\hat{T}_i \subset \hat{T}$ is a sub tree of $\hat{T}$, starting from its child node t_i and the constant α is the amount of bits required for describing a single node.

In general, the cost calculation of an optimal context tree T can be described as follows:

Step 1: If T has no child nodes, then return the accumulated code length of its root according to (4).

Step 2: For all sub trees $T_i \subset T$, starting from the child nodes of T root, calculate their optimal costs $L_{opt}(T_i)$.

Step 3: According to the found $L_{opt}(T_i)$, the vectors of counts $\tilde{n}(t)$, and $\tilde{n}(t_1),...,\tilde{n}(t_N)$, find the optimal vector $\tilde{v} = \arg\min\limits_{v} L_v(T, v)$.

Step 4: Prune out the children sub trees according the vector $\tilde{v}$.

Step 5: Return the value $L_v(T, \tilde{v})$.

The algorithm recursively prunes out all unnecessary sub trees, and finally gets the optimal structure of the context tree, see Figure 3.

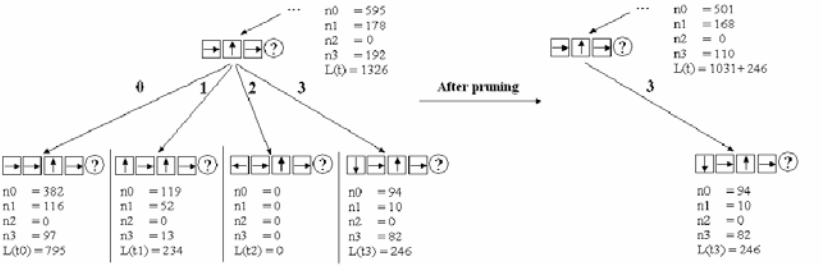


Fig. 3. An example of pruning of the context tree

4 Experiments

We provided two different series of experiments. The first one illustrates the efficiency of the optimal context tree encoding of the chain codes. The second one illustrates the ability of the independent chain encoding to increase the compression

performance of the map image compression in general. Images of Figure 4, which we use, are vector maps, rasterized with the resolution of 5000×5000 pixels. The statistics for all images are shown in Table 1. The first three images are contours of geographical objects, and the last image is a collection of elevation lines. The absolute chain codes were transformed into differential chain codes before compression. As the entropy coder we used range-coder [14].

Tables 1 and 2 show the results for different depth of the context model in the case of 8-connected and 4-connected chain code representations. The numbers in Table 1 are the estimated bit rate according (6). The numbers in Table 2 are the real bit rate, resulted after the range coder. For comparing the compression efficiency, there are results of chain codes compression *PPMd* algorithm with the maximum context order 8 [17]. Higher context order in PPM leads us to the context dilution problem and, consequently, decreasing the compression performance. For the test images the efficient range of the context tree depth is from 4 to 10.

Fig. 4. The set of test images

Table 1. Test image properties and estimated bit rate (bits per symbol)

4-connected chain codes							
	Number of chain codes	Depth					
		4	6	8	10	12	14
Image #1	37888	0.782	0.721	0.711	0.706	0.706	0.706
Image #2	86216	1.031	0.999	0.994	0.992	0.992	0.992
Image #3	208320	1.488	1.482	1.481	1.481	1.481	1.481
Image #4	519316	0.673	0.595	0.568	0.553	0.545	0.542
Average	212935	0.994	0.949	0.939	0.933	0.931	0.930
8-connected chain codes							
	Number of chain codes	Depth					
		4	6	8	10	12	14
Image #1	27306	0.992	0.977	0.970	0.970	0.970	0.970
Image #2	64220	1.386	1.377	1.375	1.375	1.375	1.375
Image #3	160763	2.273	2.272	2.272	2.272	2.272	2.272
Image #4	356839	0.841	0.794	0.782	0.777	0.775	0.775
Average	152282	1.373	1.355	1.350	1.349	1.348	1.348

The second series of experiments were aimed to estimate the efficiency of chain codes compression in comparison to an efficient raster image compression algorithm, namely the JBIG. Table 3 summarizes compressed file sizes, when compressed by the optimal context tree algorithm (CTC 4 and CTC 8), and for corresponding raster

images compressed by JBIG. The raster map images are obtained by rasterization from 4-connected chain codes. The experiments show that the running time of the CTC algorithm is than that of the JBIG. This is because of much smaller amount of encoded information: JBIG encodes 2 500 000 pixels at each image, when CTC encodes only 500 000 chain codes.

Table 4 represents the structure of the compressed file: the percentage of all three types of the data in the file: beginning of the chains (BOC), structure of the context tree (CT), and the encoded chain codes (Chain Codes). The most used contexts in Image#4 for CTC 4 and for CTC 8 compression are shown in Tables 5 and 6 consequently. The most used contexts describe horizontal, vertical or diagonal straight lines. All the experiments were provided on computer P3 500MHz, 256 Mb RAM, Windows NT.

Table 2. Real bit rate (bits per symbol)

4-connected chain codes							
	PPM	CTC 4					
Depth	8	4	6	8	10	12	14
Image #1	0.725	0.801	0.746	0.744	0.742	0.742	0.742
Image #2	1.032	1.044	1.017	1.012	1.010	1.010	1.010
Image #3	1.561	1.500	1.494	1.493	1.493	1.493	1.493
Image #4	0.578	0.678	0.601	0.577	0.565	0.560	0.559
Average	0.974	1.006	0.965	0.957	0.953	0.951	0.951
8-connected chain codes							
	PPM	CTC 8					
Depth	8	4	6	8	10	12	14
Image #1	0.994	1.020	1.022	1.015	1.015	1.015	1.015
Image #2	1.447	1.404	1.397	1.393	1.393	1.393	1.393
Image #3	2.381	2.288	2.289	2.289	2.289	2.289	2.289
Image #4	0.794	0.850	0.808	0.804	0.803	0.803	0.803
Average	1.404	1.391	1.379	1.375	1.375	1.375	1.375

Table 3. Comparison of the CTC-encoded chains and JBIG encoded raster images

	Compressed file size (bytes)			**Compression time (sec)**		
	JBIG	**CTC 4**	**CTC 8**	**JBIG**	**CTC 4**	**CTC 8**
Image #1	5719	3518	3470	99	3	14
Image #2	16027	10887	11189	100	9	50
Image #3	41789	39661	46774	104	25	97
Image #4	71128	39300	38850	100	31	83
Average	33666	23342	25071	101	17	61

Table 4. The proportions of different parts in the compressed file

	BOC	**CT**	**Chain codes**
Image #1	0.2%	1.5%	98.3%
Image #2	0.1%	0.6%	99.3%
Image #3	1.8%	0.3%	97.9%
Image #4	7.9%	1.0%	91.1%

Table 5. Three most used context for Image #4 in CTC 4

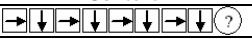
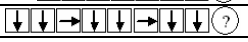
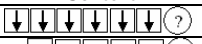
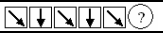
Context	n0	n1	n2	n3	total
	10577	0	2	1943	12522
	4	1355	7924	0	9283
	5904	0	8	1910	7822

Table 6. Three most used context for Image #4 in CTC 8

Context	n0	n1	n2	n3	n4	n5	n6	n7	total
	2	0	1	0	3	560	6473	854	7893
	0	0	6	681	4883	477	8	0	6055
	45	2	2	0	0	0	4879	994	5922

5 Conclusions

We have proposed context tree algorithm for encoding chain codes of contours in map images. The proposed algorithm increased the compression performance over the PPM algorithm by 2-3%. The use of chain codes, instead of the compression of rasterized contours, improves the compression by 25%, on average. The results could be improved up to the theoretical limits by using a more suitable entropy encoder, instead of sub-optimal range coder.

References

[1] Bossen, F., Ebrahimi, T.: Region shape coding, *Technical Report M0318*, ISO/IEC JTC1/SC29/WG11, November 1995

[2] Cleary, J., Witten, I.: Data compression using adaptive coding and partial string matching, *IEEE Trans. on Communications*, 32(4), April 1984, 396-402

[3] Eden, M., Kocher, M.: On performance of a contour coding algorithm in the context of image Coding Part 1: Contour Segment Coding, *Signal Processing,* 1985, 8, 381-386

[4] Estes, R., Algazi, R.: Efficient error free encoding of binary documents, In: *Proc. of IEEE Data Compression Conference*, March 1995, 122-131

[5] Freeman, H.: Computer processing of line drawing images, *ACM Computing Surveys,* 6, March 1974, 57-59

[6] Freeman, H.: Application of the generalized chain coding scheme to map data processing, In: *Proc. of IEEE Pattern Recognition and Image Processing,* May 1978, 220-226

[7] Helfgott, H., Cohn, M.: Linear-time construction of optimal context trees, In: *Proc. of the IEEE Data Compression Conference*, April 1998, 369-377

[8] Howard, P., Vitter, J.: Analyses of arithmetic coding for data compression, In: *Proc. of the IEEE Data Compression Conference,* 1991, 3-12

[9] JBIG: Progressive bi-level image compression, *ISO/IEC International Standard* 11544, 1993

[10] Kaneko, T., Okudara, M.: Encoding of arbitrary curves based on chain code representation, *IEEE Trans. on Communications*, July 1985, 33, 697-707

[11] Liu, Y.K., Zalik, B.: An efficient chain code with Huffman coding, *Pattern Recognition*, 38(4), 2005, 553-557

[12] Lu, C.C., Dunham, G.: Highly efficient coding schemes for contour lines based on chain code representations, *IEEE Trans. on Communications*, 39(10), October 1991, 1511-1514
[13] Martins, B., Forchhammer, S.: Tree coding of bi-level images, *IEEE Trans. on Image Processing,* 7(4), April 1998, 517-528
[14] Martin, G.: An algorithm for removing redundancy from a digitized message, Presented at: *Video and Data Recording Conference*, July 1979
[15] Norhe, R.: Topics in descriptive complexity, *PhD Thesis*, University of Lingköping, *Sweden*, 1994
[16] Rissanen, J.: A universal data compression system, *IEEE Transactions on Information Theory*, 29(5), September 1983, 656-664
[17] Shkarin, D.: PPM: one step to practicality, In: *Proc. of the IEEE Data Compression Conference,* April 2002, 202-211
[18] Weinberger, M., Rissanen J.: A universal finite memory source, *IEEE Trans on Information Theory,* 41(3), May 1995, 643-652
[19] Weinberger, M., Rissanen, J., Arps, R.: Application of universal context modeling to lossless compression of grey-scale images, *IEEE Transactions on Image Processing*, 5, April 1996, 575-586