Statistics-Based Chain Code Compression with Decreased Sensitivity to Shape Artefacts

David Podgorelec, Andrej Nerat and Borut Žalik
University of Maribor, Faculty of Electrical Engineering and Computer Science
Koroška cesta 46, SI-2000, Maribor, Slovenia
E-mail: david.podgorelec@um.si

Keywords: chain code, data compression, dynamic programming, pseudo-statistical model, shape artefact

Received: April 3, 2020

Chain codes compactly represent raster curves, but there is still a lot of room for improvement by means of data compression. Several statistics-based chain code compression techniques assign shorter extra codes to frequent pairs of consecutive symbols. Here we systematically extend this concept to patterns of up to k > 2 symbols. A curve may be represented by any of the exponentially many overlapped chains of codes, and the dynamic programming approach is proposed to determine the optimal chain. We also propose utilization of multiple averaged hard coded pseudo-statistical models, since the exact statistical models of individual curves are often huge, and they can also significantly differ from each other. A competitive compression efficiency is assured in this manner and, as a pleasant side effect, this efficiency is less affected by the curve's shape, rasterization algorithm, noise, and image resolution, than in other contemporary methods, which surprisingly do not pay any attention to this problem at all.

Povzetek: V članku predstavimo novo metodo za statistično stiskanje verižnih kod, ki dodeli posebne kode pogostim nizom do k simbolov. Optimalno izmed eksponentno mnogo rešitev izbere z dinamičnim programiranjem. Uporablja več povprečenih psevdo-statističnih modelov, ki jih ne shranjuje skupaj s krivuljo. V primerjavi z drugimi sodobnimi metodami doseže konkurenčno stopnjo stiskanja, hkrati pa je manj občutljiva na obliko krivulje, posebnosti rasterizacijskega algoritma, šum in ločljivost slike.

1 Introduction

Chain codes compactly represent curves in raster images. More than half a century ago, Freeman [3] used symbols $\sigma_i \in [0 .. 7]$ to represent each curve pixel p_i with the azimuth direction ($\sigma_i \cdot 45°$) from its predecessor p_{i-1} measured anticlockwise from the positive *x*-axis (Fig. 1a). Each symbol is then coded with 3 bits. Alternatively, only 2 bits per pixel are required if the representation relies on 4-connectivity, i.e. the azimuth $p_i - p_{i-1}$ is ($\sigma_i \cdot 90°$), where $\sigma_i \in [0 .. 3]$ (Fig. 1b). Several alternative chain code representations were later introduced, but the concept remains the same as in the pioneering *Freeman chain codes in eight (F8) or four (F4) directions*: symbols from a relatively small alphabet are assigned to subsequent primitives along a curve. In different representations, a primitive may refer to a curve pixel (as in F8 or F4), a vertex between the considered curve pixel and adjacent pixels (Vertex Chain Code – VCC [2] or Three-Orthogonal chain code – 3OT [10]), an edge separating the curve pixel from a background pixel (Differential Chain Code – DCC [9]), or a rectangular cell of pixels (in quasi-lossless representation from [9]). Meanwhile, a symbol models some local geometric relation e.g. relative position of the observed primitive with respect to the previous one. With other words, it represents a command how to navigate from one primitive to the adjacent one along the curve. All these basic chain code representations describe a raster curve efficiently, as they use only 2 or 3 bits per primitive instead of coding grid coordinates with, for example, $2 \cdot 16$ bits per pixel. Nevertheless, numerous successful methods have been proposed to additionally compress raster curves.

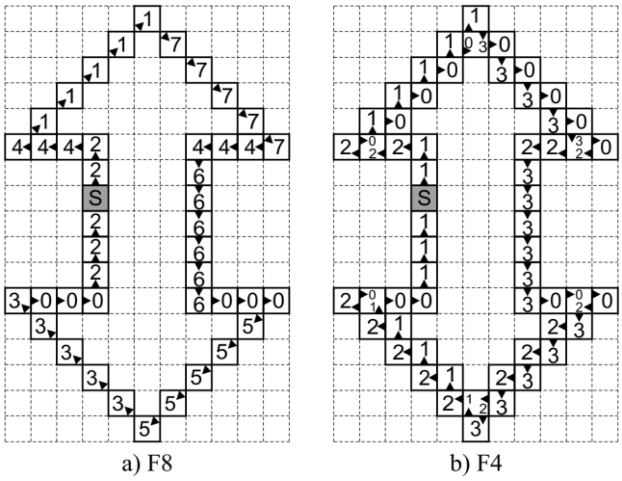

Figure 1: Freeman chain codes in 8 and 4 directions.

Statistical (Huffman or arithmetic) coding is often utilized when the symbols' probability distribution is significantly non-uniform. Further advances in statistics-

based approaches were achieved by introducing extra codes for frequent pairs of primitives [7, 8], or by utilization of multiple statistical models in so-called context-based approaches [1], where the statistical model for coding a considered symbol is conditioned on the context of M (typically 1 or 2) previous symbols. On the other hand, non-statistical approaches perform various string transformations [11, 12], e.g. Burrows-Wheeler Transform (BWT) and/or Move-To-Front Transform (MTFT) to increase the number of 0's and prepare the data for efficient run-length encoding (RLE) and/or binary arithmetic coding (BAC). BWT rearranges the sequence into runs of equal symbols, while MTFT utilizes local correlations to replace the data values with their indices from typically smaller repertoire.

In this paper, we introduce a new statistics-based approach where extra codes may represent patterns of up to $k = O(1)$ symbols. Our aim was to achieve a competitive compression efficiency, but an interesting pleasant side effect was encountered and brought into focus during the method development and testing. Namely, impacts of the curve shape, image resolution, rasterization algorithm, noise, and geometric transformations on the compression ratio are significantly reduced in comparison to other contemporary methods. This topic has been so far addressed indirectly within the context-based approaches and, partially, in the non-statistical approaches, while it was completely neglected in other related works. Section 2 illustrates the overall idea of the proposed approach. Section 3 describes the preparation and utilization of multiple averaged hard coded pseudo-statistical models, crucial for the minimization of the mentioned impacts. Section 4 experimentally confirms the compression efficiency and the reduced dependence on curve's artefacts. Finally, Section 5 summarizes the presented work, and discusses future research challenges.

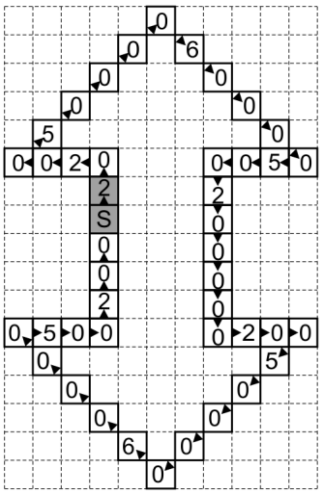

Figure 2: Freeman's chain difference coding.

2 New statistics-based method

Some years after F8 and F4, Freeman also proposed the chain-difference coding (CDC) [6]. A pixel p_i is encoded with the angle difference $\angle(p_i - p_{i-1}, p_{i-1} - p_{i-2})$. Unlike F8 where all 8 symbols have practically the same probabilities, the 0° angle difference is usually much more frequent than other 7 symbols (Fig. 2). All 8 symbols in Fig. 1a have probabilities either 6/43 ≈ 13.95% or 7/43 ≈ 16.28%, and the probabilities of four symbols in Fig. 1b are either 16/63 ≈ 25.40% or 15/63 ≈ 23.81%. On the other hand, the probability of symbol 0 in Fig. 2 is 32/42 ≈ 76.19% while the probabilities of other 7 symbols are all below 10%. Such non-uniform distribution provides a good basis for statistical coding. However, some tens of bits must be spent to store the best-fitted statistical model (BFSM) for an individual curve, which is, particularly with shorter curves, not negligible. Liu and Žalik [6] presented the directional difference chain coding (DDCC), where CDC BFSMs of over 1000 training curves are averaged into a suboptimal hard-coded statistical model (HCSM), which is then used for compression in non-training use cases. Some years later, the compressed DDCC (C_DDCC) [7] was introduced, where three extra codes for usually frequent pairs $\langle \pm 45°, \mp 45° \rangle$ and for patterns of 12 to 27 zeros were added into the HCSM. The two pairs were intuitively identified, as they periodically interrupt sequences of 0° symbols along oblique line segments (except those with slopes 45° or -45°).

Here we take a step forward by systematically extending the DDCC coding scheme with extra codes for patterns of up to $k = O(1)$ symbols, $k \geq 2$. Furthermore, we group "similar" training curves into classes and derive HCSMs separately for each class. Although this concept looks straightforward, several non-trivial issues must be considered to achieve a feasible implementation and a competitive compression efficiency. These issues can be structured into two separate phases.

1. The training phase is performed by an expert/developer in order to calibrate the algorithm for future use. A representative repertoire of training curves is first provided, and the BFSM for each curve is extracted. Features for multicriteria classification of training curves are selected (intuitively in the current implementation), and the training curves are then assigned to the classes. HCSMs are derived afterwards by separately averaging BFSMs within the classes. The detailed description follows in Section 3.
2. The exploitation phase is run by end-users in order to compress non-training curves from concrete use cases. An input curve is first analysed to determine the feature values needed to heuristically select the most appropriate of the stored HCSMs. The chosen HCSM is then utilized to compress the curve with Huffman coding. The main challenge in designing this phase is the strategy for determination of the optimal sequence (chain) of codes, which is emphasized after the definitions.

2.1 Definitions

Trail $T_{i,j} = \langle v_i, ..., v_j \rangle$, $i \le j$, is a sequence of adjacent pixels (or corresponding graph vertices) along a raster curve. The length of the trail (in pixels) is $h(i, j) = j - i + 1$. The trail $T_{1,n} = \langle v_1, ..., v_n \rangle$ corresponds to the entire raster curve of length n.

Trail decomposition splits the trail into one or more nested trails, whose concatenation reassembles the original trail. A trail $T_{u,v}$ is *nested* in $T_{i,j}$ if $i \le u \le v \le j$.

Symbol is a chain-code command aimed to be assigned to a single pixel along a raster curve.

Pattern (of symbols) $\Sigma_{i,j} = \langle \sigma_i, ..., \sigma_j \rangle$, $i \le j$, is a sequence of symbols aimed to be assigned to pixels of a trail of the same length $h(i, j)$.

Dynamic programming graph is an edge-weighted graph (G, w), where $G = (V, E)$ is a directed graph, $V = \{v_1, ..., v_{n+1}\}$ is a vertex set, $E = \{e_{i,j}\}$ is an edge set, given by pairs of vertices $e_{i,j} = (v_i, v_j)$, $i < j$, and $w : E \to \mathbb{N}$ is a weight function. Vertices $v_1, ..., v_n$ correspond to pixels along the raster curve, edge $e_{i,j}$ represents a trail $\langle v_i, ..., v_{j-1} \rangle$, and weight $w_{i,j}$ of an edge $e_{i,j}$ is the bit length of the corresponding Huffman code.

An auxiliary end vertex v_{n+1} does not represent any curve pixel and, thus, there is no need to assign a symbol to it. However, this vertex enables introduction of edges $e_{i,n+1}$, $i \le n$, corresponding to trails $T_{i,n} = \langle v_i, ..., v_n \rangle$.

***IN*(*i*)** is the set of start vertices of all graph edges with the end vertex v_i. $v_j \in IN(i) \Leftrightarrow e_{j,i} \in E$. Vertex v_j is a predecessor of v_j and the latter is a successor of v_j.

***OUT*(*i*)** is the set of end vertices of all graph edges with the start vertex v_i. $v_j \in OUT(i) \Leftrightarrow e_{i,j} \in E$.

Extra code is a Huffman code which replaces a pattern of two or more symbols in order to save some bits.

$\boldsymbol{p(\Sigma_{i,j})}$ is the probability of a pattern $\Sigma_{i,j} = \langle \sigma_i, ..., \sigma_j \rangle$ in a considered statistical model. If the latter corresponds to the BFSM of a curve described with $T_{1,n} = \langle v_1, ..., v_n \rangle$, then we get equation (1):

$$p(\Sigma_{i,j}) = f(\Sigma_{i,j}) / (n - h(i, j) + 1), \tag{1}$$

where $f(\Sigma_{i,j})$ be the number of appearances of $\Sigma_{i,j}$ in the pattern assigned to $\langle v_1, ..., v_n \rangle$. However, $p(\Sigma_{i,j})$ in some HCSM is obtained by averaging the corresponding probabilities from all participating BFSMs.

Note that an edge $e_{i,j}$, $i < j - 1$, is added into the graph only if an extra code exists for the corresponding pattern assigned to $T_{i,j-1}$. On the other hand, edges $e_{i,i+1} = (v_i, v_{i+1})$ correspond to single-pixel trails $T_{i,i} = \langle v_i \rangle$ and they are unconditionally added to the graph. This assures that the algorithm of parsing the curve pixels will always reach the end vertex v_{n+1}, as any other vertex has at least one successor, i.e. $i \le n \Rightarrow |OUT(i)| \ge 1$.

2.2 Exploitation phase

The existing chain code techniques construct the chain of codes by a greedy algorithm. A raster curve is parsed primitive by primitive, and each of them is immediately coded either alone or as a member of some longer pattern. If different possibilities for coding a primitive exist, the predefined priority is decisive. In C_DDCC, for example, extra codes for $\langle \pm 45°, \mp 45° \rangle$ pairs have higher priority than the corresponding single-pixel codes. However, such priority-based greedy algorithms cannot be simply adjusted to efficiently handle higher number of extra codes for longer patterns of symbols. In the proposed approach, each pixel can be coded with its own code or, theoretically, with one of k codes of longer trails. For $k = 6$ as used in the current implementation and tests (the decision for this value will be explained at the end of Section 3.2), these trails include two pairs, three triplets and so on till six sextets. A longer context of patterns before and behind the considered symbol determines which of the $1 + 2 + \ldots + k = k\,(k + 1) / 2$ possibilities (21 for $k = 6$) shall be used to code the pixel. We therefore have a combinatorial optimization problem where we look for an optimal chain from a large set of multiply overlapped chains. Unlike greedy algorithms, we found dynamic programming capable to provide an optimal choice. Its utilization also facilitates the so-called context dilution problem [1, 7]. Namely, introduction of extra codes for longer patterns of symbols usually extends codes of several symbols and other patterns. For example, introduction of four extra C_DDCC codes for patterns $\langle 0°, \pm 45° \rangle$ and $\langle \pm 45°, 0 \rangle$ prolongs by 1 bit the codes for ±90°, 180°, RLE of zeros, 135° and/or –135°. Furthermore, examples of chains can be found where individual extra codes do not save any bits.

The proposed dynamic programming approach is adaptation of the so-called exon chaining algorithm from the field of bioinformatics, the simplest of the so-called similarity-based gene prediction approaches [5].

The dynamic programming optimizes the Bellman equation (2), where s_i represents the total bit length of the optimal chain from v_1 to v_{i-1}, $1 < i \le n + 1$. Additionally, s_0 is set to 0 to enable the recursive calculation of s_1.

$$s_i = min_{v_j \in IN(i)}(s_j + w_{j,i}) \tag{2}$$

The vertex $pred_i \in IN(i)$, which indeed participates to the minimum s_i, is also memorized for each v_i. The s_{n+1} represents the total bit length of the overall solution, and the optimal chain itself is then reconstructed by following the vertices $pred_i$ from v_{n+1} backwards to v_1. Bold edges in Fig. 3 represent the optimal chain for the given example. Trails $T_{1,2}$, $T_{3,5}$, and $T_{6,9}$ are coded with $4 + 6 + 8 = 18$ bits. Note that an equivalent solution with $s_n = 18$ exists, where the first trail terminates with v_3, as demonstrated with a pair of dashed edges in Fig. 3.

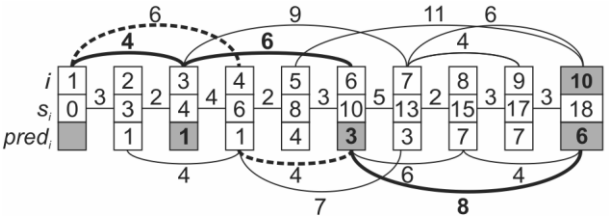

Figure 3: Dynamic programming graph.

The remarkable performance of the dynamic programming-based optimization is highlighted with Theorems 1 and 2. Although the growth of the number of solution candidates is exponential in curve length, the algorithm runs in linear time.

Theorem 1. *The number of possible decompositions of a trail grows exponentially with the trail length (in pixels) if extra codes for patterns of up to k > 1 symbols are used.*

Proof. Let $c_i(k)$ be the number of possible decompositions of $T_{1,\,i}$, where the lengths of nested trails obtained by the decomposition do not exceed k. Each of these decompositions ends with the trail $T_{i-l+1,\,i}$ of l vertices, $1 \leq l \leq min(i, k)$, preceded with one of $c_{i-l}(k)$ possible decompositions of $T_{1,\,i-l}$. Note that for $i < k$, there are less than k symbols available and, thus, the upper bound for the length of the ending trail is $min(i, k)$.

The ending trail $T_{i-l+1,\,i}$ can span through the entire $T_{1,\,i}$ (when $i = l$), resulting in an empty preceding $T_{1,\,0}$. Unlike the definition of trail in Section 2.1, we exceptionally allow $i > j$ here. This situation is indicated by $c_0(k) = 1$.

Equation (3) defines the calculation of $c_i(k)$, $i > 0$.

$$c_i(k) = \sum_{l=1}^{min(i,\,k)} c_{i-l}(k)\,,\, i > 0 \tag{3}$$

Obviously, $c_1(k) = c_0(k) = 1$ as the first pixel of $T_{1,\,i}$ can be preceded by an empty trail only in a single way. Similarly, $c_i(1) = 1$ since $T_{1,\,i}$ can be decomposed into single-pixel trails only in a single way. For $k = 2$, $i > 1$, equation (4) is obtained.

$$c_i(2) = c_{i-1}(2) + c_{i-2}(2),\, i > 1 \tag{4}$$

Let F_i represent the i-th Fibonacci number. The Fibonacci sequence is defined by $F_0 = 0$, $F_1 = 1$, and $F_i = F_{i-2} + F_{i-1}$ for $i > 1$. This recursive formula gives $F_2 = 1$ and we may thus match $c_0(k) = F_1$ and $c_1(k) = F_2$. The equation (4) then gives: $c_2(2) = F_2 + F_1 = F_3$, $c_3(2) = F_3 + F_2 = F_4$, and $c_i(2) = F_i + F_{i-1} = F_{i+1}$. As $F_{i+1} > F_i$, $i > 1$, we thus get the inequality (5).

$$c_i(2) > F_i,\, i > 1 \tag{5}$$

This result can be generalized to $k > 2$ by using the relation (6), which must be proved beforehand.

$$c_i(k + m) > c_i(k),\, m \geq 1,\, i > 1 \tag{6}$$

The proof is actually trivial. Due to the transitivity of "Is greater than", it suffices to consider $m = 1$. The key observation is that all the decompositions counted by $c_i(k)$ are also counted by $c_i(k + 1)$ which, however, additionally counts the decompositions with at least one nested trail of length $k + 1$. The relation (5) may thus be generalized to the relation (7).

$$c_i(k) > F_i,\, k > 1 \wedge i > 1 \tag{7}$$

As the Fibonacci sequence F_i has the proven exponential growth, we may confirm that the sequence $c_i(k)$ also grows (at least) exponentially for $k > 1$. Theorem 1 is thus proved. □

Note that we assumed in the theorem, that all the trails of up to k pixels are represented by edges of the dynamic programming graph, but this is usually not a case due to the statistical model reduction (Section 3.2). The proved exponential growth therefore represents only the theoretical worst case. However, as BFSMs and particularly HCSMs typically contain the majority of the patterns of length 2 ($k = 2$ suffices for the exponential growth) and also quite a few longer patterns, the expected growth may also be considered exponential.

An interesting finding is that the recursion in equation (3) can be solved easily for $k \geq i$. Namely, substitution $c_{i-2}(k) + \ldots + c_0(k) = c_{i-1}(k)$ transforms $c_i(k) = c_{i-1}(k) + \ldots + c_0(k)$ into $c_i(k) = 2\, c_{i-1}(k)$. We may then recursively progress with such substitutions, i.e. $2\, c_{i-1}(k) = 2^2\, c_{i-2}(k) = \ldots = 2^{i-1}\, c_1(k)$, towards the equation (8).

$$c_i(k) = 2^{i-1},\, k \geq i > 0 \tag{8}$$

The result in equation (8) is expected, as any decomposition is obtained by breaking apart the trail in some interruption points between successive pixels. Since $k \geq i$, there are no limitations in the length of nested trails obtained by the decomposition. All the combinations from i single-pixel trails to a single trail spanning through entire $T_{1,\,i}$ are valid. There are $i - 1$ interruption points in a trail with i symbols and thus 2^{i-1} possible decompositions.

Theorem 2. *Optimal chain detection, based on the dynamic programming and utilization of extra codes for patterns of up to k = O(1) symbols, runs in $\Theta(n)$ time, where n is the curve length in pixels.*

Proof. The cardinality $|IN(i)|$, $1 < i \leq n$, cannot exceed k, as each v_i may only represent the end of a trail (edge) of length between 1 and k. The upper bound for time complexity of calculating s_i, $1 < i \leq n$, is thus $O(k\, n) = O(n)$ time if $k = O(1)$. The lower bound however is achieved if the statistical model contains only single-pixel symbols. But even in this case, the linear time is needed to parse the dynamic programming graph. The $\Theta(n)$ time complexity is thus proved. □

3 Training phase

Linear performance proved in Theorem 2 is not the only reason for limiting the length of patterns with attached extra codes to $k = O(1)$ symbols. This also reduces the size of the statistical model, which has a mitigating effect on the context dilution problem. In the proposed study, $k = 6$ have been chosen among different considered values. The reasons for this decision shall be explained in Section 3.2. Even in this way, the statistical model derived from the basic DDCC scheme can theoretically contain $8 + 8^2 + \ldots + 8^6 = 299{,}592$ entries. Although many of these patterns never appear in practice, and even

if we manage to further reduce the size of the statistical model (to some tens entries in practice), there is the only practical possibility to use an averaged statistical model (or more of them). Its derivation requires a careful consideration of the following important issues.

3.1 Training set

In the reported C_DDCC tests [7], relative compression ratio to F8 only slightly varies (between 0.46 and 0.55). This may lead to a conclusion that the derived HCSM serves well for all use cases. Furthermore, similar conclusions can be adopted for practically all existing methods, no matter whether they belong to statistics-based or non-statistical approaches, and whether, in the first case, they use a HCSM or BFSMs. However, we must be aware that the training sets and testing use cases in presentations of these methods usually follow some curve creation and rasterization methodology and, thus, they share some evident common artefacts. In C_DDCC tests, for example, there were a huge probability of shorter sequences of 0° symbols, relatively high probabilities of ⟨±45°, ∓45°⟩ pairs, and rather low probabilities of ±90° symbols. In our method, we may expect even bigger impact of the curve's shape on the compression efficiency, as the distributions of longer patterns from a bigger repertoire can vary considerably from curve to curve. An averaged statistical model can thus deviate significantly from both, the BFSMs of individual training curves used to construct it in the training phase, and the distributions of patterns used to code testing curves in the exploitation phase. Since the latter directly affects the compression efficiency, we decided to use multiple averaged statistical models and, consequently, to classify the training curves and testing use cases regarding some chosen measurable artefacts. In this manner, the method gains generality, as the compression efficiency becomes less dependent on the curve creation and rasterization methodology.

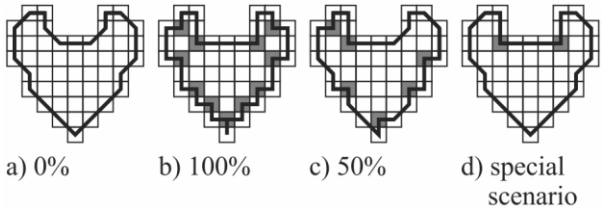

Figure 4: Different levels of forcing the 4-connectivity.

To provide an adequate training set and a relevant mixture of testing use cases, we have implemented a tool with functionalities of image rotation and scaling, manual inversions of binary values of selected pixels, and extraction of the boundary chain of a presented binary object. In this last operation, the parameter *Force-4-connectivity* controls the amount of ±90° symbols along oblique edges and, thus, simulates different rasterization methodologies. Value 0% (Fig. 4a) means that the boundary chain consists only of pixels which share edges with the object's exterior. On the other hand, value 100% (Fig. 4b) adds into the chain all the pixels which are vertex-connected with the object's exterior. Such pixel is 4-connected with both adjacent chain pixels. In Fig. 4c, half of possible pixels of this kind (coloured grey) are randomly chosen and inserted into the chain. Finally, a special scenario is supported (Fig. 4d) where a 4-connected pixel is only inserted if it represents a concave vertex between a horizontal and vertical edge (each at least two pixels long).

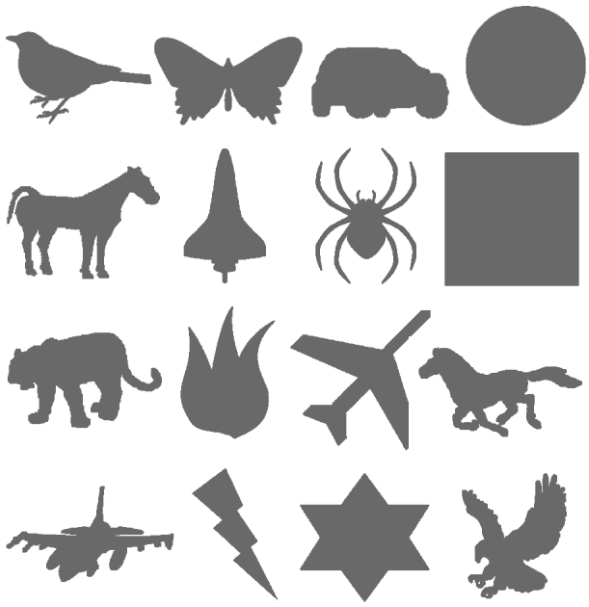

Figure 5: Examples of training and testing objects.

Basic shapes from the training set and use cases are shown in Fig. 5. They were mostly inherited from the tests made in [7, 11, 12]. Objects from the first two rows were used for testing (see Table 1), while the others belong to the training set. All together we used 50 basic shapes, i.e., 30 in the training set and 20 test cases. A variety of instances of these shapes in different orientations and scales, ranging between 150 and 20,000 boundary curve pixels, and with different levels of forcing the 4-connectivity were utilized in the experiments. There are 500 shapes in the training set.

3.2 Statistical model reduction

The first step towards reducing huge amount of data in each BFSM and mitigation of the context dilution effect was already made by limiting k to $O(1)$ symbols. We also do not have to consider patterns with probability 0. Furthermore, we may set even stronger conditions for the probability $p(\Sigma_{i,j})$ of a pattern to be accepted in a BFSM. Namely, a pattern $\Sigma_{i,j} = \langle \sigma_i, ..., \sigma_j \rangle$ is inserted in the statistical model only if $p(\Sigma_{i,j})$ is higher than the product of probabilities (weighted with w_2) of any sequence of shorter patterns whose concatenation forms $\Sigma_{i,j}$. To prevent insertion of too low probabilities, we use additional threshold w_1. The following statement considers patterns of length $l = 3$.

if ($p(\Sigma_{1,3}) > max(w_1, w_2 * max(p(\Sigma_{1,1})p(\Sigma_{2,2})p(\Sigma_{3,3}), p(\Sigma_{1,1})p(\Sigma_{2,3}), p(\Sigma_{1,2})p(\Sigma_{3,3}))))$
then insert $((\Sigma_{1,3}, w_3 * 3 * p(\Sigma_{1,3}))$ into BFSM.

As the patterns of lengths 2, 4, 5 and 6 must also be considered, as well as eventual future extensions, we generate all the concatenations algorithmically. A concatenation is obtained by breaking apart the pattern in some interruption points between successive symbols. There are $l - 1$ possible interruption points in a pattern of l symbols and thus $2^{l-1} - 1$ possible concatenations. Here the subtracted 1 represents the non-interrupted pattern. For patterns of lengths 2 to 6, we thus must test 1 + 3 + 7 + 15 + 31 = 57 products. Obviously, the method must first evaluate shorter patterns, as their probabilities are used in acceptance criteria for longer ones. As we mentioned at the end of Section 2.1, single-pixel symbols are unconditionally included in a BFSM.

Note that the weights w_1, w_2, and w_3 offer a lot of possibilities for experimentation. They were also crucial for decision to use patterns of up to $k = 6$ symbols in our tests. As the probabilities are usually decreasing with the pattern length (with possible exceptions), the value of w_1 must be decreased if $k = 7$ is used instead of $k = 6$. However, this causes that additional shorter patterns of lengths 6, 5, 4 etc. are also accepted into a BFSM, increasing the size of the BFSM and emphasizing the context dilution effect in a negative way. On the other hand, this problem appears less evident when comparing $k = 5$ and $k = 6$. Although we have not performed a complete sensitivity analysis yet, the decision for $k = 6$ seems a reasonably good choice confirmed by the results in Section 4.

3.3 Statistical vs. pseudo-statistical model

We do not wish (and neither we are able) to split probabilities of symbols and patterns among some longer patterns, as this would lead to the priority-based greedy approach, which we intentionally try to avoid. Each symbol consequently participates to probabilities of all the patterns, which include it. Strictly speaking, we use weighted probabilities (multiplied with $w_3 * l$) to reward longer patterns by assigning shorter codes to them. The sum of such weighted probabilities in a model may be as high as $(1 + 2 + 3 + 4 + 5 + 6) * w_3 = 21w_3$. It is however lower because the patterns are added selectively, but it still exceeds 1. We apparently do not deal with true statistical models but with pseudo-statistical models instead. We shall use the acronyms BFPSM and HCPSM instead of BFSM and HCSM from this point on. Nevertheless, all weighted "pseudo-probabilities" are involved in a single Huffman tree construction.

3.4 Averaging pseudo-statistical models

Averaging is a two-stage process. During the extraction and reduction of the BFPSM of a considered training curve, several simply assessed curve artefacts are computed. These are then utilized for multicriteria classification, which assigns the curve into one of the pre-defined classes. From all the assessed features that will be used in future to algorithmically select optimal classification criteria, we currently use three intuitively chosen criteria listed below, each with a single threshold.

- Average turn per pixel. Each ±45° symbol participates 1 to this value, ±90° symbols 2, ±135° symbols 3, and 180° symbols 4. The sum is then divided with the curve length in pixels. This feature separates smooth curves from more winding and noisy ones. It is negatively correlated with the probabilities of 0° symbols and their longer runs.
- Probability of ⟨±45°, ∓45°⟩ pairs is higher in curves with oblique segments than in those with mostly axis-aligned and/or ideally diagonal segments.
- Probability of ±90° symbols is usually higher in images of man-made objects than in natural objects.

Three single-threshold criteria result in 8 classes with binary indices from 000 to 111, where the first bit represents the first criterion, and the third bit refers to the last criterion. 0's signify values below the thresholds, and 1's those above the thresholds. In the current setting, the thresholds were computed by averaging the described quantities over the BFPSMs of all training curves.

It turns out that the classes with indices $010_{(2)}$ and $101_{(2)}$ are nearly twice more populated than others. In our training set with 500 shapes, there are 112 shapes in the class $101_{(2)}$ and 99 shapes in the class $010_{(2)}$, while the remaining six classes contain between 37 and 55 shapes. The testing use cases are also distributed in a similar way. This deviation can be explained by suboptimal training set, suboptimal thresholds selection and suboptimal classification criteria, which are all among the most important challenges for our future work. However, we may immediately establish that the currently used criteria are all correlated with the *Force-4-connectivity* value. Firstly, all additional 4-connected pixels are coded with ±90° symbols and thus increase the third criterion value. Secondly, such pixels are often inserted in the middle of ⟨±45°, ∓45°⟩ pairs, changing them into ⟨±90°, ∓90°, 0°⟩ triplets. Finally, a pair ⟨±45°, ∓45°⟩ participates 2 to the first criterion (1 per pixel), while a ⟨±90°, ∓90°, 0°⟩ triplet participates 4 (1.33 per pixel). The first and the last criterion are thus positively correlated, and there is a negative correlation between them and the second one. The indices $010_{(2)}$ and $101_{(2)}$ of above-average populated classes also confirm this finding, as the second bit is in both cases the inverse of the other two.

In the second stage, after the training curves are classified (into 8 classes in the current setting), HCSMs are derived by separately averaging BFSMs within each class. However, the BFSMs in a particular class may still significantly differ from each other, although expectedly (and confirmed by the testing results) not as much as the BFSMs from different classes. Consequently, the HCSMs must also be reduced by using the same acceptance criteria as in the BFPSM reduction (Section 3.2).

As we are aware, that the current classification is not optimal, we try to mitigate impacts of wrongly classified training curves by using soft borders between the classes. This means that averaging in an observed class also considers weighted probabilities from BFPSMs of all "adjacent" classes, distinct in one criterion from the considered one. For example, classes 001, 010, 100 are adjacent to the class 000, while, e.g., 011 is not. In the tests presented in Section 4, the probabilities are weighted in a manner that BFPSMs from an observed class contribute two thirds to the corresponding HCPSM, and those from the three adjacent classes contribute a third (a ninth each).

4 Results

In this section, we compare some typical results of the proposed method and some state-of-the-art (SOTA) chain code compression methods. 3OT, VCC, C_DDCC, and three variants of MTFT+ARLE (Move-To-Front Transform + Adaptive Run-Length Encoding) [11], i.e., MTFT+ARLE VCC, MTFT+ARLE 3OT, and MTFT+ARLE NAD (four-symbol Normalised Angle-Difference chain code) [11] were used in the tests.

The training set and use cases from Section 3.1 were used, and the weights w_1, w_2 and w_3 for the pseudo-statistical models reduction (see Section 3.2) were set to 0.02, 1.0 and 1.0, respectively. As we already stressed and explained, the length of patterns to be considered is limited to k = 6. The classification thresholds (Section 3.4) computed for the utilized training set were initialized to 0.92 for the average turn, 0.12 for $p(\langle \pm 45°, \mp 45° \rangle)$, and 0.295 for p(±90°).

Object	Transform	Pixels	bpp (SOTA)	bpp (new method)
Basic ("user friendly") shapes				
Bird	100, 0, 0	4080	1.11[(1)]	1.03
Butterfly	100, 0, 0	1122	1.45[(1)]	1.33
Car	100, 0, 0	541	1.48[(1)]	1.25
Circle	100, 0, 0	1831	1.13[(2)]	0.99
Horse	100, 0, 0	2143	1.51[(3)]	1.39
Shuttle	100, 0, 0	969	1.19[(1)]	1.08
Spider	100, 0, 0	1770	1.20[(2)]	1.04
Square	100, 0, 0	1088	0.30[(2)]	0.35
Sophisticated instances				
Bird	10, 50, 70	671	1.60[(3)]	1.31
Butterfly	140, 45, 100	2681	1.68[(3)]	1.21
Car	200, 33, 50	1472	1.84[(3)]	1.49
Circle	20, 0, 0	308	1.39[(2)]	1.06
Horse	50, 15, 20	1284	1.93[(1)]	1.51
Shuttle	100, 30, 0	980	1.31[(1)]	0.94
Spider	120, 45, 25	2218	1.31[(1)]	1.08
Square	100, 70, 30	1228	0.75[(3)]	0.62

Table 1: Test cases and compression results [bpp]. The listed best SOTA results were obtained by C_DDCC[(1)], MTFT+ARLE NAD[(2)], or MTFT+ARLE VCC[(3)].

Table 1 shows the results for pairs of different instances of eight objects from the top two rows in Fig 5. Basic "user friendly" shapes refer to smooth, noiseless instances as being usually employed in testing the state-of-the-art (SOTA) chain code compression methods. The "sophisticated" instances were generated by transforming the basic ones with the scaling factor, rotation angle, and/or amount of additional 4-connectivity pixels different from 100%, 0°, 0%, respectively (see column *Transform*). The column *bpp (SOTA)* shows efficiency in bits per pixel (bpp) of the best of the compared SOTA methods. Comparison of the last two columns reveals that the new method is superior in most cases. The only exception is the basic axis-aligned square where all three MTFT-ARLE variants and also C_DDCC substantially benefit from long runs of 0's.

Ratios between the efficiencies of the new and best SOTA method are given in columns *A* and *B* of Table 2, separately for the basic and transformed instances. The new algorithm is mostly for 10 to 15% more efficient than SOTA in the basic configurations, and for additional 10% in the sophisticated cases. Columns *C* and *D* show ratios between the efficiencies for sophisticated and adequate basic configurations. SOTA is considered in column *C*, and the new method in column *D*. The results confirm that sophisticated curve artefacts much more affect SOTA methods (average ratio 1.22 means lower efficiency for 22%, compared to basic shapes) than the new method (average ratio 1.06). The latter even achieves better compression of some transformed shapes (butterfly and shuttle) in comparison to the basic ones. It also surpasses SOTA in the transformed square example, where all the considered methods achieve significantly worse results (omitted in the above average ratios) than in the axis-aligned instance.

Object	A	B	C	D
Bird	0.93	0.82	1.44	1.27
Butterfly	0.92	0.72	1.16	0.91
Car	0.84	0.81	1.24	1.19
Circle	0.88	0.76	1.23	1.07
Horse	0.92	0.78	1.28	1.08
Shuttle	0.91	0.72	1.10	0.87
Spider	0.87	0.82	1.09	1.04
Square	1.17	0.83	2.50	1.77

Table 2: Analysis of the compression results.

5 Conclusions

In this paper, we introduce a new statistics-based chain code compression methodology by using multiple averaged pseudo-statistical models correlated with some measurable curve artefacts, and by heuristically selecting the most appropriate of these models prior to the compression. Furthermore, the introduced models contain extra codes for systematically selected patterns of up to k symbols (k = 6 in the presented tests), and the dynamic programming approach replaces the common greedy method in order to determine the optimal chain of patterns. The early results are promising, but there is a

plenty of work left to ultimately affirm the proposed methodology.

The methodology incorporates the training phase and the exploitation phase. The former obviously associates this research with machine learning, but classification of the training curves with respect to three intuitively pre-selected and even mutually correlated criteria is quite far from this paradigm. However, one of our future goals is to adapt the introduced methodology to other basic chain code representations (VCC, 3OT, F4, F8, and NAD), which shall certainly require more advanced and adjustable feature extraction, learning and selection, leading into optimized classification algorithms. This goal also requires an extensive sensitivity analysis by varying the number and values of classification thresholds, weights in the pattern acceptance criteria, etc. Other future goals include:

- comparison to modern non-statistical methods on both, "standard" and less "user-friendly" cases,
- improving the training set and preparation of rich repertoire of benchmarks,
- utilization of arithmetic coding instead of Huffman codes, and
- inclusion of RLE codes for longer patterns of 0's.

6 Acknowledgements

The authors acknowledge the financial support from the Slovenian Research Agency (Research Core Funding No. P2-0041). We are also grateful to our former colleague Denis Špelič for his contribution to the implementation.

7 References

[1] Akimov A.; Kolesnikov A.; Fränti P. (2007). Lossless compression of map contours by context tree modeling of chain codes, *Pattern Recognition*, Elsevier Science, vol. 40, iss. 3, pp. 944-952. https://doi.org/10.1007/11499145_33

[2] Bribiesca E. (1999). A new chain code. *Pattern Recognition*, Elsevier Sci., vol. 32, iss. 2, pp. 235-251. https://doi.org/10.1016/s0031-3203(98)00132-0

[3] Freeman H. (1961). On the encoding of arbitrary geometric configurations. *IRE Transactions on Electronic Computers*, IEEE, vol. EC10, iss. 2, pp. 260-268. https://doi.org/10.1109/tec.1961.5219197

[4] Freeman H. (1974). Computer processing of line drawing images. *ACM Computing Surveys*, ACM, vol. 6, iss. 1, pp. 57-97. https://doi.org/10.1145/356625.356627

[5] Jones N. C.; Pevzner P. A. (2004). *An Introduction to Bioinformatics Algorithms*. The MIT Press.

[6] Liu Y. K.; Žalik B. (2005). An efficient chain code with Huffman coding. *Pattern Recognition*, Elsevier Science, vol. 38, iss. 4, pp. 553-557. https://doi.org/10.1016/j.patcog.2004.08.017

[7] Liu Y. K.; Žalik B.; Wang P.-J.; Podgorelec D. (2012). Directional difference chain codes with quasi-lossless compression and run-length encoding. *Signal Processing: Image Commun*., Elsevier Science, vol. 27, iss. 9, pp. 973-984. https://doi.org/10.1016/j.image.2012.07.008

[8] Liu Y. K.; Wei W.; Wang P.-J.; Žalik B. (2007). Compressed vertex chain codes. *Pattern Recogn*., Elsevier Science, vol. 40, iss. 11, pp. 2908-2913. https://doi.org/10.1016/j.patcog.2007.03.001

[9] Nunes P.; Marqués F.; Pereira F.; Gasull A. (2000). A contour based approach to binary shape coding using multiple grid chain code. *Signal Processing: Image Communication*, Elsevier Science, vol. 15, iss. 7-8, pp. 585-599. https://doi.org/10.1016/s0923-5965(99)00041-7

[10] Sánchez-Cruz H.; Bribiesca E.; Rodríguez-Dagnino R. M. (2007). Efficiency of chain codes to represent binary objects. *Pattern Recognition*, Elsevier Science, vol. 40, iss. 6, pp. 1660-1674. https://doi.org/10.1016/j.patcog.2006.10.013

[11] Žalik B.; Lukač N. (2014). Chain code lossless compression using Move-To-Front transform and adaptive Run-Length Encoding. *Signal Processing: Image Commun.*, Elsevier Science, vol. 29, iss.1, pp. 96-106. https://doi.org/10.1016/j.image.2013.09.002

[12] Žalik B.; Mongus D.; Lukač N.; Rizman Žalik K. (2018). Efficient chain code compression with interpolative coding. *Information Sciences*, Elsevier Science, vol. 439-440, pp. 39-49. https://doi.org/10.1016/j.ins.2018.01.045