

Compressing a Set of CHoG Features

Vijay Chandrasekhar^a, Sam S. Tsai^a, Yuriy Reznik^b,
Gabriel Takacs^a, David M. Chen^a and Bernd Girod^a

^aStanford University, Stanford, CA

^bQualcomm Inc., San Diego, CA

ABSTRACT

State-of-the-art image retrieval pipelines are based on “bag-of-words” matching. We note that the original order in which features are extracted from the image is discarded in the “bag-of-words” matching pipeline. As a result, a set of features extracted from a query image can be transmitted in any order. A set of m unique features has $m!$ orderings, and if the order of transmission can be discarded, one can reduce the query size by an additional $\log_2(m!)$ bits. In this work, we compare two schemes for discarding ordering: one based on Digital Search Trees, and another based on location histograms. We apply the two schemes to a set of low bitrate Compressed Histogram of Gradient (CHoG) features, and compare their performance. Both schemes achieve approximately $\log_2(m!)$ reduction in query size for a set of m features.

Keywords: feature descriptors, CHoG, ordering gain, compression of sets

1. INTRODUCTION

Mobile visual applications allow users to use their camera phone to initiate search queries about objects in their visual proximity. Such applications can be used, e.g., for identifying products, comparison shopping, finding information about movies, CDs, real estate, print media or artworks. Commercial deployments of such systems include Google Goggles,¹ Nokia Point and Find,² Kooaba,³ Ricoh iCandy^{4–6} and Amazon Remembers.⁷

Feature compression has been identified as one of the key issues for reducing latency in such mobile visual search and augmented reality applications. The size of the data sent over the network needs to be as small as possible to offer an interactive and responsive user experience. Prior work in the field has shown that extracting descriptors on the mobile device and transmitting compressed descriptors can reduce query latency significantly.⁸ Several compression schemes have been proposed in recent literature. The Compressed Histogram of Gradients (CHoG) descriptor,⁸ Binary Robust Independent Elementary Features (BRIEF),⁹ Product Quantized SIFT descriptors¹⁰ are some examples of low bitrate descriptors.

The focus of prior work on feature compression has been primarily around compressing each individual descriptor to obtain a compact representation. Readers are referred to⁸ for a detailed survey of low bitrate descriptors. In this work, we study the problem of compressing a set of features jointly. In particular, we are interested in the problem where the order in which data are transmitted does not matter, as in the case of local image features.

State-of-the-art image retrieval pipelines are commonly based on “bag-of-words” matching, i.e., query features are vector-quantized and histograms of query and database features are compared to obtain a ranked list of database images.¹¹ We note that the original order in which features are extracted is discarded. As a result, a set of features extracted from a query image can be transmitted in any order. A set of m unique features has $m!$ orderings, and if the order of transmission can be discarded, we should be able to reduce the query image size by an additional $\log_2(m!)$ bits.

In prior work on compressing feature sets, Chen et al.¹² first noticed that a tree-based representation can be used to discard the order of elements stored in it. Chen et al. propose storing a vocabulary tree¹¹ on the mobile device and computing the “bag-of-words” histogram locally. Run-length encoding of the non-zero bins in the histogram results in a significant reduction in query size. One drawback of this approach is that it requires the dictionary to be stored on the mobile device, which might not be feasible on RAM constrained devices. Tsai et al.¹³ propose an ordering on features based on their (x, y) locations in the image. A histogram map is generated based on feature locations. The histogram map is then encoded efficiently to reduce the query size. Reznik proposes constructing codes for unordered sets using Digital Search Trees (DST).¹⁴ Chandrasekhar et al.¹⁵ extend the DST based scheme to compress a set of local image features.

Further author information- Send correspondence to Vijay Chandrasekhar at vijayc@stanford.edu



Figure 1. Typical query image with m features. Note that the features in the image can be transmitted in any order. As a result, by clever ordering of the feature set, one can reduce the query size by $\log_2(m!)$ bits.

In this work, we discuss and compare two schemes for compressing a set of features: Digital Search Trees (DST) coding¹⁵ and Location Histogram Coding (LHC).¹³ Local image features typically consist of descriptor information, and geometric information (e.g., x , y , scale and orientation). The DST-based scheme imposes an ordering on the descriptor information to achieve the $\log_2(m!)$ gain, while the LHC scheme orders the x , y location data. The advantage of both schemes is that they do not require storing a vocabulary tree on the mobile device. Here, we show how both schemes can be applied to a set of CHoG descriptors as they work well at low bitrates¹⁶ making them suitable for mobile visual search applications. Both DST and LHC schemes can be applied to a set of SIFT,¹⁷ SURF¹⁸ or other local feature descriptors.

The outline of the paper is as follows. In Section 2, we describe the DST algorithm for compressing a binary input sequence. We show how the DST algorithm can be applied to impose an ordering on a set of variable bitrate CHoG descriptors. In Section 3, we describe the LHC scheme, which imposes an ordering on the location data. In Section 4, we present results for the two schemes, and compare their performance.

2. DIGITAL SEARCH TREES CODING

In Section 2.1, we first describe the DST compression problem for a simple example: a set of fixed-length words produced by a symmetric memoryless source. In Section 2.2, we discuss how a set of input words can be organized into a DST and discuss different schemes for representing DSTs efficiently. Next, we discuss the compression and decompression algorithms in Section 2.3. Finally, in Section 2.4, we discuss how the scheme can be applied to a set of CHoG descriptors.

2.1 Problem description

Let $\{f_1, \dots, f_m\}$ be a set of words that we need to encode. For simplicity, we first assume that these words are binary, distinct, have the same length $|f_i| = n$, and produced by a *binary symmetric i.i.d source* (we relax these assumptions in subsequent sections). In other words, zeros and ones appear with same probability $p = 1 - p = 1/2$ regardless of their positions or order. The entropy rate of such a source is 1 bit,¹⁹ implying, that conventional sequential encoding of words f_1, \dots, f_m will cost at least mn bits on average. Hereafter, we will often refer to an example set of words shown in Table 1 (second column). In this case: $m = 8$, $n = 5$, and total length $mn = 8 \times 5 = 40$ bits. We show how tree-based representations can be used to reduce the number of bits.

2.2 Tree based representation

In order to construct a more compact representation of the set $\{f_1, \dots, f_m\}$, we employ a data structure known as a *Digital Search Tree*.²⁰⁻²² We start with a single root node, and assume that it corresponds to an empty word. We then pick our first word f_1 , and depending on the value of its first bit, we add a left or right branch to the root node, and insert a new node there. We also store a pointer to f_1 at that node. With second and subsequent words, we traverse the tree starting from the root node by following their leading bits. Once we hit a leaf, we extend the DST by creating a new node and storing a pointer to the current word in it. This process is repeated m times, so that all words from our input set $\{f_1, \dots, f_m\}$ are inserted. We assume that our words $\{f_1, \dots, f_m\}$ are longer than the *height* (longest path) of random DST,²³ and so they can be uniquely parsed.¹⁵ This assumption implies that all words will be different with high probability.

The DST structure constructed over our example set is shown in Figure 2. The paths from root to other nodes in the tree correspond to portions (prefixes) of words inserted in this structure. We list such prefixes in the third column in Table 1.

Table 1. Example set of binary words $\{f_1, \dots, f_m\}$.

Index <i>i</i>	Word <i>f_i</i>	DST Prefix <i>p_i</i>	Suffix <i>s_i</i>
1	01011	0	1011
2	00111	00	111
3	10001	1	0001
4	01010	01	010
5	10010	10	010
6	00001	000	01
7	00110	001	10
8	00000	0000	0
Bits:	$8 \times 5 = 40$	18	22

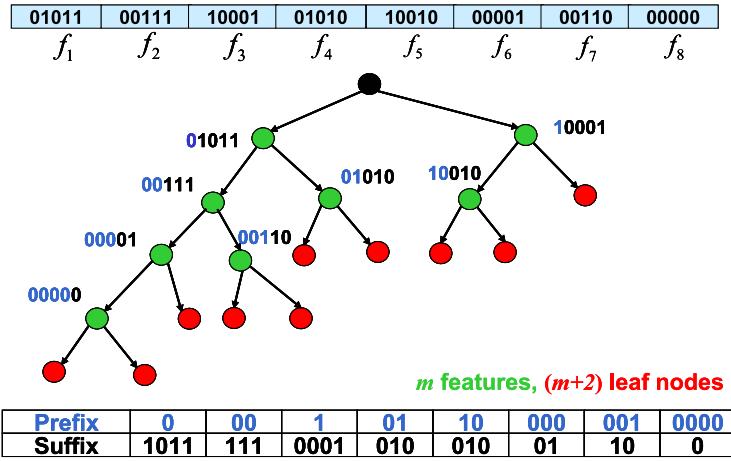


Figure 2. DST construction. Illustration of how a DST structure is constructed over our example set of words f_1, \dots, f_8 . For m features, the tree has $m + 1$ internal nodes and $m + 2$ external nodes. Each word or feature corresponds to an internal node in the tree. The prefixes and suffixes for each word are shown in blue and black respectively.

The fourth column in Table 1 lists the remainders (suffixes) of each word. We observe that DST construction effectively “splits” words f_i ($i = 1, \dots, m$) in two parts:

$$f_i = p_i | s_i,$$

where p_i are prefixes covered by paths in the tree, and s_i are the remaining suffixes. Overall lengths of prefixes and the suffixes will be denoted by

$$P_m = \sum_{i=1}^m |p_i|, \text{ and } S_m = \sum_{i=1}^m |s_i| = m n - P_m \quad (1)$$

correspondingly. In our example, shown in Table 1, the overall DST path length is $P_m = 18$, and the length of the remaining suffixes is $S_m = 40 - 18 = 22$.

In passing, we note that the order in which words $\{f_1, \dots, f_m\}$ are inserted may affect the number of bits that are “absorbed” by the tree. However, it does not change the average statistics of the tree (such as the expected path length $E P_n$), which we note is key to achieving the $\log_2(m!)$ ordering gain.

Our next task is to encode the structure of the DST efficiently. More specifically, we need to encode the shape of the binary tree. This tree contains $i = m + 1$ nodes: (m nodes associated with input words) + (one root node). We further add external or leaf nodes to the tree, as shown in Figure 2. This way, each feature corresponds to an interior node in the tree. This results in a tree with $m + 1$ internal nodes or $m + 2$ external nodes.

One simple technique for representing the tree is to scan it recursively using pre-order tree traversal,²⁴ and assigning labels “1” to the internal nodes, and “0” to external ones (see Figure 3). Such a sequence contains $2i + 1$ digits, and serves as a unique representation of a tree with i nodes.^{25,26} We call the resulting sequence of labels an x -sequence. The x -sequence may serve as a code to represent the DST, but we show how a more compact representation can be achieved. For the example in Figure 2, a pre-order traversal produces the sequence 1111100010010011000, with 1 and 0 representing internal and external nodes respectively.

In general, it is known, that the total number of possible ordered binary trees with i internal nodes is given by the Catalan number:²⁵

$$C_i = \frac{1}{i+1} \binom{2i}{i}, \quad (2)$$

implying that a tree can be uniquely represented by a fixed length code of

$$\lceil \log_2 C_i \rceil \sim 2i - \frac{3}{2} \log_2 i + O(1) \text{ [bits].} \quad (3)$$

We describe one possible coding technique based on the Zaks algorithm²⁶ that achieves this rate in Appendix A. The Zaks algorithm computes the index of the tree in the space of all possible trees with i internal nodes. For the example in Figure 2, the code of the tree is $\lceil \log_2 C_{m+1} \rceil = \lceil \log_2 C_9 \rceil = 13$ bits-long. As observed, this code is shorter than the $2i + 1 = 19$ bits required for the pre-order traversal representation.

We are now ready to describe the remaining steps in our DST coding scheme for sets of words.

2.3 Compression and Decompression Algorithms

Given a set of m words $\{f_1, \dots, f_m\}$, the proposed algorithm performs the following operations:

1. Build, encode, and transmit DST structure over the input set $\{f_1, \dots, f_m\}$.
2. Scan the tree recursively, and define a canonical order of nodes and the corresponding prefixes p_{i_1}, \dots, p_{i_m} in the DST.
3. Encode and transmit suffixes according to same order s_{i_1}, \dots, s_{i_m} .

The construction of the DST structure and its encoding is performed as discussed in previous sections. To define a canonical order of nodes we use the standard pre-order tree traversal,²⁴ and assign each node a serial number, starting with 0, assigned to the root node (see Figure 3). As we reach a j -th node during the traversal, we can also recover the prefix of a word f_{i_j} that was inserted in it. This produces an order i_1, \dots, i_m in which prefixes of all words from our set can be retrieved from the tree. We omit the root node in this sequence. For example, for a tree in Figure 2, this produces the ordering $i_1 = 1, i_2 = 2, i_3 = 6, i_4 = 8, i_5 = 7, i_6 = 4, i_7 = 3, i_8 = 5$. In order to transmit information about corresponding suffixes, we simply arrange and encode them in the same order: s_{i_1}, \dots, s_{i_m} . Any standard source coding technique (such as Huffman coding or arithmetic coding) can be applied to this sequence.

The decoder performs the following inverse operations:

1. Decode the DST tree structure.
2. Scan nodes in the same order as encoder, and reconstruct prefixes p_{i_1}, \dots, p_{i_m} .
3. Sequentially decode corresponding suffixes s_{i_1}, \dots, s_{i_m} , and form complete decoded words: $f_{i_j} = p_{i_j} s_{i_j}, j = 1, \dots, m$.

We conclude our presentation of the algorithm by showing a complete code constructed for our example set of words (see Table 1, and Figures 2, 3, 4). Note that the DST constructed from the set of input features in Table 1 can be represented by the Zaks algorithm with index 381 (13 bits) (see Appendix A).

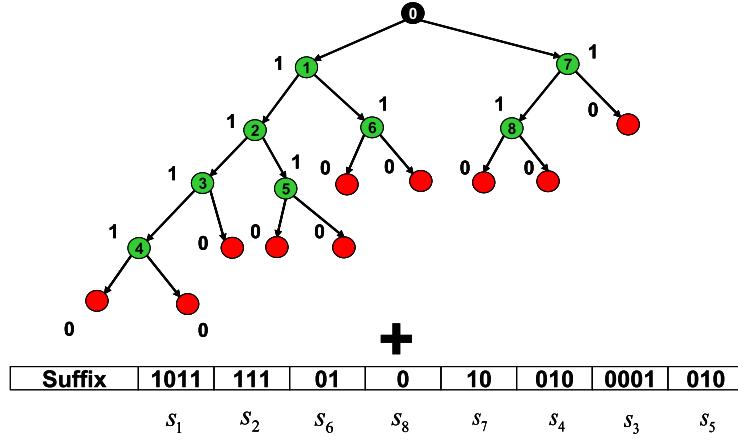


Figure 3. DST compression. Once the DST is constructed, the word set can be fully represented by (1) the structure of the tree, and (2) an ordered set of suffixes. Note that the structure of the tree captures the prefixes of all the words. For (1), the DST can be represented by a pre-order traversal. A pre-order travel produces the sequence 1111100010010011000, with 1 and 0 representing internal and external nodes respectively. The order of the traversal is indicated within each node. A more compact representation of the tree structure can be obtained by computing an index of the tree in the space of all possible trees with $m + 2$ external nodes, using the Zaks algorithm. For (2), the pre-order traversal imposes an ordering on the words, and the corresponding reordered suffixes, $s_1, s_2, s_6, s_8, s_7, s_4, s_3, s_5$ are shown below the tree.

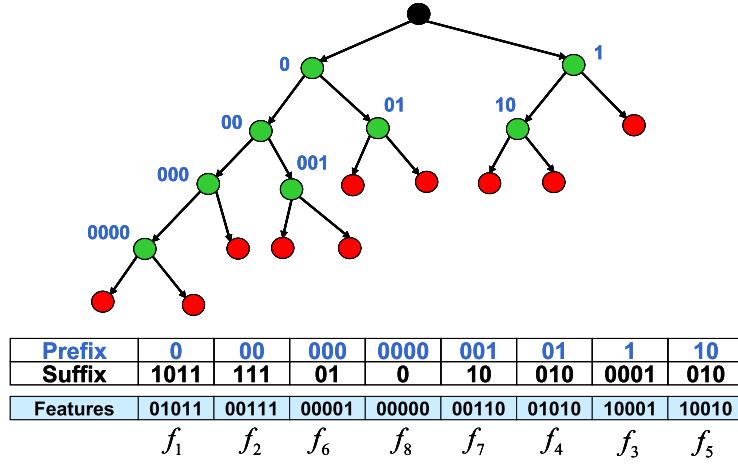


Figure 4. Decoding the DST coded data. First, the pre-order tree traversal code or Zaks index is used to reconstruct the tree. Next, the pre-order traversal of the tree is used to generate the prefixes of the features. The prefixes are combined with the corresponding suffixes to generate the final set of features (reordered).

$$\begin{aligned}
 \text{Code}(\{f_1, \dots, f_m\}) &= \text{Bin}_{\lceil C_{m+1} \rceil}(\text{index}), s_{i_1}, \dots, s_{i_m} \\
 &= \text{Bin}_{\lceil C_9 \rceil}(381), s_1, s_2, s_6, s_8, s_7, s_4, s_3, s_5 \\
 &= 000010111101, 1011, 111, 01, \\
 &\quad 0, 10, 010, 0001, 010.
 \end{aligned}$$

As evident, the length of this code is $13 + 22 = 35$ bits, which is by $40 - 35 = 5$ bits shorter than the length of a straightforward sequential encoding of words in this set.

In prior work,^{14,15} we prove that the DST coding scheme reduces the bitrate by $\sim \log_2(m!)$ bits. Intuitively, we do not transmit the prefixes for each feature, which results in $\sim \log_2 m$ savings for each feature. For a set of m features, this

results in a $\sim m \log_2 m$ reduction in bitrate, which by Sterling's approximation is close to $\log_2(m!)$. More precisely,¹⁵

$$L_{DST}(m, t) = Ht - \log_2 m! + O(m). \quad (4)$$

where L_{DST} is the length of the set with DST coding, t is the total length of the input source in bits, m is the number of words, and H is the entropy of the source. The results holds for a broad variety of input sources. Readers are referred to^{15,27} for details of the proof.

Finally, we show how the DST coding scheme can be applied for efficiently encoding a set of Compressed Histogram of Gradients²⁸ descriptors.

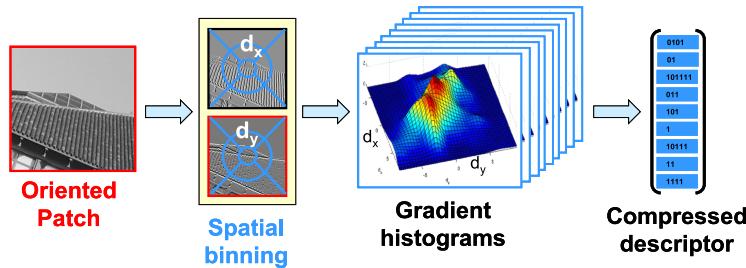


Figure 5. CHoG descriptor computation pipeline. Canonical patches around interest points are divided into log polar spatial bins. The gradient histogram in each spatial bin is represented by a variable length code.

2.4 Application to CHoG features

2.4.1 CHoG Feature Structure

First, we briefly review the structure of the CHoG descriptor and how it is computed. The pipeline for computing CHoG descriptors is shown in Figure 5. We first start with patches obtained from interest points (e.g., corners, blobs) at different scales. The patches at different scales are oriented along the dominant gradient. Next, we divide the scaled and oriented canonical patches into log-polar spatial bins. Then, we perform independent quantization of histograms in each spatial bin. The histogram in each spatial bin is quantized using Huffman Trees, Type Coding or Vector Quantization, and mapped to an index.⁸ The resulting indices are then encoded using variable length codes, based on their different probabilities. The final bitstream of the feature descriptor is obtained by a concatenation of codes representative of histograms in each spatial bin.

2.4.2 Compression and Decompression Algorithm

The DST compression and decompression schemes are similar to those in Section 2.3. One important difference is that CHoG features are of variable length. We show how the DST algorithm can be extended to handle variable length features. Note, also, that features need not be unique. However, duplicate CHoG features within the same image are highly unlikely. In case duplicate features are observed, a small number of bits can be used to signal the count of non-unique features.

We use the same DST coding scheme described in Section 2.3. Let S be the number of variable-length Huffman codes, representing the S spatial components of each CHoG descriptor. Previously, the length of each feature was fixed, and so, it was used to determine the termination of suffix symbols for each feature. Now the decoding of S Huffman codes signal termination of each feature. The modified decoding algorithm is shown below:

1. Decode the DST tree structure

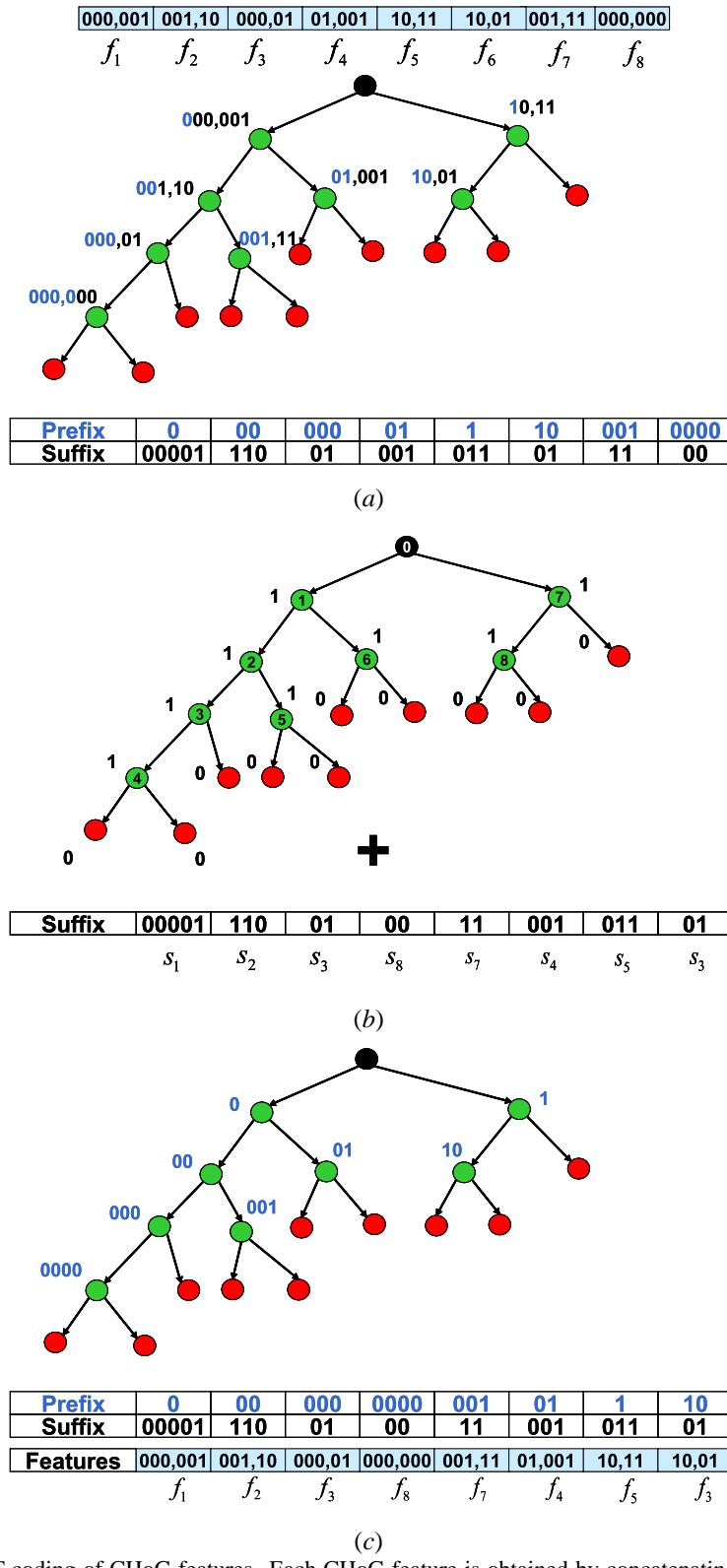


Figure 6. Illustration of DST coding of CHoG features. Each CHoG feature is obtained by concatenating a number of variable length prefix-free Huffman codes (one for each spatial bin). In this example, each CHoG feature is obtained by concatenating of two prefix-free Huffman codes from the set (000), (001), (01), (10), (11). In Figure (a), we construct the DST structure in the same fashion as before. Note that the features are now of variable length. Multiple symbols within each feature are shown as comma-delimited. Next, as shown in Figure (b), we order the features based on a pre-order traversal and transmit the tree structure, and the corresponding reordered suffixes. Finally, in the decompression step, we reconstruct the DST from the tree code. After scanning prefixes, we scan the suffix stream until we decode 2 Huffman prefix-free codes and reconstruct each feature descriptor.

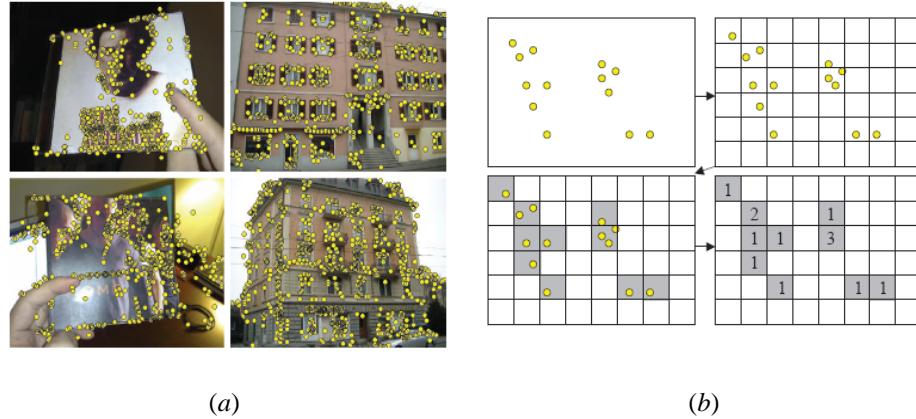


Figure 7. (a) Interest point locations in images tend to cluster spatially. (b) We represent the location of the features using a location histogram. The image is first divided into evenly spaced blocks. We enumerate the features within each spatial block generating a location histogram.

2. Scan nodes in the same order as encoder, and retrieve prefixes p_{i_1}, \dots, p_{i_m}
3. Scan the suffix stream until S Huffman encoded parts of CHoG descriptor are decoded. The complete decoded feature is obtained by combining the prefix and suffix data. The process is repeated until all features f_{i_j} are decoded.

An example of CHoG DST coding is illustrated in Figure 6. In the example shown in Figure 6, each CHoG descriptor is obtained by concatenating $S = 2$ variable length prefix-free Huffman codes from the set ((000), (001), (01), (10), (11)).

3. LOCATION HISTOGRAM CODING

In Section 2, we discuss how a DST-based representation can be used to order a set of feature descriptors. Here, we consider an alternate approach proposed in prior work¹³ which uses location data associated with features to order the feature set. Each interest point has a location, scale and orientation associated with it. Interest point locations are needed in the geometric verification step to validate potential candidate matches. The location of each interest point is typically stored as two numbers: x and y co-ordinates in the image at sub-pixel accuracy.¹⁷

The LHC scheme orders the feature set by their (x, y) location to achieve the ordering gain using location histograms. Furthermore, it exploits the spatial correlation of features in images. As shown in Fig. 7(a), the interest points in images are spatially clustered.

To code location data, we first generate a histogram from the locations of the descriptors (see Fig. 7 (b)). We divide the image into spatial bins and create a 2-D histogram of the number of features within each spatial bin. We quantize the (x, y) location to 4-pixel blocks as it suffices for precise geometric verification.¹³

Encoding the locations of a set of m features as a histogram reduces the bitrate by $\sim \log_2(m!)$, compared to encoding each feature location using fixed length codes.¹³ Let m denote the number of features in the image, and let n denote the number of spatial bins in the histogram. Typically, for a VGA resolution image, there are ~ 1000 features in the image. Let w be the size of a spatial bin (in pixels). Note that $n = (640 \times 480)/w^2$. The number of all possible histograms with n cells and m features is given by a multiset coefficient, i.e., the number of partitions of parameter m into n terms $k_1 + \dots + k_n = m$ is:

$$\binom{n}{m} = \binom{m+n-1}{n-1}. \quad (5)$$

Typically, for small w (e.g., 4×4 block), $n \gg m$. The number of bins in the histogram is much larger than the number of features to be encoded. Consequently, the total number of bits needed to encode the histogram is as follows:

$$R_{hist}(n, m) \leq \lceil \log_2 \binom{n}{m} \rceil \sim m \log_2 n - \log_2(m!) \text{ if } n \gg m. \quad (6)$$

	9	6	10	
7	3	2	4	8
5	1	X		

Figure 8. Context used in encoding binary map in LHC.

From Equation 6, we can see that encoding the histogram can potentially reduce the bitrate by $\sim \log_2(m!)$, compared to fixed length encoding of each feature location with $\log_2(n)$ bits. Readers are referred to^{13,29} for techniques to encode histograms. Here, we discuss one efficient coding technique that can be used to achieve the ordering gain $\log_2(m!)$. We represent the histogram using a binary map, indicating which spatial bins contains features, and a sequence of feature counts, representing the number of features in occupied bins (see Figure 7). The two sequences are encoded separately. We encode the binary map in a raster scan order using a trained context-based arithmetic coder, with neighbouring bins being used as the context for each spatial bin. A context of size 10 is used, as shown in Figure 8. Finally, if precision beyond 4×4 pixels is needed, it be obtained by sending refinement bits to indicate where features lie within each block.

With LHC, we can exploit the spatial correlation between the locations of different features as illustrated in Fig. 7, using neighbouring bins as context. Even if the feature points are uniformly scattered in the image, LHC is still able to exploit the ordering gain, which results in $\sim \log_2(m!)$ saving in bits. Finally, we present results and compare DST and LHC schemes for compressing a set of feature descriptors.

4. RESULTS

For evaluating the performance of the set coding schemes, we use 1000 images from the *Mixed Text and Graphics* data set of the MPEG evaluation framework for “Compact Descriptors for Visual Search”.^{30,31} The images are also available as part of the Stanford Mobile Visual Search data set.³² All processing is done on VGA resolution images. For feature extraction, we use a DoG interest point detector and ~ 70 -bit reference CHoG descriptor, which consists of 9 spatial bins, represented by 9 variable-length Huffman codes.³³ The number of features is varied by selecting features with the highest Hessian response for a given feature budget. In our experiments, we quantize (x, y) locations to 4-pixel blocks.

In Figure 9, we vary the number of features, and present results for the following schemes:

1. Original: The original order of features is maintained. Feature descriptors and their locations are entropy coded.
2. DST: Feature descriptors are reordered using the DST coding scheme. The x, y locations of features are entropy coded for the new ordering.
3. LHC: Features are reordered based on their (x, y) location. Feature descriptors are entropy coded according to the new ordering.
4. Original-Ordering Gain: This refers to the expected savings in bitrate by discarding ordering of features. The predicted gain is shown in Figure 9(a).

First, we observe that both DST and LHC schemes result in close to $\log_2(m!)$ savings over a range of query sizes. At the highest query size, we can reduce the amount of data by 0.5 KB. Second, we note that there is a small gap between the predicted $\log_2(m!)$ ordering gain and the DST scheme. The small gap arises, as predicted by theory, due to the $O(m)$ term in Equation 4. Third, at high rates, we observe that the LHC scheme slightly outperforms the ordering gain. This is because LHC not only exploits the ordering gain achieved by using histogram coding, but also the spatial correlation

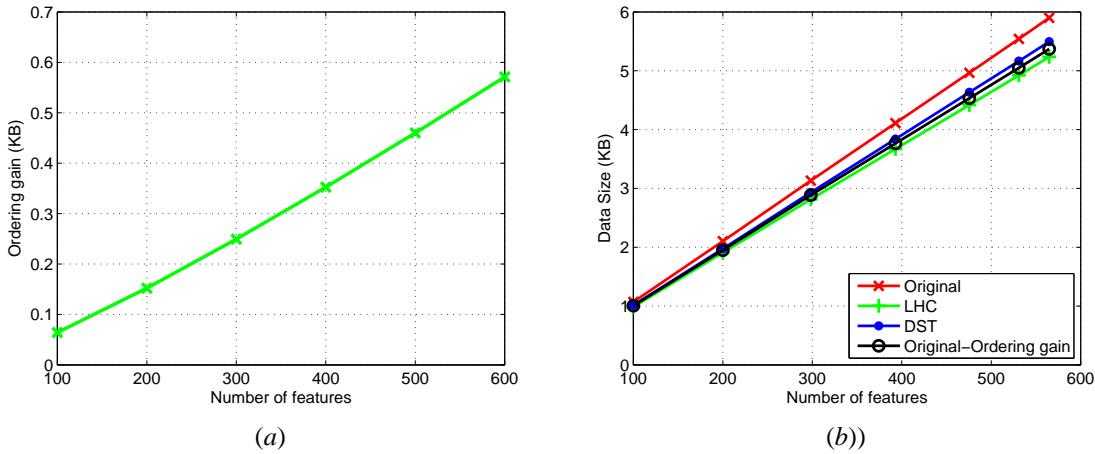


Figure 9. (a). Reduction in data size by discarding order. (b). Results of DST and LHC coding schemes for CHoG descriptors. We note that both schemes reduce the data by $\sim \log_2(m!)$ bits, the ordering gain.

Table 2. Coefficients $a_{i,j}$ used in lexicographic enumeration of trees.

$i\backslash j$	0	1	2	3	4	5	6	7	8
1	1								
2	2	1							
3	5	3	1						
4	14	9	4	1					
5	42	28	14	5	1				
6	132	90	48	20	6	1			
7	429	297	165	75	27	7	1		
8	1430	1001	572	275	110	35	8	1	
9	4862	3432	2002	1001	429	154	44	9	1

between features. By using the context of nearby histogram bins, we effectively predict the feature location distribution, and hence can compress the location histogram further. For a set of m features, the DST coding scheme is $O(m \log(m))$ in complexity, the time it takes to build the search tree. The LHC scheme is $O(m)$ in complexity, the time it takes to build the location histogram. Finally, we note that both LHC and DST schemes have their advantages. The DST scheme is universal and can be used for arbitrary input sources. The LHC coding scheme, on the other hand, exploits the spatial correlations between feature locations for different interest point detectors.

5. CONCLUSIONS

A set of features extracted from an image can be transmitted in any order. A set of m unique features has $m!$ orderings, and if the order of transmission can be discarded, one can reduce the query size by an additional $\log_2(m!)$ bits. In this work, we compare two schemes for discarding ordering: one based on Digital Search Trees, and another based on location histograms. We apply the two schemes to a set of low bitrate Compressed Histogram of Gradient (CHoG) descriptors, and compare their performance. Both schemes achieve approximately $\log_2(m!)$ reduction in query size for a set of m features.

APPENDIX A. ZAKS ALGORITHM

The Zaks tree enumeration algorithm is used to generate an index of the tree structure. The algorithm is briefly reproduced here, readers are referred to²⁶ for more details. Given an x -sequence for a tree, we produce a list of positions of symbols “1” in it. We will call it a z -sequence $z = z_1, \dots, z_i$. For example, for a sequence $x = 1111100010010011000$, corresponding to a tree in Figure 3, we produce: $z = 1, 2, 3, 4, 5, 9, 12, 15, 16$. We next define a rule for incremental reduction of z -sequences. Let j^* be the largest j , such that $z_j = j$. By $z^* = z_1^*, \dots, z_{i-1}^*$ we will denote a new sequence that omits value

z_{j^*} , and subtracts 2 from all subsequent values in the original sequence:

$$z_j^* = \begin{cases} z_j, & j = 1, \dots, j^* - 1; \\ z_{j+1} - 2, & j \geq j^*. \end{cases}$$

Then, a lexicographic index (or *Zaks rank*) of a tree is recursively computed as follows:²⁶

$$\text{index}(z) = \begin{cases} 1, & \text{if } j^* = i; \\ a_{i,j^*} + \text{index}(z^*), & \text{if } j^* < i, \end{cases} \quad (7)$$

where

$$a_{i,j} = \frac{j+2}{2i-j} \binom{2i-j}{i-j-1}, \quad 0 \leq j \leq i-1$$

are some constants (see Table 2).

For example, for the tree in Figure 3, Zaks ranking algorithm (7) produces:

$$\begin{aligned} \text{index}(1, 2, 3, 4, 5, 9, 12, 15, 16) &= a_{9,5} + \text{index}(1, 2, 3, 4, 7, 10, 13, 14) \\ \text{index}(1, 2, 3, 4, 7, 10, 13, 14) &= a_{8,4} + \text{index}(1, 2, 3, 5, 8, 11, 12) \\ \text{index}(1, 2, 3, 5, 8, 11, 12) &= a_{7,3} + \text{index}(1, 2, 3, 6, 9, 10) \\ \text{index}(1, 2, 3, 6, 9, 10) &= a_{6,3} + \text{index}(1, 2, 4, 7, 8) \\ \text{index}(1, 2, 4, 7, 8) &= a_{5,2} + \text{index}(1, 2, 5, 6) \\ \text{index}(1, 2, 5, 6) &= a_{4,2} + \text{index}(1, 3, 4) \\ \text{index}(1, 3, 4) &= a_{3,1} + \text{index}(1, 2) \\ \text{index}(1, 2) &= 1; \end{aligned}$$

resulting in

$$\begin{aligned} \text{index}(1, 2, 3, 4, 5, 9, 12, 15, 16) &= a_{9,5} + a_{8,4} + a_{7,3} + a_{6,3} \\ &\quad + a_{5,2} + a_{4,2} + a_{3,1} + 1 \\ &= 154 + 110 + 75 + 20 \\ &\quad + 14 + 4 + 3 + 1 \\ &= 381. \end{aligned}$$

The code of this tree is a $\lceil \log_2 C_{m+1} \rceil = \lceil \log_2 C_9 \rceil = 13$ bits-long binary record of its index:

$$\text{Bin}_{\lceil \log_2 C_{m+1} \rceil}(\text{index}) = \text{Bin}_{13}(381) = 0000101111101.$$

As easily observed, this code is shorter than the $2i + 1 = 19$ bits required for the pre-order traversal representation.

REFERENCES

- [1] Google-Goggles (2009). www.google.com/mobile/goggles/.
- [2] Nokia, *Nokia Point and Find* (2006). <http://www.pointandfind.nokia.com>.
- [3] Kooaba, *Kooaba* (2007). <http://www.kooaba.com>.
- [4] Erol, B., Antúnez, E., and Hull, J., “Hotpaper: multimedia interaction with paper using mobile phones,” in [*Proc. of the 16th ACM Multimedia Conference*], (2008).
- [5] Graham, J. and Hull, J. J., “Icandy: a tangible user interface for itunes,” in [*Proc. of CHI ’08: Extended abstracts on human factors in computing systems*], (2008).
- [6] Hull, J. J., Erol, B., Graham, J., Ke, Q., Kishi, H., Moraleda, J., and Olst, D. G. V., “Paper-based augmented reality,” in [*Proc. of the 17th International Conference on Artificial Reality and Telexistence (ICAT)*], (2007).
- [7] Amazon, *SnapTell* (2007). <http://www.snaptell.com>.
- [8] Chandrasekhar, V., Takacs, G., Chen, D. M., Tsai, S. S., Grzeszczuk, R., Reznik, Y., and Girod, B., “Compressed Histogram of Gradients: A Low Bitrate Descriptor,” in [*International Journal of Computer Vision, Special Issue on Mobile Vision*], (2010). Accepted.
- [9] Calonder, M., Lepetit, V., and Fua, P., “Brief: Binary robust independent elementary features,” in [*Proc. of European Conference on Computer Vision (ECCV)*], (October 2010).

- [10] Jegou, H., Douze, M., and Schmid, C., "Product Quantization for Nearest Neighbor Search," *Accepted to IEEE Transactions on Pattern Analysis and Machine Intelligence* (2010).
- [11] Nistér, D. and Stewénius, H., "Scalable recognition with a vocabulary tree," in [Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)], (June 2006).
- [12] Chen, D. M., Tsai, S. S., Chandrasekhar, V., Takacs, G., Singh, J., and Girod, B., "Tree histogram coding for mobile image matching," in [Proc. of IEEE Data Compression Conference (DCC)], (March 2009).
- [13] Tsai, S. S., Chen, D. M., Takacs, G., Chandrasekhar, V., Singh, J. P., and Girod, B., "Location coding for mobile image retrieval systems," in [Proc. of International Mobile Multimedia Communications Conference (MobiMedia)], (September 2009).
- [14] Reznik, Y., "Coding sets of words," in [Proc. of IEEE Data Compression Conference (DCC)], (March 2011).
- [15] Chandrasekhar, V., Reznik, Y., Takacs, G., Chen, D. M., Tsai, S. S., Grzeszczuk, R., and Girod, B., "Compressing a set of features with digital search trees," in [Submitted to IEEE International Conference on Computer Vision (ICCV)], (October 2011).
- [16] Girod, B., Chandrasekhar, V., Chen, D. M., Cheung, N. M., Grzeszczuk, R., Reznik, Y., Takacs, G., Tsai, S. S., and Vedantham, R., "Mobile Visual Search," in [Proceedings of IEEE Signal Processing Magazine, Special Issue on Mobile Media Search], (2010).
- [17] Lowe, D., "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision* **60**(2), 91–110 (2004).
- [18] Bay, H., Tuytelaars, T., and Gool, L. V., "SURF: Speeded Up Robust Features," in [Proc. of European Conference on Computer Vision (ECCV)], (May 2006).
- [19] Cover, T. M. and Thomas, J. A., [Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)], Wiley-Interscience (2006).
- [20] E. G. Coffman, J. and Eve, J., "File structures using hashing functions," *Communications of the ACM* **13**(7), 427–436 (1970).
- [21] Flajolet, P. and Sedgewick, R., "Digital search trees revisited," *SIAM Journal of Computing* **15**, 748–767 (1986).
- [22] Knuth, D., [The Art of Computer Programming. Sorting and Searching. Vol. 3], Addison-Wesley, Reading MA (1973).
- [23] Pittel, B., "Asymptotic growth of a class of random trees," *Annals of Probability* **18**, 414–427 (1985).
- [24] Sedgewick, R., [Algorithms. Parts 1-4. Fundamentals, Data Structures, Sorting, Searching], Addison-Wesley, Reading MA (1998).
- [25] Knuth, D., [The Art of Computer Programming. Fundamental Algorithms. Vol. 1], Addison-Wesley, Reading MA (1968).
- [26] Zaks, S., "Lexicographic generation of ordered trees," *Theoretical Computer Science* **10**, 63–82 (1980).
- [27] Reznik, Y., "Coding for unordered sets of words," in [Proc. of IEEE International Symposium on Information Theory (ISIT)], (July 2011).
- [28] Chandrasekhar, V., Takacs, G., Chen, D. M., Tsai, S. S., Grzeszczuk, R., and Girod, B., "CHoG: Compressed Histogram of Gradients - A low bit rate feature descriptor," in [Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)], (June 2009).
- [29] Chandrasekhar, V., Reznik, Y., Takacs, G., Chen, D., Tsai, S., Grzeszczuk, R., and Girod, B., "Quantization schemes for low bitrate compressed histogram of gradients descriptors," in [Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on], 33–40 (june 2010).
- [30] "Compact descriptors for visual search: Evaluation framework," ISO/IEC JTC1 SC29 WG11 output document N12202 (July 2011).
- [31] "Compact descriptors for visual search: Call for proposals," ISO/IEC JTC1 SC29 WG11 output document N12201 (July 2011).
- [32] Chandrasekhar, V., D.M.Chen, S.S.Tsai, N.M.Cheung, H.Chen, G.Takacs, Y.Reznik, R.Vedantham, R.Grzeszczuk, J.Back, and B.Girod, *Stanford Mobile Visual Search Data Set* (2010). http://mars0.stanford.edu/mvs_images/.
- [33] *Compressed Histogram of Gradients - binary release* (2010). <http://www.stanford.edu/~dmchen/mvs.html>.