Inverted Index Compression for Scalable Image Matching

David M. Chen[1], Sam S. Tsai[1], Vijay Chandrasekhar[1], Gabriel Takacs[1],
Ramakrishna Vedantham[2], Radek Grzeszczuk[2], Bernd Girod[1]
[1]Department of Electrical Engineering, Stanford University, Stanford, CA 94305
[2]Nokia Research Center, Palo Alto, CA 94304

Abstract

To perform fast image matching against large databases, a Vocabulary Tree (VT) uses an inverted index that maps from each tree node to database images which have visited that node. The inverted index can require gigabytes of memory, which significantly slows down the database server. In this paper, we design, develop, and compare techniques for inverted index compression for image-based retrieval. We show that these techniques significantly reduce memory usage, by as much as $5\times$, without loss in recognition accuracy. Our work includes fast decoding methods, an offline database reordering scheme that exploits the similarity between images for additional memory savings, and a generalized coding scheme for soft-binned feature descriptor histograms. We also show that reduced index memory permits memory-intensive image matching techniques that boost recognition accuracy.

1. Introduction

In the past few years, advances in visual search have led to the development of numerous image-based retrieval systems [1] [2] [3] [4]. Most of the advances are based on local image features such as SIFT [5], SURF [6], and CHoG [7] which can be matched robustly despite photometric and geometric distortions. Equally important for large-scale image search are data structures which efficiently index local features, and the most notable of these are the Vocabulary Tree (VT) [8] and its recent extensions [9] [10]. To facilitate fast search through a database containing billions of features, the VT splits the feature descriptor space by tree-structed vector quantization (TSVQ), where the VQ centroids are the tree nodes. Each image's set of feature descriptors can then be compactly summarized by a tree histogram, which counts how often each tree node is visited.

To speed up image search, the VT of [8] uses an inverted index. For each tree node, the inverted index stores a list of image IDs indicating which database images have visited that node, as well as the visit frequencies. Fig. 1 shows a VT-based retrieval system with an inverted index. During a query, features are extracted from the query image and the feature descriptors are classified through the VT. Then, similarity scores are efficiently computed for all database images by traversing only the inverted lists at tree nodes visited by the query feature descriptors. Finally, the few hundred database images most similar to the query image are compared more thoroughly by pairwise geometric consistency checks.

A key challenge to scaling image search up to larger databases is the amount of memory consumed. In a VT-based system, the most memory-intensive structure is the inverted index. For example, in a database of one million images where each image contains hundreds of features, the inverted index consumes nearly 3 GB of memory. Such large memory usage limits the ability to run other concurrent processes on the same server, such as recognition systems for other

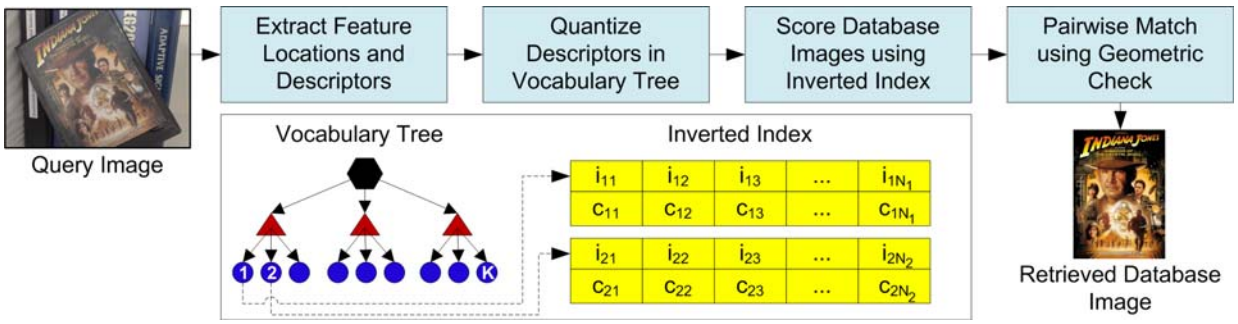


Figure 1. Image retrieval system based on a vocabulary tree with inverted index.

databases. When the index's memory usage becomes too large, swapping between main and virtual memory occurs, which significantly slows all processes.

In this paper, we design and implement efficient methods for compressing an inverted index in memory. By exploiting the statistics of image IDs and visit counts contained in the inverted lists, memory usage can be reduced as much as $5\times$. As the database size increases, our coding methods become more efficient, i.e., spend fewer bits in memory per image. Special care is taken to achieve short decoding times, so that the compressed inverted index can be effectively integrated into our real-time mobile image search system [2]. With fast decoding, reducing the inverted index's memory usage can actually speed up retrieval on a server with limited memory, because long delays due to memory congestion are avoided.

Inverted index compression has been previously studied in text retrieval [11] [12] [13] [14], where memory usage was also identified as a bottleneck hindering scalability of large-scale text search. A good comprehensive overview is given in [15]. Recently, a compressed inverted index for image databases was proposed in [16] but differs from our work in a couple of important aspects. Whereas the authors of [16] only measure entropies, we measure actual memory usage and develop practical fast decoding techniques for real-time image matching. For simplicity, only binary counts are considered in [16], while our coding scheme more generally compresses multilevel counts and even fractional counts.

The rest of the paper is organized as follows. In Sec. 2, we review VT-based image retrieval in greater detail and highlight the memory usage of an uncompressed inverted index. Our compressed inverted index is presented in Sec. 3, where first the principal coding methods are explained and then important extensions are described. Sec. 4 shows experimental results for compressing large databases. We demonstrate that memory usage is reduced $5\times$ with our basic coding scheme and by an additional 37 percent with a database reordering, while incurring very small decoding delays. For large databases, index compression allows us to avert the long delays caused by swapping between main and virtual memory. The reduced memory also enables useful memory-intensive image matching techniques, like soft-binned tree search for more accurate feature descriptor quantization and multi-view trees for better matching under perspective distortion.

2. Background on Vocabulary Tree Search

The Vocabulary Tree is a tree-structured vector quantizer constructed by hierarchical k-means clustering of feature descriptors. If the VT has L levels excluding the root node and each interior node has C children, then a fully balanced VT contains $K = C^L$ leaf nodes. Fig. 1 shows a VT with $L = 2$, $C = 3$, and $K = 9$. In practice, $L = 6$ and $C = 10$ are commonly selected for good classification performance [8], in which case the VT has $K = 10^6$ leaf nodes.

The inverted index associated with the VT maintains two lists per leaf node. For node k, there is a sorted array of image IDs $\{i_{k1}, i_{k2}, \cdots, i_{kN_k}\}$ indicating which N_k database images have visited that node. Similarly, there is a corresponding array of counts $\{c_{k1}, c_{k2}, \cdots, c_{kN_k}\}$ indicating the frequency of visits. During a query, a database of N total images can be quickly scored by traversing only the nodes visited by the query descriptors. Let $s(i)$ be the similarity score for the i^{th} database image. Initially, $s(i)$ is set to 0. Suppose node k is visited by the query descriptors a total of q_k times. Then, that node's inverted list of images $\{i_{k1}, \cdots, i_{kN_k}\}$ will all have their scores incremented. The score for image i_{k1} will be updated as

$$s(i_{k1}) := s(i_{k1}) + \frac{w_k^2 c_{k1} q_k}{\Sigma_{i_{k1}} \Sigma_q} \quad , \tag{1}$$

where w_k is an inverse document frequency (IDF) weight used to penalize often-visited nodes, $\Sigma_{i_{k1}}$ is a normalization for database image i_{k1}, and Σ_q is a normalization for the query image.

$$w_k = \log\left(N/N_k\right) \quad , \tag{2}$$

$$\Sigma_{i_{k1}} = \sum_{j=1}^{K} w_j \left(\text{count for database image } i_{k1} \text{ at node } j\right) \quad , \tag{3}$$

$$\Sigma_q = \sum_{j=1}^{K} w_j \left(\text{count for query image at node } j\right) \quad . \tag{4}$$

Scores for images $\{i_{k2}, \cdots, i_{kN_k}\}$ are updated similarly, as are scores for images at other nodes.

In a database with one million images and a VT with $K = 10^6$ leaf nodes, each image ID can be stored in a 32-bit unsigned integer and each count fits in an 8-bit unsigned integer. The memory usage of the entire inverted index is $\sum_{k=1}^{K} N_k \cdot (32 \text{ bits} + 8 \text{ bits})$, where N_k is the length of the inverted list at the k^{th} tree node. For a database of one million CD/DVD/book covers, we measured nearly 3 GB memory usage by the inverted index. It is this substantial memory usage we reduce in the next section by creating a compressed inverted index.

3. Compression of Inverted Index

In this section, we design a compressed inverted index that significantly reduces memory usage. Our basic coding scheme in Sec. 3.1 is lossless and does not affect matching accuracy. Large compression gains can be realized by exploiting the statistics of the image IDs and counts contained in the inverted lists. In Sec. 3.2, we present fast decoding techniques which give almost the same coding efficiency as the basic coding scheme. Keeping decoding latency low is important for integrating the compressed inverted index into a real-time retrieval system. An interesting inter-image correlation phenomenon is described in Sec. 3.3 and exploited to yield further coding gains. Lastly, in Sec. 3.4, soft binning for tree search is introduced to improve descriptor quantization. Because soft binning requires fractional counts, a generalized index coding method with Lloyd-Max quantization is developed.

3.1. Encoding of Image IDs and Visit Counts

The memory expression $\sum_{k=1}^{K} N_k \cdot (32 \text{ bits} + 8 \text{ bits})$ in Sec. 2 applies to an uncompressed inverted index. First, because each list of image IDs $\{i_{k1}, i_{k2}, \cdots, i_{kN_k}\}$ is sorted, it is more efficient to store consecutive ID differences $\left\{d_{k1} = i_{k1}, d_{k2} = i_{k2} - i_{k1}, \cdots, d_{kN_k} = i_{kN_k} - i_{k(N_k-1)}\right\}$ in place of the IDs. This practice is also used in text retrieval [15]. Second, the distributions of the ID differences and visit counts are far from uniform, as plotted in Fig. 2 for a database

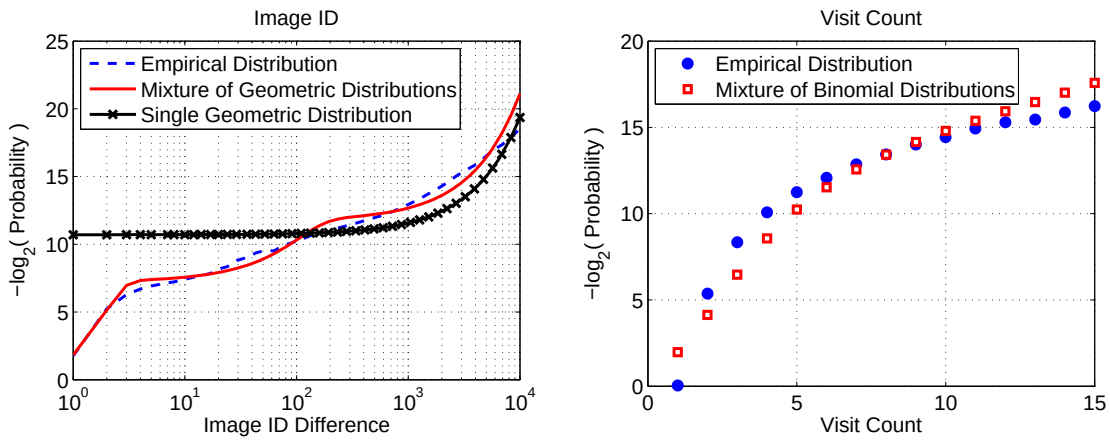


Figure 2. Probability distributions for (a) Image ID differences and (b) Visit counts.

of one million CD/DVD/book cover images. These peaky distributions suggest that variable-length coding can be much more rate-efficient than fixed-length coding. The ID difference and count distributions can be well modeled as mixtures of geometric and binomial distributions, respectively [17]. In fact, the deviation of the ID difference distribution from that of a single geometric random variable is beneficial for coding, a property we further exploit in Sec. 3.3.

Using the probabilities of the ID differences and counts, we encode each list $\{d_{k1}, \cdots, d_{kN_k}\}$ and $\{c_{k1}, \cdots, c_{kN_k}\}$ in memory by an arithmetic coder [18]. The lists at different nodes are separately encoded to allow random access during a query. Coding results in Sec. 4 will show that arithmetic coding enables the inverted index to be compressed nearly $5\times$. As database size increases, the list length N_k increases and fixed coding overheads are amortized over more encoded symbols, resulting in fewer bits spent per database image.

3.2. Fast Decoding

Keeping the decoding delay low is very important for a real-time image retrieval system as in [2]. To provide faster decoding, the carryover code was proposed in [13]. The carryover code fits as many data symbols as possible into a single 32-bit computer word. For each 32-bit word, there is a 2-bit selector and a 30-bit data portion. The selector indicates the precision (number of bits) needed to binary-encode each data symbol packed into the current word. A data symbol's precision is determined by truncating all leading zeros from its binary encoding. For example, if the selector indicates that 30 consecutive 1-bit data symbols are packed into the data portion, then all 30 data symbols are 0 or 1. If the selector indicates that 15 consecutive 2-bit data symbols follow, then all 15 data symbols are 0, 1, 2, or 3. One word's selector is conditionally coded from the previous word's selector to exploit correlation between words. Unlike the bitstream of an arithmetic coder, a carryover coder respects 32-bit computer word boundaries and provides regular memory access patterns.

Similar to the carryover code, the Recursive Bottom Up Complete (RBUC) code [14] also accesses memory efficiently during decoding. First, an even-length sequence $X = \{x_1, x_2, \cdots, x_M\}$ is grouped into symbol pairs $X' = \{(x_1, x_2), (x_3, x_4), \cdots, (x_{M-1}, x_M)\}$. Second, the precision needed for binary encoding of each data symbol is calculated in the same manner as in the carryover code. Denote these precisions $P = \{p_1, p_2, \cdots, p_M\}$. The precision for each pair of

data symbols is defined as the max of their two precisions, giving a sequence

$$P' = \left\{ p'_1 = \max(p_1, p_2), p'_2 = \max(p_3, p_4), \cdots, p'_{M/2} = \max(p_{M-1}, p_M) \right\} \quad , \tag{5}$$

where the lengths of the sequences are related by $|P'| = |X|/2 = |P|/2$. Now, the data symbols in X are binary-encoded according to the precisions indicated in P', so that given P' and the binary encodings, X can be losslessly reconstructed. The final task is to encode P', which is accomplished by treating P' as a new data sequence and recursively coding it. Since $|P'| = |X|/2$, the RBUC recursion is only repeated $\log_2(|X|)$ times. The use of selectors again regularizes memory access patterns, so RBUC decoding is much faster than arithmetic decoding. As will be shown in Sec. 4, the carryover and RBUC codes are competitive with arithmetic coding for inverted index compression but significantly reduce the decoding time.

3.3. Image ID Reordering

An interesting phenomenon can be discovered by examining Fig. 2(a). The empirical distribution for the image ID differences has much higher probabilities for small ID difference values than does the single geometric distribution. This suggests that feature descriptors are correlated between images, since otherwise the single geometric distribution would be empirically observed. For text retrieval, correlations between database documents have also been observed and exploited for improved compression [12] [19] [20]. To the best of our knowledge, we are the first to report this phenomenon in image databases and exploit it for inverted index coding.

When we assembled the image database, we coincidentally grouped similar images together. Although this "natural" order is already favorable to coding, we want to further reorder the database images into a new order that maximizes compression efficiency. Intuitively, the goal should be to minimize the image ID differences, averaged over the lists at all leaf nodes. Similar to [20], we pose the following optimization problem:

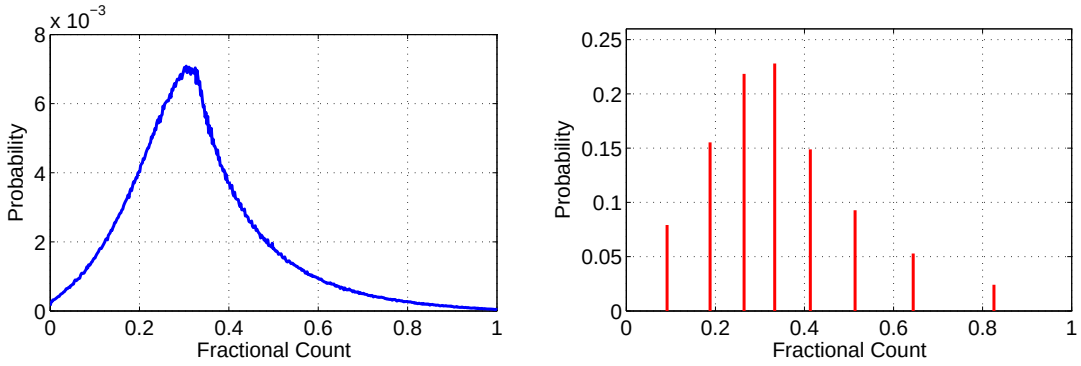
$$\min_{\pi} \quad \sum_{k=1}^{K} p_k \sum_{j=1}^{N_k} \left(\pi\left(i_{kj}\right) - \pi\left(i_{k(j-1)}\right) \right) \quad , \tag{6}$$

where π is a permutation of the image IDs, p_k is the probability that a feature descriptor is classified to the k^{th} leaf node, K is the number of leaf nodes, N_k is the length of the inverted list at the k^{th} leaf node, and i_{kj} is the j^{th} image ID in that inverted list. Let us define $\pi(i_{k0}) = 0$ for all k, so that $\pi(i_{k1}) - \pi(i_{k0}) = \pi(i_{k1})$. It has been shown that the problem in Eq. (6) can be cast as a traveling salesman problem (TSP), where the cities are documents in the database and the inter-city distances are the correlations between documents [19] [20]. The problem is thus NP-hard. We use the greedy heuristic proposed in [20]: 1) randomly visit an image i_1^*, 2) visit the image i_2^* whose features are most similar to those of i_1^*, and 3) visit the image i_3^* whose features are most similar to those of i_2^*, and so on, and 4) choose the permutation π^* which solves the problem in Eq. (6) as $i_1^* \rightarrow 1, i_2^* \rightarrow 2, \cdots, i_N^* \rightarrow N$. This reordering will enable us to reduce the database memory by an additional 37 percent compared to the natural order, as shown in Sec. 4. The image ID reordering is done offline and does not increase the query latency.

3.4. Soft-Binned Tree Search

Classification of feature descriptors through a VT is performed with a greedy search that explores multiple tree branches before deciding on the nearest tree node [21]. The authors of [10] took the idea one step further by classifying a feature descriptor to the k nearest tree nodes. Their soft-binned tree search gives noticeable improvement in classification accuracy over the

previous hard-binned approach. Unfortunately, because each database feature now appears in k different inverted lists, the inverted index is about k times as large. For the value $k = 3$ recommended in [10] and a database with one million images, the memory expands to 12 GB without compression, an even more significant burden on a server than before.



(a) (b)

Figure 3. Probability distributions for (a) Unquantized fractional counts from soft binning and (b) Lloyd-Max quantized fractional counts.

We generalize the coding method that we developed in Sec. 3.1 to encode a soft-binned inverted index. Image IDs can still be encoded by taking differences followed by entropy coding. The new challenge is that the counts are no longer integers but rather fractional values. In [10], the k closest tree nodes for a feature descriptor are assigned fractional counts

$$c_i = 1/C \cdot \exp\left(-0.5\, d_i^2/\sigma^2\right), \quad i = 1, \cdots, k \tag{7}$$

$$C = \sum_{i=1}^{k} \exp\left(-0.5\, d_i^2/\sigma^2\right) \quad , \tag{8}$$

where d_i is the Euclidean distance between the i^{th} closest tree node centroid and the feature descriptor, and σ is appropriately chosen to maximize classification accuracy. For the SURF descriptor [6], $\sigma \in [0.3, 0.4]$ was found to be a good range. Fig. 3(a) shows the probability distribution of fractional counts, for our database of one million CD/DVD/book cover images and $k = 3$. These fractional counts are quantized to provide a compact representation in the inverted index. Eight centroids are chosen by the Lloyd-Max algorithm, as shown in Fig. 3(b), so each centroid can be indexed by 3 bits with fixed-length coding and even fewer bits with variable-length coding. The quantization has negligible impact on classification accuracy and enables the soft-binned inverted index to be compressed $6\times$, as shown in the next section.

4. Experimental Results

Our index coding methods are tested on a database of one million CD/DVD/book cover images. Each image has 500×500 pixels resolution. We extract SURF features [6] for all images. A vocabulary tree with $K = 10^6$ leaf nodes is built from randomly chosen database descriptors, and then the corresponding inverted index is constructed. To test image matching performance, we use a set of 1000 query images exhibiting challenging photometric and geometric distortions [22]. In Sec. 4.1, we present the basic coding results. Then in Sec. 4.2-4.3, we show how index coding enables accuracy-boosting extensions, like soft-binned descriptor quantization and multi-view trees, to be applied with much lower memory requirements.

4.1. Coding Efficiency and Search Latency

To evalulate the efficiency of the coding methods presented in Sec. 3.1-3.2 at different database sizes, subsets of the one million images are seleced and an inverted index is constructed for each subset. Fig. 4(a) plots the memory usage of the inverted index versus database size. Without compression, memory usage grows rapidly and reaches 2.5 GB at one million images. Carryover, RBUC, and arithmetic coding all enable significant reduction in memory usage, by as much as $5\times$. The compression gain increases somewhat with database size. Compression performance of all three methods is close, with arithmetic coding performing the best.

The advantage of the carryover and RBUC codes becomes evident in Fig. 4(b), which plots decoding time versus database size. Although the arithmetic code has the highest coding efficiency, its decoding time is much longer than that of the carryover or RBUC code. In a real-time image retrieval system, the carryover and RBUC codes would be preferred for large databases.

In Sec. 3.3, we described how reordering the database images can lead to further gains in compression. Fig. 4(c) compares the memory usage after index compression for two orders: 1) the natural order in which we assembled the database, with coincidental but still suboptimal grouping of similar images, and 2) the optimal order calculated by the permutation algorithm of Sec. 3.3. Image ID reordering yields additional memory savings, as much as 37 percent. The reordering does not change the query time, as the permutation is computed offline and stored for later use.

Another benefit of index coding is that it avoids long latencies caused by memory congestion. We performed an experiment in which two retrieval systems run on the same server. The systems respectively serve two image databases of the same size, and their two inverted indices share the same memory. Also, 1000 query images are processed through the two systems at the same time. Since the server has four processors, the main resource conflict is limited memory. Fig. 4(d) plots the average search time with and without index coding at different database sizes. For small databases, the two uncompressed indices fit together in main memory, so swapping between main and virtual memory is avoided. Beyond 700k images, however, both uncompressed indices cannot coexist in main memory, so memory swapping is triggered, which significantly slows down the retrieval systems. This problem is avoided by index coding, which allows both compressed indices to fit in main memory even for the larger database sizes.

4.2. Improved Matching from Soft-Binned Tree Search

A comparison of the image recognition accuracy for soft-binned and hard-binned tree search is given in Fig. 5(a). Especially for large databases, soft-binned descriptor quantization provides a noticeable increase in accuracy. The main cost of soft binning is the increase in inverted index memory, which we address with the generalized coding scheme presented in Sec. 3.4. Memory usage before and after index compression is plotted in Fig. 5(b). After coding, memory usage is reduced as much as $6\times$; at one million images, memory drops from 12 GB to just 2 GB. Notice in Fig. 5(a) that the compressed and uncompressed soft-binned indices give very similar recognition performance, so the memory savings do not come at the expense of accuracy. Since Lloyd-Max quantization of fractional counts causes nearby database and query counts to fall into the same quantization bin, the correctly matching database and query tree histograms become more similar after quantization. Thus, the compressed index performs slightly better than the uncompressed index.

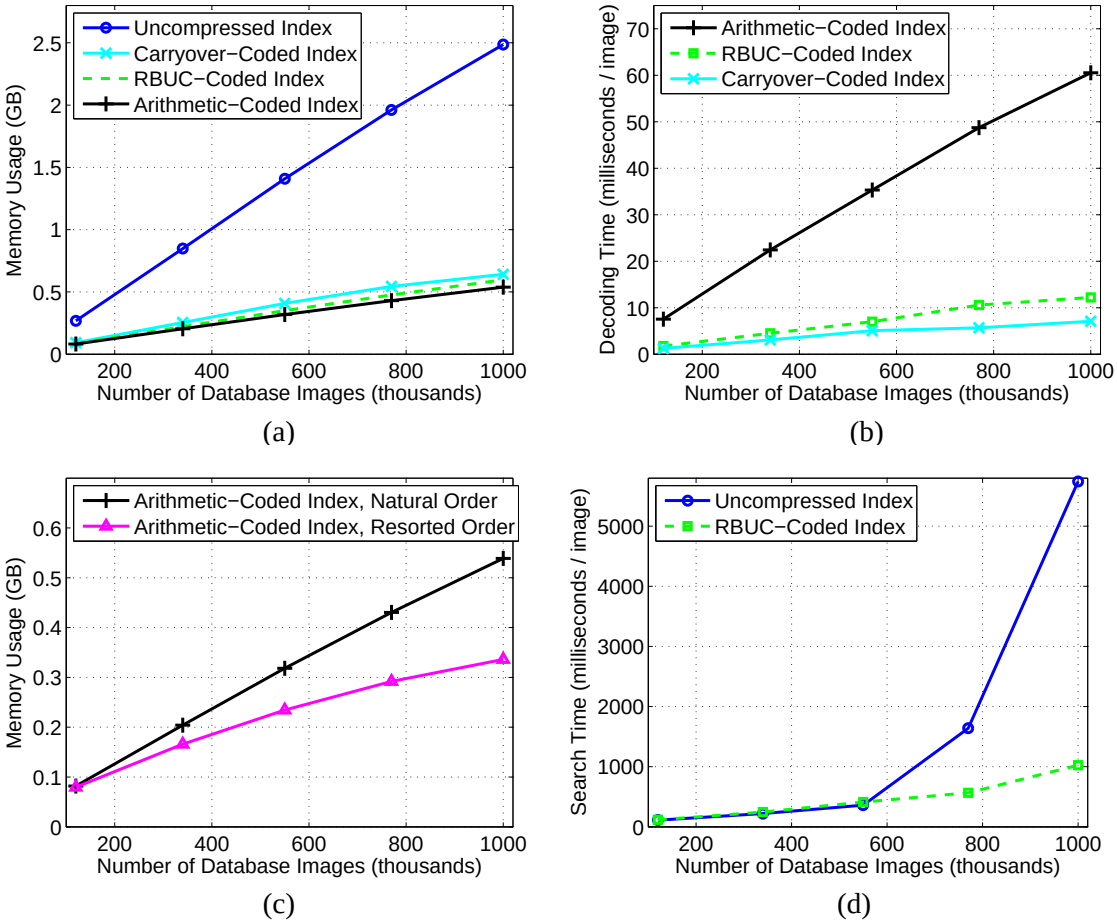
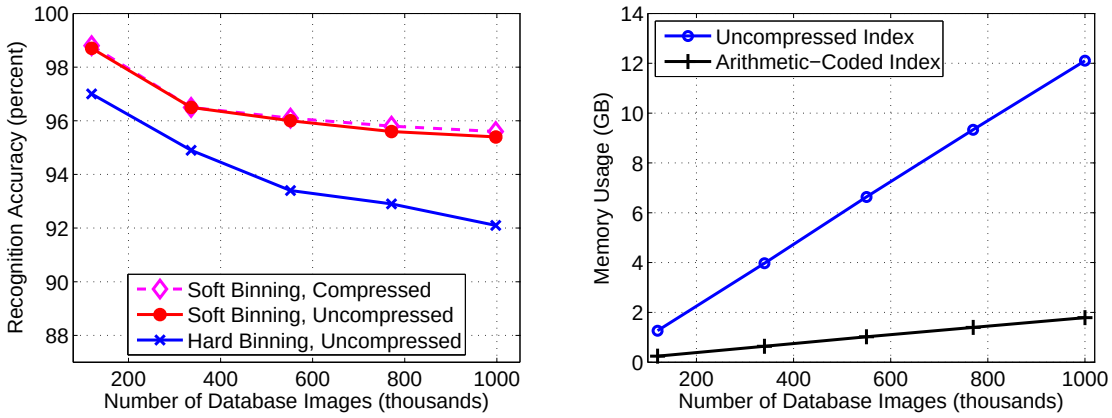


Figure 4. (a) Memory usage for uncompressed and compressed inverted indices. (b) Decoding latencies with different codecs. (c) Memory usage before and after image ID reordering. (d) Search latencies with and without index compression, for two databases of the same given size running simultaneously.

4.3. Improved Matching from Multi-View Trees

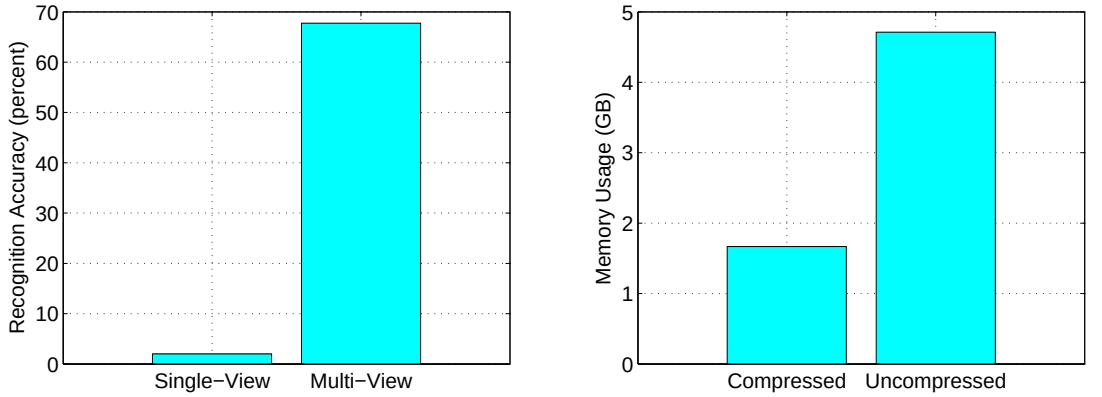
In [23], we demonstrated that multi-view trees for different views gives substantially more robustness against perspective distortions compared to single-view trees. Fronto-parallel database images are warped into four new perspective views, and a separate VT and inverted index is built for each of the five views (front and four perspective views). For a query image, this forest of five trees is searched in parallel, and the best matching database image out of the five trees is reported. Fig. 6 compares the recognition accuracy using the multi-view trees versus versus using a single VT built from fronto-parallel database images, for a database of 670k CD/DVD/book cover images and 400 query images showing perspective distortion [24]. Whereas the single fronto-parallel VT fails to recognize almost all the query images, multi-view trees perform reasonably well.

Like soft-binned tree search, the main cost of multi-view trees is an increase in memory. We again apply index coding to alleviate the memory burdens. Fig. 6(b) plots the memory for the five trees' inverted indices before and after compression. The large memory footprint before coding becomes a manageable amount afterwards. Also, index coding reduces memory conflicts between the five simultaneously active inverted indices in the multi-view group.



(a) (b)

Figure 5. (a) Image matching accuracies for soft-binned and hard-binned indices. (b) Memory usage of soft-binned index before and after compression.



(a) (b)

Figure 6. (a) Image matching accuracies for single-view tree and multi-view trees. (b) Memory usage for inverted indices of multi-view trees before and after compression.

5. Conclusion

Large memory usage by an inverted index hinders image database scalability and slows down processes on a memory-congested server. We have presented a compressed inverted index that significantly reduces memory usage, as much as $5\times$, without any effect on classification accuracy. Arithmetic coding gives the best coding performance, but two alternative codes provide similar coding gains with much faster decoding. An efficient offline database reordering results in an additional 37 percent memory savings. We showed how index coding averts long delays caused by memory swapping and enables two memory-intensive, accuracy-boosting techniques, soft-binned tree search and multi-view trees, to be more easily implemented. Future work should examine other image matching techniques which may now be feasible after the development of the compressed inverted index.

References

[1] G. Takacs, V. Chandrasekhar, N. Gelfand, Y. Xiong, W.-C. Chen, T. Bismpigiannis, R. Grzeszczuk, K. Pulli, and B. Girod, "Outdoors augmented reality on mobile phone using loxel-based visual feature organization," in *ACM Multimedia Information Retrieval*, 2008, pp. 427–434.

[2] S. Tsai, D. Chen, J. Singh, and B. Girod, "Image-based retrieval with a camera-phone," in *IEEE Internal Conference on Acoustics, Speech, and Signal Processing*, 2009, technical demo. [Online]. Available: http://www.youtube.com/user/ivmscibrvideos

[3] Kooaba, "Product logo recognition," http://www.kooaba.com/technology/labs.

[4] SnapTell, "Media jacket recognition," http://www.snaptell.com/demos/DemoLarge.htm.

[5] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, November 2004.

[6] H. Bay, T. Tuytelaars, and L. J. V. Gool, "SURF: Speeded up robust features," in *European Conference on Computer Vision*, 2006, pp. I: 404–417.

[7] V. Chandrasekhar, G. Takacs, D. Chen, S. Tsai, R. Grzeszczuk, and B. Girod, "CHoG: compressed histogram of gradients," in *IEEE Computer Vision and Pattern Recognition*, 2009, pp. 1–8.

[8] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," in *IEEE Computer Vision and Pattern Recognition*, 2006, pp. II: 2161–2168.

[9] H. Jegou, H. Harzallah, and C. Schmid, "A contextual dissimilarity measure for accurate and efficient image search," in *IEEE Computer Vision and Pattern Recognition*, 2007, pp. 1–8.

[10] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Lost in quantization: Improving particular object retrieval in large scale image databases," in *IEEE Computer Vision and Pattern Recognition*, 2008, pp. 1–8.

[11] A. Moffat and L. Stuiver, "Exploiting clustering in inverted file compression," in *IEEE Data Compression Conference*, 1996, pp. 82–93.

[12] D. Blandford and G. Blelloch, "Index compression through document reordering," in *IEEE Data Compression Conference*, 2002, pp. 342–351.

[13] V. N. Anh and A. Moffat, "Inverted index compression using word-aligned binary codes," *Information Retrieval*, vol. 8, no. 1, pp. 151–166, January 2005.

[14] A. Moffat and V. N. Anh, "Binary codes for non-uniform sources," in *IEEE Data Compression Conference*, 2005, pp. 133–142.

[15] J. Zobel and A. Moffat, "Inverted files for text search engines," *ACM Computing Surveys*, vol. 38, no. 2, p. 6, July 2006.

[16] H. Jegou, M. Douze, and C. Schmid, "Packing bag-of-features," in *IEEE International Conference on Computer Vision*, 2009, pp. 1–8.

[17] K. W. Church and W. A. Gale, "Poisson mixtures," *Natural Language Engineering*, vol. 1, pp. 163–190, 1995.

[18] A. Said, "Comparative analysis of arithmetic coding computational complexity," in *HP Labs Technical Report*, 2004.

[19] A. Gelbukh, S. Han, and G. Sidorov, "Compression of boolean inverted files by document reordering," in *IEEE Conference on Natural Language Processing and Knowledge Engineering*, 2003, pp. 244–249.

[20] C.-S. Cheng, C.-P. Chung, and J. J.-J. Shann, "Fast query evaluation through document identifier assignment for inverted file-based information retrieval systems," *Information Processing and Management*, vol. 42, no. 3, pp. 729–750, May 2006.

[21] G. Schindler, M. Brown, and R. Szeliski, "City-scale location recognition," in *IEEE Computer Vision and Pattern Recognition*, 2007, pp. 1–7.

[22] D. Chen, S. Tsai, R. Vedantham, R. Grzeszczuk, and B. Girod, *CD/DVD Query Images*, April 2009. [Online]. Available: http://vcui2.nokiapaloalto.com/∼dchen/cibr/testimages/

[23] D. Chen, S. Tsai, V. Chandrasekhar, G. Takacs, J. Singh, and B. Girod, "Robust image retrieval with multiview scalable vocabulary trees," in *SPIE Visual Communications and Image Processing*, 2009, p. 72570V.

[24] D. Chen, S. Tsai, R. Vedantham, R. Grzeszczuk, and B. Girod, *Multiview CD/DVD Query Images*, April 2009. [Online]. Available: http://vcui2.nokiapaloalto.com/∼dchen/cibr/testimagesmultiview/