Joint Inverted Indexing

Yan Xia[1]* Kaiming He[2] Fang Wen[2] Jian Sun[2]
[1]University of Science and Technology of China [2]Microsoft Research Asia

Abstract

Inverted indexing is a popular non-exhaustive solution to large scale search. An inverted file is built by a quantizer such as k-means or a tree structure. It has been found that multiple inverted files, obtained by multiple independent random quantizers, are able to achieve practically good recall and speed.

Instead of computing the multiple quantizers independently, we present a method that creates them jointly. Our method jointly optimizes all codewords in all quantizers. Then it assigns these codewords to the quantizers. In experiments this method shows significant improvement over various existing methods that use multiple independent quantizers. On the one-billion set of SIFT vectors, our method is faster and more accurate than a recent state-of-the-art inverted indexing method.

1. Introduction

Inverted indexing [32] is of central importance in modern search engines for both text retrieval [4] and image/video retrieval [29]. In text search engines like Google [5], a document is represented by a vector of weighted word frequencies. Inverted indexing is built as a vocabulary of words, and each word has a list of the documents that contain this word. Online, the search engine retrieves the lists corresponding to the query words and ranks the returned documents. This is a non-exhaustive solution because only a very small portion of documents are checked. Inverted indexing provides a practical solution to large scale problems.

The above approach has been adapted to image/video retrieval by introducing "visual words" [29]. The visual words (codewords) are obtained by quantizing visual vectors, *e.g.*, via k-means [29], k-means trees [24], or other tree structures [11, 10, 20]. The inverted indexing is built as a codebook of the codewords. Each codeword has a list of all vectors (or their IDs) belonging to this codeword. Given a query vector, the system finds the codeword of the query and checks those items in the list of this codeword. The checking can be distance computation (using raw vectors, binary embedding [17], or product-quantized codes [19]) or with geometric considerations [17, 33].

An effective way to improve the recall of inverted indexing methods is to use multiple quantizers (also known as multiple hash tables [8, 26]). Multiple quantizers introduce redundant coverage among the lists. A query is compared to the items in multiple (overlapping) lists to reduce the chance of missing true neighbors. The Locality Sensitive Hashing (LSH) [16, 8, 2] is a widely used multiple-quantizer method[1]. In LSH each quantizer (hash table) is binning a low dimensional random subspace. In [28, 23] multiple randomized k-d trees are used and have shown advantages over LSH. In [26] Paulevé *et al*. compare the performance of lattices, trees, and k-means as the hash tables. They recommend using randomly re-initialized k-means as the multiple quantizers. This method is named as "k-means LSH" or KLSH [26].

In the spirit of hash, all the above multi-quantizer methods create the quantizers independently and randomly. The KLSH method [26] enjoys an advantage that its individual quantizer is (locally) optimal in the sense of minimizing quantization error [14]. But we point out that for KLSH, the codewords from different codebooks tend to be similar. This is because the k-means algorithm tends to put the codewords around densely distributed data even in different random trials. Actually, the codewords can be different from each other only because of the local optimality of the k-means nature. The similar codewords would reduce the redundant coverage and limit the gain of multiple quantizers.

To illustrate, we consider simple 1-d data subject to a two-peak distribution (Fig. 1). Suppose we want two quantizers each with two codewords. If each quantizer is constructed randomly and independently as in KLSH, the two codewords in a quantizer will always be placed near the two peaks. The space partitioning of the two quantizers are almost the same (Fig. 1 left), and the gain of having the second quantizer is lost. If a query is located very close to the partitioning boundary, the recall of retrieving the true k-NNs can be low.

*This work was done when Yan Xia was an intern at MSRA.

[1]LSH has also been used as compact binary encoding for distance ranking [30]. Unless specified, in this paper we refer to LSH as a multiple-quantizer method that uses inverted indexing, as it was described in [8, 2].

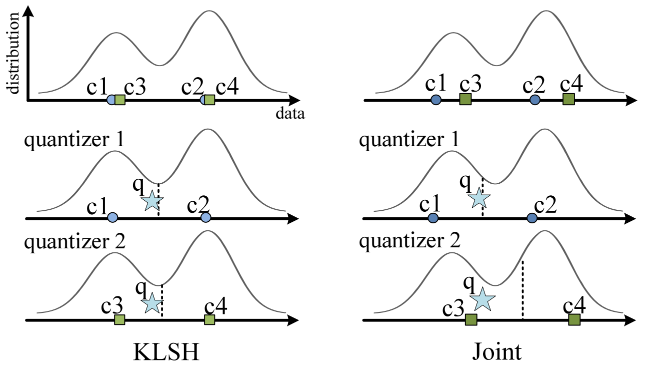

Figure 1. KLSH vs. joint quantizers. We illustrate two quantizers with two codewords in each quantizer. Top: the locations of the four codewords (c_1 to c_4). Middle and Bottom: quantizer 1 and quantizer 2, with the partitioning boundaries. When a query (q) is near any partitioning boundary, the independent quantizers may miss many true neighbors, while the joint quantizers can retrieve more true neighbors because the query is well inside at least one partition.

In the above example, we can instead generate all the four codewords *jointly* via a single pass of k-means clustering. These four codewords are distant from each other (Fig. 1 right). Then we construct each quantizer by mutual-exclusively selecting two codewords with some care, and the two resulting quantizers can be substantially different. Any query that lies near the partitioning boundary of one quantizer can still find its k-NNs through the other quantizer, and the recall is improved.

With this motivation, in this paper we propose *joint inverted indexing* - a novel multiple-quantizer method. Rather than construct quantizers independently, we create them jointly. We divide this challenging problem into two subproblems, each of which is formulated as optimization. We first jointly optimize all the codewords of all quantizers. The optimized codewords are then assigned to the quantizers, with a consideration that the total distortion of all quantizers is small.

Experiments on million/billion-scale datasets show that our method performs significantly better than a various multiple-quantizer methods like LSH [2], randomized trees [28, 23], and KLSH [26].

On a one-billion SIFT set, our method is faster and more accurate than a recent state-of-the-art method called "inverted multi-index" [3]. When searching in one billion items for the first nearest neighbor of a query, our method takes less than ten milliseconds using a single thread and achieves over 90% recall in the first retrieved one hundred items (detailed in Sec. 4 and Table 3). The superiorities of our method to [3] are at the cost of more memory consumption. It depends on applications whether this is desired. But we believe accuracy and speed are both of central importance in many practices, and our method can be favored in these cases.

2. Related Work

Recent progress on large scale search are mainly in two ways: exhaustive and non-exhaustive. One could efficiently exhaust millions of items using compact encoding like binary embedding (*e.g.*, [30, 31, 13, 15]) or product quantization [19, 12]. But for larger datasets, non-exhaustive search is still highly desired, and compact encoding is often used for re-ranking the retrieved data [17, 19, 3]. In this paper, we focus on non-exhaustive search based on inverted indexing.

Inverted indexing could be built using a quantizer like k-means [22], a binary tree [10, 7], hierarchical k-means [11, 24], or a lattice [1]. An index represents a cluster, a tree leaf, or a bin. The k-means quantizer is (locally) optimal in terms of quantization distortion [14].

It is an attractive way to use multiple quantizers to improve search accuracy. The LSH method [16, 6, 8] randomly generates L hash tables, each of which is binning a low dimensional subspace. For each single hash table, an inverted indexing system is built with each bin as an index, and each index has a list containing all the vectors in the bin. This is repeated for each of the L hashing tables. Given a query, the algorithm retrieves L lists and checks all the items in them.

Instead of binning, the methods in [28, 23] use randomized trees. Each single tree is generated by random projection. In [28] it is empirically observed that if the trees are independently rotated, the error rate is smaller when visiting shallow nodes in all trees than deeply backtracking a single one. In [23] this way is found superior to LSH.

The KLSH method [26] uses randomly re-initialized k-means to build each quantizer. The experiments in [26] (also in this paper) show this is superior to various alternatives including lattices and trees. But as discussed in the introduction, the codewords in the multiple quantizers can be similar in random trials.

Most recently, an *inverted multi-index* method [3] has been proposed. This is actually a *single* quantizer method, whose quantizer is the Cartesian product of multiple sub-quantizers. As such, each codeword is indexed by the Cartesian product of multiple sub-indices. The quantization of the space can be much finer (*e.g.*, than classic k-means). The finer quantization, in conjunction with a soft-assignment scheme, has shown great superiority to a single k-means quantizer [3]. This is a methodology different from multiple quantizers. We will compare with this method experimentally.

3. Joint Inverted Indexing

We first introduce the formulation of the method, and then present the algorithm.

3.1. Formulation

Denote the number of quantizers as L, and the number of codewords in each quantizer as K. We assume K and L are pre-defined and application-based (*e.g.*, in [26] K=2^n and L=16).

Background It is well-known that a k-means quantizer attempts to minimize the *quantization distortion* [14]:

$$\min_{\{\mathbf{c}_k\}} \sum_{k=1}^{K} \sum_{\mathbf{x}\in\mathcal{C}_k} d(\mathbf{x}, \mathbf{c}_k), \tag{1}$$
$$\text{with:} \quad \mathcal{C}_k = \{\mathbf{x} \mid d(\mathbf{x}, \mathbf{c}_k) < d(\mathbf{x}, \mathbf{c}_{k'}), \quad \forall k' \neq k\}.$$

Here $\mathbf{x}$ belongs to the training dataset, $\mathbf{c}_k$ is a codeword, $\mathcal{C}_k$ is a cluster, and $d(\cdot,\cdot)$ is the squared Euclidean distance between two vectors. The optimization is w.r.t. the set of codewords $\{\mathbf{c}_k\}$. The distortion in (1) can be optimized via the classical EM-alike algorithm: alternatively update the codewords $\{\mathbf{c}_k\}$ and assign data to the clusters $\{\mathcal{C}_k\}$.

For L independent k-means quantizers as in KLSH [26], we can formulate them as minimizing the total distortion of all quantizers:

$$\min_{\{\mathbf{c}_k^l\}} \sum_{l=1}^{L} \left(\sum_{k=1}^{K} \sum_{\mathbf{x}\in\mathcal{C}_k^l} d(\mathbf{x}, \mathbf{c}_k^l) \right), \tag{2}$$
$$\text{with:} \quad \mathcal{C}_k^l = \left\{\mathbf{x} \mid d(\mathbf{x}, \mathbf{c}_k^l) < d(\mathbf{x}, \mathbf{c}_{k'}^l), \quad \forall k' \neq k\right\}.$$

Here we use $\mathcal{C}_k^l$ and $\mathbf{c}_k^l$ to denote the l-th quantizer's cluster and codeword. If all the codewords $\{\mathbf{c}_k^l \mid 1 \leq k \leq K,\ 1 \leq l \leq L\}$ are initialized randomly, minimizing (2) is equivalent to independently minimizing L separate problems as in (1). This is the way of KLSH. If it were not for the local optimality of k-means clustering, all the quantizers should be identical.

Joint Codewords Optimization In (2) the codewords in one quantizer are unaware of those in other quantizers. So the independent optimization of any two quantizers may push the codewords towards similar positions due to the k-means nature. To overcome this issue, we propose to optimize all the codewords jointly.

We notice that in multiple inverted files, it is sufficient for a true datum to be correctly retrieved if it is covered by *one* of the L lists. Motivated by this, it is reasonable for us to consider the smallest distortion of a datum $\mathbf{x}$ among all L quantizers. Formally, we consider:

$$\min_{l} \left(\min_{k} d(\mathbf{x}, \mathbf{c}_k^l) \right), \tag{3}$$

or equivalently $\min_{(l,k)} d(\mathbf{x}, \mathbf{c}_k^l)$, as the smallest distortion of $\mathbf{x}$. To jointly optimize all codewords of all quantizers, we propose to minimize the sum of the smallest distortion of all data: $\sum_{\mathbf{x}} \left(\min_{(l,k)} d(\mathbf{x}, \mathbf{c}_k^l) \right)$. It is easy to show that this problem is equivalent to optimizing:

$$\min_{\{\mathbf{c}_k^l\}} \sum_{l=1}^{L} \sum_{k=1}^{K} \left(\sum_{\mathbf{x}\in\mathcal{S}_{(l,k)}} d(\mathbf{x}, \mathbf{c}_k^l) \right), \tag{4}$$
$$\text{with:}$$
$$\mathcal{S}_{(l,k)} = \left\{\mathbf{x} \mid d(\mathbf{x}, \mathbf{c}_k^l) < d(\mathbf{x}, \mathbf{c}_{k'}^{l'}), \ \forall (l',k') \neq (l,k)\right\}.$$

Unlike in (2) where each $\mathbf{x}$ has been counted L times, in (4) each $\mathbf{x}$ has been counted only once. Replacing (l,k) by a one-to-one mapping to an integer j: $\mathcal{M}(l,k) \mapsto j \in [1, L \times K]$, we can rewrite (4) as this optimization problem:

$$\min_{\{\mathbf{c}_j\}} \sum_{j=1}^{L\times K} \left(\sum_{\mathbf{x}\in\mathcal{S}_j} d(\mathbf{x}, \mathbf{c}_j) \right) \tag{5}$$
$$\text{with:} \quad \mathcal{S}_j = \{\mathbf{x} \mid d(\mathbf{x}, \mathbf{c}_j) < d(\mathbf{x}, \mathbf{c}_{j'}), \quad \forall j' \neq j\}.$$

Comparing (5) to (1), one can find the problem (5) is a *single* pass of k-means with $L \times K$ clusters, so can be easily optimized. We denote the optimized codewords as $\mathbb{C} = \{\mathbf{c}_j \mid 1 \leq j \leq L \times K\}$.

The above joint codewords optimization has not yet considered the belonging of any codeword to the quantizers. A naive way is to *randomly assign* the codewords in $\mathbb{C}$ without replacement[2] to the L quantizers. We call this way as "Joint (rand-assign)". In experiments, this simple way has already shown better search performance than KLSH (detailed in Sec. 4, Fig. 6). This implies that the joint codewords optimization is an effective method for multiple quantizers. This is due to two nice properties of joint codewords optimization: (i) each $\mathbf{x}$ is expected to be accurately quantized at least once, in the sense of optimizing (4); (ii) the codewords among the quantizers are guaranteed to be sufficiently different from each other.

Joint Codewords Assignment Suppose the set $\mathbb{C}$ has been fixed, *i.e.*, the problem in (4) (or equivalently (5)) has been optimized. It is still possible for us to improve the performance of "Joint (rand-assign)" by a better assignment.

Notice that the quantization distortion of any individual quantizer has not been considered in (4). For a better assignment, we consider the total distortion of all quantizers subject to a constraint. Formally, we consider:

$$\min \sum_{l=1}^{L} \left(\sum_{k=1}^{K} \sum_{\mathbf{x}\in\mathcal{C}_k^l} d(\mathbf{x}, \mathbf{c}_k^l) \right), \tag{6}$$
$$\text{with:} \quad \mathcal{C}_k^l = \left\{\mathbf{x} \mid d(\mathbf{x}, \mathbf{c}_k^l) < d(\mathbf{x}, \mathbf{c}_{k'}^l), \quad \forall k' \neq k\right\}$$
$$\text{s.t.} \quad \mathbf{c}_k^l \in \mathbb{C}, \quad \text{and} \quad \mathbf{c}_k^l \neq \mathbf{c}_{k'}^{l'} \quad \forall (l',k') \neq (l,k). \tag{7}$$

[2]The term "*without replacement*" indicates each sample can only be selected once.

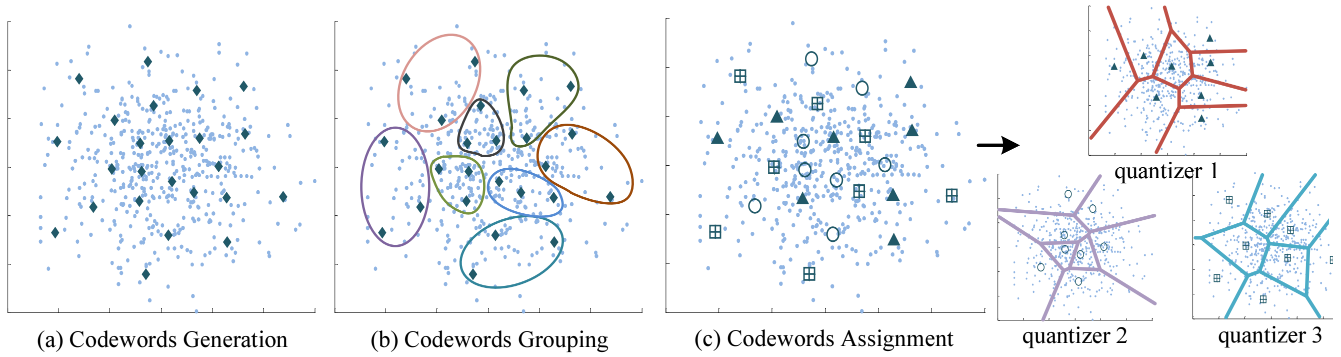

Figure 2. Joint Inverted Indexing. We illustrate L=3 quantizers with K=8 codewords in each quantizer. Each dot represents a data point. (a) **Codewords Generation**. All $L \times K$ codewords in all quantizers are generated simultaneously by k-means clustering. Each ♦ represents a codeword. (b) **Codewords Grouping**. The codewords are grouped into K balanced groups. (c) **Codewords Assignment**. A quantizer is constructed by randomly selecting one codeword from each group without replacement. The codewords in quantizer 1, 2, 3 are represented by ▲, ◯, ⊞ respectively. Each quantizer is shown on the right.

The objective function in (6) is the total distortion of all quantizers as in (2). The constraint in (7) means that the codewords must be mutual-exclusively selected from the fixed $\mathbb{C}$. As such, the codewords must be subject to the jointly optimized codewords in (4).

The constraint (7) leads to a fundamentally different optimization problem in (6). Because the set $\mathbb{C}$ has been fixed, the codewords cannot be arbitrarily updated (*e.g.*, via averaging in-cluster data). In fact, in (6) we can only optimize w.r.t. the one-to-one mapping $\mathcal{M}(l,k) \mapsto j$. This is a combinatorial problem and an (even locally) optimal solution can only be approximately achieved.

3.2. Algorithm

Based on the above formulations, we present an algorithm to jointly generate the quantizers. It has three steps:

Step 1 - Codewords Generation. $L \times K$ codewords are generated by a single pass of k-means (Fig. 2(a)). This solves the optimization problem in (5) and gives the set $\mathbb{C}$.

Step 2 - Codewords Grouping. We divide the $L \times K$ codewords into K groups (each group contains L codewords) (Fig. 2(b)). This is achieved by a balanced clustering algorithm like a binary tree. We group the codewords by a random projection tree [9].

Step 3 - Codewords Assignment. A K-codeword quantizer is generated by randomly selecting one codeword from each of the K groups without replacement. We repeat this for L times and construct all the L quantizers (Fig. 2(c)).

Discussions:

Intuitively, to make a quantizer with small distortion, we expect the K codewords of this quantizer to be dispersed. So we group the codewords by a second pass of clustering (in Step 2, by a tree), and randomly select one codeword from each group to form a quantizer (in Step 3). As such, we can avoid the codewords in any quantizer from being too concentrated. We can expect such a quantizer to have smaller distortion than a quantizer that randomly selects from the whole set $\mathbb{C}$ (as in "Joint (rand-assign)"). We denote the way in Step 1-3 as "Joint", and quantitatively compare the total distortion of "Joint (rand-assign)" and "Joint" on a SIFT1M dataset [19] in Table 1. We see that when subject to the same constraint, the "Joint" way has smaller total distortion. Table 1 also gives the standard deviations (*std*) of total distortion in 10 random trails for both methods (all trails subject to the same constraint). We see that it is very hard for different random trails to substantially impact the distortion. We have also tried to optimize the total distortion by greedily swapping pairs of codewords between quantizers. But we find even tens of thousands of swaps can only lead to ignorable reduction to the total distortion ($\ll$ *std*). This is perhaps due to the combinatorial nature of the problem. So we does not apply this greedy way.

	K=2^9	K=2^{12}
Joint (rand-assign)	9.612 ± 0.006	8.225 ± 0.002
Joint	9.466 ± 0.005	8.111 ± 0.001

Table 1. The total distortion of "Joint (rand-assign)" and "Joint" in SIFT1M. We use L=16 quantizers. For each K, the distortion is averaged on 10 random trails subject to the same fixed constraints. The standard deviation is also evaluated. All numbers in the table have a unit of $\times 10^{11}$.

Visualization:

We visualize the behaviors of KLSH [26] and our method by a toy example. In Fig. 3 we randomly generate 500 data points subject to a 2-d Gaussian Mixture Model (GMM). For this database we build 3 quantizers with 8 codewords in each quantizer ($L = 3$, $K = 8$).

We can see that in KLSH some codewords of different quantizers are very similar. In Fig. 4(a) we show the be-

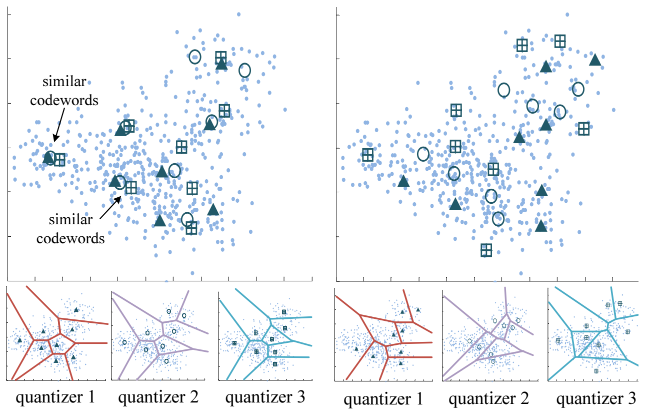

Figure 3. A toy data of 2-d GMM distribution. (a) Quantizers of KLSH. (b) Our quantizers. Top: the locations of all codewords in all quantizers. Bottom: the space partitions by individual quantizers. Each dot represents a data point. The codewords in quantizer 1, 2, 3 are represented by ▲, ◯, ⊞ respectively.

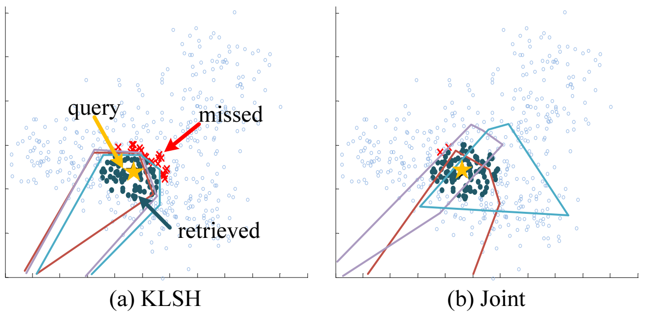

Figure 4. Visualization of querying. (a) KLSH, (b) our method. Given a query (★), the cell where the query lies in each quantizer is shown by the partitioning boundary. A dark dot ● represents a successfully retrieved nearest neighbor, and a red cross × represents a missed neighbor.

havior of these quantizers to a typical query point: due to the similar codewords, the gain of the multiple quantizers is limited. Many true k-NNs of this query are not retrieved.

In contrast, all the codewords are jointly optimized by our method (Fig. 3(b)). Given the query point, the clusters can cover more potential candidates (Fig. 4(b)) and retrieve more true k-NNs.

4. Experiments

We evaluate on three public datasets from [19, 18]: (i) SIFT1M with one million 128-d SIFT vectors [21]; (ii) GIST1M with one million 960-d GIST vectors [25]; and (iii) SIFT1B with one billion SIFT. All datasets have provided independent queries for evaluation (10,000 in the two SIFT sets and 1,000 in the GIST set).

All on-line experiments are conducted on a workstation with an Intel Xeon 2.67GHz CPU (using a single thread) and 96GB RAM. All algorithms are implemented in C++.

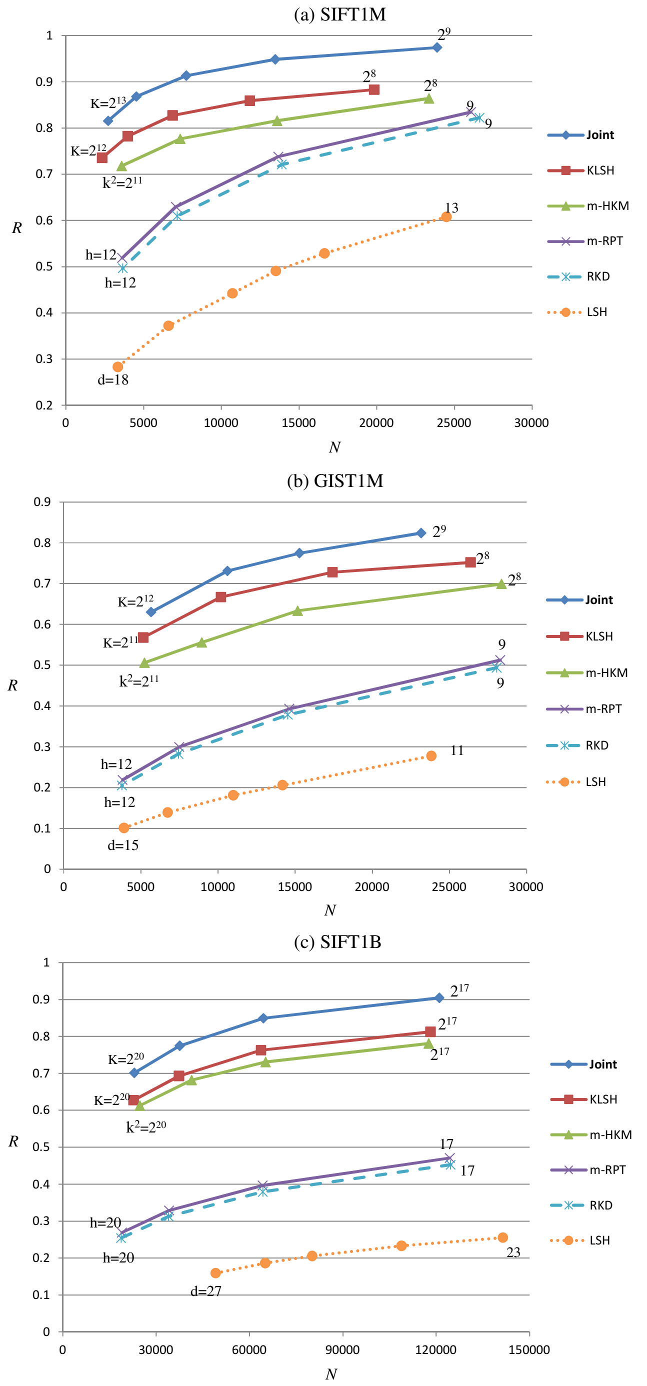

Figure 5. Comparisons with multiple-quantizer methods on (a) SIFT1M, (b) GIST1M, and (c) SIFT1B. All the methods are using L=16 quantizers. R and N are averaged over all queries. The parameter settings control the fineness of the quantization. For m-HKM, we use two levels of k-means with k centers each level. For m-RPT and RKD, we use trees of h levels. For LSH, we use d-dimensional subspaces [2]. The parameters are given near each marker. Note a straight line that links two markers is only for illustration - there is no value recorded on the line.

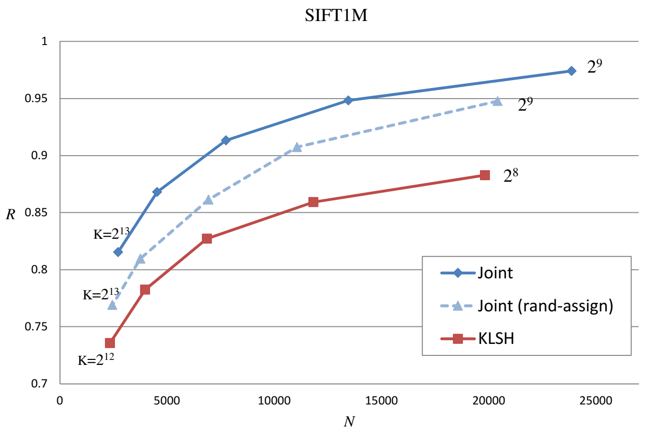

Figure 6. Comparisons among the Joint, Joint (rand-assign), and KLSH. We use L=16 here.

Comparisons with Multiple-Quantizer Methods We compare our method with the following multiple-quantizer methods: LSH [6], randomized k-d trees (RKD) [28], and KLSH [26]. We also compare with two alternatives: multiple hierarchical k-means (m-HKM) and multiple random projection trees (m-RPT). They are straightforward extensions of hierarchical k-means [23] and random projection trees [9] by independently and randomly generating multiple quantizers.

Both KLSH and our method should scan all codewords given a query. This can be time-consuming when the number of codewords is large, *e.g.*, in SIFT1B. To speedup, in this set for both KLSH and our method, we adopt a k-d tree [10] to approximately search for the codeword of a query. The performance on SIFT1B is reported based on this implementation.

In Fig. 5, we adopt the evaluation setting as in the KLSH paper [26]. We evaluate the recall R versus N, where N is the number of data retrieved by all L lists (*i.e.*, N is the size of the *union* set of the L lists [26]). The number N is considered because it is the number of data actually checked (*e.g.*, re-ranking or geometric consistency checking [17]). We treat the 100 nearest neighbors of each query as the ground truth. In Fig. 5 all methods use the same number of quantizers (L). Each method is evaluated using several parameter settings. The parameters control the fineness of the quantization (*e.g.*, the number of codewords, leaves, or bins). See the caption of Fig. 5 for the details of the parameters.

Fig. 5 shows that our joint quantizers outperform KLSH and other alternatives. Fig. 5 also shows that KLSH is better than the other competitors that use independent quantizers. This implies KLSH benefits from the k-means quantizers that give smaller distortion than the other independent quantizers.

In Fig. 6 we further compare with the "Joint (rand-assign)" way. As discussed in Sec. 3.2, "Joint (rand-assign)" is a way that has jointly optimized codewords but has larger total distortion than the "Joint" way (Table 1). Fig. 6 shows that even the "Joint (rand-assign)" way outperforms KLSH. This implies the jointly optimized codewords are very essential for good performance. Fig. 6 also shows that the "Joint" way is better than "Joint (rand-assign)". This implies that given the jointly optimized codewords, the search accuracy can be improved by reducing the total distortion.

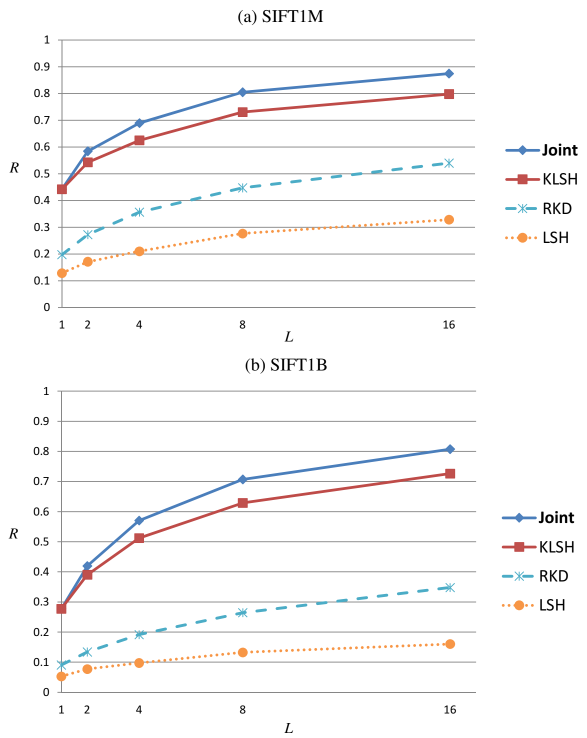

Figure 7. Recall vs. L. (a) SIFT1M (N = 5000). (b) SIFT1B (N = 10000). Because all methods studied cannot retrieve exactly N candidates, in this figure the recall is linearly interpolated from the recall obtained by the nearest two values of N.

	parameters	N	R	time (ms)
Joint	K=2^{18}	6.44×10^5	0.849	5.6
KLSH	K=2^{18}	6.37×10^5	0.763	5.6
RKD	h=18	6.42×10^5	0.379	3.4
LSH	d=26	6.50×10^5	0.186	3.4

Table 2. Retrieval time on SIFT1B. The parameters of each method are set such that $N \simeq 6\times10^5$. We use L=16 here.

In Fig. 7 we investigate the recall versus the number of quantizers (L) at a given number N. We see that our method consistently outperforms KLSH and other competitors for different L.

Table 2 shows the retrieval time per query in SIFT1B. The retrieval time is the time used to retrieve the L lists, and is without data re-ranking (the case with re-ranking will be

methods	parameters	R@100	R@300	R@1000	time(ms)	RAM (GB)
Multi-D-ADC	N=10000	$0.748_{(0.740)}$	0.749	0.751	$6.8_{(7)}$	20
Multi-D-ADC	N=30000	$0.885_{(0.885)}$	0.886	0.887	$16.9_{(16)}$	20
Multi-D-ADC	N=50000	0.929	0.932	0.934	27.9	20
KLSH-ADC	K=2^{19}	0.836	0.854	0.861	5.6	80
KLSH-ADC	K=2^{18}	0.860	0.882	0.889	7.8	80
KLSH-ADC	K=2^{17}	0.894	0.917	0.924	11.8	80
Joint-ADC	K=2^{19}	0.884	0.904	0.911	5.6	80
Joint-ADC	K=2^{18}	0.920	0.945	0.952	7.8	80
Joint-ADC	K=2^{17}	0.938	0.964	0.972	11.8	80

Table 3. Comparisons in SIFT1B. The ADC are all using 16 bytes per vector. For Multi-D-ADC, we use the second-order index with k=2^{14} as in [3]. The values in brackets are reported in [3], showing our re-implementation is very comparable to [3].

shown in Table 3). We set the parameters of each method such that the number N of each method is about the same. We see that both our method and KLSH slightly trade some retrieval time for substantially higher recall.

Comparisons with Inverted Multi-Index [3] We also compare with *inverted multi-index* [3], a recent state-of-the-art inverted indexing method. As discussed in Sec. 2, this method is actually a *single* quantizer method with finer quantization and soft-assignments. This is a different scheme with multiple-quantizer methods including KLSH and ours. We compare with this method by experiments, using a protocol as the one in [3].

We compare with the "Multi-D-ADC" version of [3] (its best performed version). We briefly introduce it as below. This method uses a soft-assignment way [27]. Due to its finer quantization, this method retrieves a large number (often thousands) of very short lists, until the total number of data in all lists reaches some given N. All the retrieved data in these lists are re-ranked, using the compact encoding method of product quantization (PQ) [19]. The Multi-D-ADC in [3] encodes the residual vector to a codeword and re-ranks the data using the asymmetric distance computation (ADC) [19]. The term '-D-ADC' implies residual encoding.

Because it can be more complicate to encode residual vectors in multiple quantizers (KLSH and ours), we simply use PQ to encode the original vector (it is observed that encoding the original vector is inferior [3]). Then we use ADC to re-rank the retrieved data. We term these methods as KLSH-ADC and Joint-ADC (ours).

Following [3], in this experiment we consider the first nearest neighbor of a query as the ground truth. The accuracy is measured by "R@n", that is, the recall at the top-n ranked data after re-ranking. In Table 3 we show the querying time and recall in SIFT1B. The querying time consists of both retrieval time and re-ranking time. Both KLSH and our method use L=16 quantizers here. Because both KLSH and our method cannot retrieve exactly N data, their performance is tuned by K.

Table 3 shows that our Joint-ADC outperforms Multi-D-ADC in both speed and accuracy. The Multi-D-ADC can improve its performance on "R@n" by increasing N. But because the short lists in its finer quantizer are "too short", this operation retrieves a very large number of lists and takes more time (cache-unfriendly). On the contrary, our method (and KLSH) retrieves a fixed number (L) of longer lists, and spends some more time on the ADC re-ranking. The systematic comparisons in Table 3 show that our method is a better practice.

The inverted multi-index is a soft-assignment method with a single but very fine quantizer. In [3] it has been shown that this method is faster and more accurate than other soft-assignment methods with a coarser quantizer, such as the IVFADC/IVFADC+R methods [18, 19]. So we omit the comparisons with these methods in this paper.

Discussions All the multiple-quantizer methods (including LSH, RKD, KLSH, and ours) require considerable memory consumption. Each single quantizer uses one inverted file that stores all the vector IDs. For a one-billion set, each ID takes 4 bytes ($\frac{1}{8}\lceil\log_2 10^9\rceil$). So each inverted file consumes $10^9\times 4$ bytes. So it takes 64GB to store all L=16 inverted files, plus 16GB due to ADC (independent on L). On the contrary, the inverted multi-index method [3] is memory efficient as a single quantizer method. It takes only 4GB for storing IDs (still plus 16GB due to ADC). In this sense, our method trades memory consumption ($\sim 4\times$) for both faster speed and higher accuracy. But we believe this is a desired property in many practices. Further, because [3] is a single-quantizer method, it is still an open problem of improving its performance if more memory is available. One may increase the number of bytes (for re-ranking) used in Multi-D-ADC. But re-ranking takes effect only when the true neighbors have been included in the short lists. Table 3 shows R@n of Multi-D-ADC becomes saturated (*e.g.*, from 0.748 to 0.751) when n increases. This implies Multi-D-ADC could hardly be improved by using more bytes for reranking. Besides, using more bytes also consumes more time in re-ranking.

As a k-means-based method, our method requires considerable *off-line* training time. The training time is mainly on obtaining the $L \times K$ codewords in the single pass of k-means. But as in most k-means practices, we empirically find it is not necessary to use all data for training. *E.g.*, in the experiments on the SIFT1B set, we only use 10% random samples for training the codebooks. For the results reported in this paper, the training time of SIFT1B is around 4-6 hours (8-core), depending on the parameters. We believe this *off-line* cost is acceptable, given the good *on-line* querying performance of the algorithm.

In this paper all the on-line querying performance are evaluated using a single thread. Our method is friendly to multi-core implementation as a multi-quantizer method. We will study this topic in the future.

References

[1] E. Agrell, T. Eriksson, A. Vardy, and K. Zeger. Closest point search in lattices. *IEEE Transactions on Information Theory*, 48(8):2201–2214, 2002.

[2] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, 51:117–122, 2008.

[3] A. Babenko and V. S. Lempitsky. The inverted multi-index. In *CVPR*, 2012.

[4] R. Baeza-Yates, B. Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. 1999.

[5] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1):107–117, 1998.

[6] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *ACM Symposium on Theory of Computing*, pages 380–388, 2002.

[7] S. Dasgupta and Y. Freund. Random projection trees and low dimensional manifolds. In *STOC*, 2008.

[8] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Symposium on Computational Geometry*, pages 253–262, 2004.

[9] Y. Freund, S. Dasgupta, M. Kabra, and N. Verma. Learning the structure of manifolds using random projections. In *NIPS*, 2007.

[10] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3:209–226, 1977.

[11] K. Fukunaga and P. M. Narendra. A branch and bound algorithm for computing k-nearest neighbors. *IEEE Trans. Computers*, 24:750–753, 1975.

[12] T. Ge, K. He, Q. Ke, and J. Sun. Optimized product quantization for approximate nearest neighbor search. In *CVPR*, 2013.

[13] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *CVPR*, 2011.

[14] R. Gray. Vector quantization. *ASSP Magazine, IEEE*, 1(2):4–29, 1984.

[15] K. He, F. Wen, and J. Sun. K-means Hashing: an Affinity-Preserving Quantization Method for Learning Binary Compact Codes. In *CVPR*, 2013.

[16] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998.

[17] H. Jegou, M. Douze, and C. Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *ECCV*, pages 304–317, 2008.

[18] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *TPAMI*, 33:117–128, 2011.

[19] H. Jegou, R. Tavenard, M. Douze, and L. Amsaleg. Searching in one billion vectors: re-rank with source coding. In *ICASSP*, pages 861–864, 2011.

[20] H. Lejsek, F. H. Ásmundsson, B. T. Jónsson, and L. Amsaleg. Nv-tree: An efficient disk-based index for approximate search in very large high-dimensional collections. *TPAMI*, 31(5):869–883, 2009.

[21] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60:91–110, 2004.

[22] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297. University of California Press, 1967.

[23] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP*, pages 331–340, 2009.

[24] D. Nistr and H. Stewnius. Scalable recognition with a vocabulary tree. In *CVPR*, 2006.

[25] A. Oliva and A. Torralba. Modeling the shape of the scene: a holistic representation of the spatial envelope. *IJCV*, 42:145–175, 2001.

[26] L. Paulevé, H. Jégou, and L. Amsaleg. Locality sensitive hashing: a comparison of hash function types and querying mechanisms. *Pattern Recognition Letters*, 31:1348–1358, 2010.

[27] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Lost in quantization: Improving particular object retrieval in large scale image databases. In *CVPR*, 2008.

[28] C. Silpa-Anan and R. Hartley. Optimised kd-trees for fast image descriptor matching. In *CVPR*, 2008.

[29] J. Sivic and A. Zisserman. Video google: a text retrieval approach to object matching in videos. In *ICCV*, pages 1470–1477, 2003.

[30] A. B. Torralba, R. Fergus, and Y. Weiss. Small codes and large image databases for recognition. In *CVPR*, 2008.

[31] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, pages 1753–1760, 2008.

[32] I. H. Witten, A. Moffat, and T. C. Bell. *Managing gigabytes: compressing and indexing documents and images*. Morgan Kaufmann, 1999.

[33] Z. Wu, Q. Ke, M. Isard, and J. Sun. Bundling features for large scale partial-duplicate web image search. In *CVPR*, 2009.