Shortlist Selection with Residual-Aware Distance Estimator for K-Nearest Neighbor Search

Jae-Pil Heo[1], Zhe Lin[2], Xiaohui Shen[2], Jonathan Brandt[2], Sung-Eui Yoon[1]
[1] KAIST [2] Adobe Research

Abstract

In this paper, we introduce a novel shortlist computation algorithm for approximate, high-dimensional nearest neighbor search. Our method relies on a novel distance estimator: the residual-aware distance estimator, that accounts for the residual distances of data points to their respective quantized centroids, and uses it for accurate shortlist computation. Furthermore, we perform the residual-aware distance estimation with little additional memory and computational cost through simple pre-computation methods for inverted index and multi-index schemes. Because it modifies the initial shortlist collection phase, our new algorithm is applicable to most inverted indexing methods that use vector quantization. We have tested the proposed method with the inverted index and multi-index on a diverse set of benchmarks including up to one billion data points with varying dimensions, and found that our method robustly improves the accuracy of shortlists (up to 127% relatively higher) over the state-of-the-art techniques with a comparable or even faster computational cost.

1. Introduction

Approximate K-nearest neighbor (ANN) search is a fundamental problem in computer science, which has many practical applications, especially in many computer vision tasks such as image retrieval, feature matching, tracking, object recognition, etc. Conventional ANN techniques can be inefficient in both speed and memory, when the size of the database is large and the dimensionality of the feature space is high, as is the case for large-scale image retrieval using holistic descriptors.

In order to achieve high scalability, recent search methods typically adopt an inverted index-based representation with a compact data representation to perform large-scale retrieval in two steps: candidate retrieval and candidate reranking. These approaches first collect candidates for K-nearest neighbors called a *shortlist* by quantized indices, and then reorder them by exhaustive distance computations with more accurate distance approximations. Accurate shortlist retrieval is a crucial first step for large-scale retrieval systems as it determines the upper-bound performance for the K-nearest neighbor search in such two-step search process.

Previous methods have attempted to introduce better quantization models (e.g., product quantization [14]) and inverted indexing schemes (e.g., the inverted index and inverted multi-index [1]). These approaches identify inverted lists whose centroids are close to the query, and include all the data points in those inverted lists to the shortlist. While these approaches are very efficient for collecting shortlists, they do not consider fine-grained positions of those data points, and thus the computed shortlist may still contain many data points that are too far away from the query, and close neighbors could be missed in the shortlist due to the quantization error.

Our contributions. In this paper, we introduce a novel shortlist computation algorithm based on the inverted lists for high-dimensional, approximate K-nearest neighbor search. We first propose a novel distance estimator, residual-aware distance estimator, between a query and data points by considering the residual distances to the quantized centroids (Sec. 4.1). We also propose effective precomputation methods of using our distance estimator for runtime queries with minor memory and computation costs with the inverted index (Sec. 4.2) and multi-index (Sec. 4.3). We have extensively evaluated our method on a diverse set of large-scale benchmarks consisting of up to one billion data with SIFT, GIST, VLAD, and CNN features. We have found that our method significantly improves the accuracy of shortlists over the state-of-the-art techniques with a comparable or even faster computational performance (Sec. 5).

2. Related Work

There have been many tree-based techniques for ANN search, since those hierarchical structures provide a logarithmic search cost. Notable approaches include KD-tree [5], randomized KD-tree forests [24], HKM (Hierarchical K-means tree) [21], etc. Unfortunately, those tree-based methods provide less effective indexing for large-scale high-dimensional data.

Designing inverted indexing structures based on vector quantization is a popular alternative to the tree-based approaches. In such methods, the index for a data point is defined by its cluster centroid in high-dimensional data, and the data point is assigned to the nearest cluster according to the distance to the centroid. Jégou et al. [14] have applied vector quantization to the approximate nearest neighbor search problem. Inverted multi-index [1] has been proposed to use product quantization [14] to generate the index. The technique can acquire a large number of clusters without incurring a high computational overhead in indexing and search. Ge et al. [7] have optimized the inverted multi-index technique by reducing the quantization error based on their prior optimization framework [6], and they mostly used two dimensional index using two subspaces. Iwamura et al. [13] have proposed a bucket distance hashing scheme that uses higher-dimensional multi-index to increase the number of indices to cover the database size, and a shortlist retrieval method specialized to their indexing method. Xia et al. [27] have proposed the joint inverted index that defines multiple sets of centroids for higher accuracy.

At a high level, the aforementioned methods based on vector quantization have been mostly focused on reducing the quantization error. In other words, they have designed more accurate vector quantization methods by increasing the number of centroids or optimizing the subspaces. While these prior techniques show high accuracy, they are mainly designed and evaluated for one nearest neighbor search, i.e., 1-NN. In contrast, our goal is to develop an accurate shortlist retrieval method for K-nearest neighbor search, where K can be large (e.g. 100, and 1000), which is useful for large-scale visual search in practice. Furthermore, these prior works are mostly evaluated on SIFT [20] and GIST [23] descriptors, but are not evaluated against very high-dimensional (e.g., 8K) and recent image descriptors such as VLAD [15] or deep convolutional neural network (CNN) features [18].

Once a shortlist is selected, the data in the shortlist is re-ranked based on exhaustive distance computations. It is impractical to use raw vectors of the data due to the consequent high computational and memory cost. Hence there have been a lot of techniques to represent data as compact codes. Those compact data representations provide benefits to both of computational and memory costs. There are two popular approaches, hashing and product quantization. Examples of hashing techniques include LSH [12, 4, 19], spectral hashing [26], ITQ [8], and etc. [9, 16, 10]. Examples of quantization-based methods include PQ [14], transform coding [2], OPQ [6], and and etc. [17, 22]. Regardless of distance computation methods used in these techniques, the performance of overall retrieval systems is highly dependent on the accuracy of the shortlist computed by indexing schemes. In this paper, we propose a shortlist method that can be used with different indexing schemes to improve the overall accuracy without incurring a high computational overhead.

3. Background

We explain the background of computing shortlists with an inverted indexing scheme.

Suppose that an inverted file consists of M inverted lists, $L_1, ..., L_M$. Each inverted list L_i has its corresponding centroid $c_i \in \mathbb{R}^D$. In general, the centroids are computed by the k-means clustering algorithm [15]. Given a database $X = \{x_1, x_2, ..., x_N\}$, each item $x \in X$ is assigned to an inverted list based on the nearest centroid index computed by a vector quantizer $q(x)$:

$$q(x) = \operatorname*{argmin}_{c_i} d(x, c_i),$$

where $d(\cdot, \cdot)$ is the Euclidean distance between two vectors. Each inverted list L_i contains data points whose nearest centroid is c_i:

$$L_i = \{x | q(x) = c_i, x \in X\} = \{x^i_1, ..., x^i_{n_i}\}.$$

When processing a query y, a shortlist S is first identified to be a set of candidate search results, whose size is T. To collect T data items from the inverted file, inverted lists are traversed in the order of increasing distance to the centroids $d(y, c_i)$. Once the shortlist S is prepared, the items in S are re-ranked by exhaustive distance evaluations with either the original data or their compact codes. The problem that we address in this paper is identifying an optimal shortlist $S \subset X$, which maximizes the recall rate for retrieval.

4. Our Approach

In this section, we first explain our distance estimator, followed by its applications to the inverted index and multi-index schemes for handling large-scale search problems.

4.1. Residual-Aware Distance Estimator

In the conventional approach, the residual distance from the data point x to its corresponding centroid $q(x)$ is omitted. In this paper, we propose a more accurate distance estimator by taking the residual distance into account. We denote this residual distance as r_x:

$$r_x = d(x, q(x)).$$

Similarly, we denote the distance between a query y and the quantized data $q(x)$ as $h_{y,x}$:

$$h_{y,x} = d(y, q(x)).$$

The exact squared distance between a query y and a data item x can be written as the following according to the law of cosines:

$$\begin{aligned} d(y,x)^2 &= h_{y,x}^2 + r_x^2 - 2h_{y,x}r_x\cos\theta \\ &= h_{y,x}^2 + r_x^2(1 - \frac{2h_{y,x}}{r_x}\cos\theta), \end{aligned} \quad (1)$$

where θ is the angle between two vectors of $y - q(x)$ and $x - q(x)$.

While the term $1 - \frac{2h_{y,x}}{r_x}\cos\theta$ depends on specific x and y, we approximate the exact distance by treating this term as a constant α_K. The reason is to constrain the distance estimator to have a factorized representation in terms of $h_{y,x}^2$ depending on y, and r_x^2, which is independent from y, for efficiency. This results in our residual-aware distance estimator:

$$\hat{d}(y,x)^2 = h_{y,x}^2 + \alpha_K r_x^2, \quad (2)$$

where α_K is a constant value within the range $[0,1]$. Shortlists computed by the residual-aware distance estimator (Eq. 2) with $\alpha_K = 0$ is identical to those of the conventional approach.

Note that two random vectors are highly likely to be orthogonal or near-orthogonal in a high-dimensional space [3, 11] and the orthogonality holds better with increasing dimensionality. As a result, we use 1 as the default value of α_K instead of zero. The distance estimator with $\alpha_K = 1$, however, is likely to overestimate distances, when two vectors of $y - q(x)$ and $x - q(x)$ are not perfectly orthogonal.

To mitigate the overestimation problem of our distance estimator, we train α_K depending on the target number of true neighbors, K, that we aim to search for. For our training process, we first randomly choose N_s data $\{s_1, ..., s_{N_s}\}$ from the database X, and compute K-nearest neighbors for each sample, s_i. Let us denote n_j^i as the j^{th} nearest neighbor of the training sample s_i. We could compute an average α_K from this set of nearest neighbor data, but it can result in over-fitting. To avoid the over-fitting issue, we also randomly select another K(=the target number of true neighbors) different data points for each s_i, denoted by $\{m_1^i, ..., m_K^i\}$. We then train α_K value with a simple equation that computes the average value from those two different data sets:

$$\alpha_K = \frac{1}{2KN_s}\sum_{i=1}^{N_s}(\sum_{j=1}^{K} f(s_i, n_j^i) + \sum_{j=1}^{K} f(s_i, m_j^i)), \quad (3)$$

where

$$f(y,x) = 1 - \frac{2h_{y,x}}{r_x}\cos\theta = \frac{d(y,x)^2 - h_{y,x}^2}{r_x^2}.$$

While training α_K values, we ignore any sample that is the cluster centroid itself (i.e., $x = q(x)$), to avoid the zero denominator. Since limited numbers of K are commonly used such as $K = 1, 50, 100$, or 1000 in practice, we can precompute α_K for a discrete set of K parameters. When we need to use a new K value that is untrained, we can simply use the default value 1 for α_K or linearly interpolated α_K based on precomputed neighboring parameters. In practice, using α_K values computed by this training process shows up to 20% higher accuracy over the default value $\alpha_K = 1$.

4.2. Inverted Index

We first explain our method with the inverted index scheme. We introduce a simple lookup table precomputation method that enables an effective and efficient way of our distance estimator for accurate shortlist computation.

4.2.1 Lookup Table Precomputation

In order to compute a shortlist according to our distance estimator (Eq. 2), we need to have the distances from data points to their corresponding cluster centroids, e.g., $r_x = d(x, q(x))$ and $h_{y,x} = d(y, q(x))$ in Eq. 2, in runtime. Unfortunately, computing such distances on-the-fly is impractical due to its computational cost and memory overhead. Furthermore, the data points are encoded into compact codes so we cannot even access the original values of those data.

To overcome these issues, we propose an efficient lookup table-based method. Our distance estimator (Eq. 2) consists of two decoupled variables $h_{y,x}$ and r_x. Since r_x is independent from a query, we can precompute those values and retrieve them in run-time. However, storing those values requires an additional memory overhead, i.e., 4 bytes for each item. Instead, we propose to use a lookup table that only contains the number of data items whose r_x belongs to a certain range, which yield a negligible storage/memory overhead. In the following, we explain the details of the look-up table construction method.

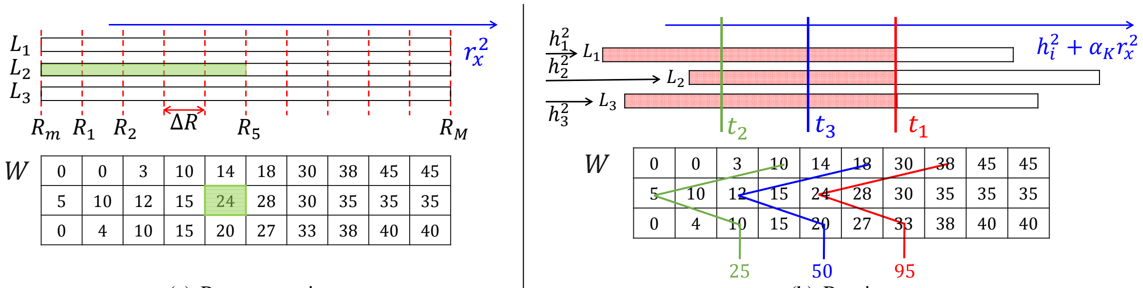
During the inverted file construction stage, we first prepare data points, x_j^i, for each inverted list, L_i, and then sort them in the non-decreasing way according to the distance between x_j^i and its centroid, c_i. Next, we compute the global minimal and maximal squared distances in the database as the following:

$$R_m = \min d(x, q(x))^2, \; R_M = \max d(x, q(x))^2.$$

We then uniformly partition the range $[R_m, R_M]$ of those squared distances into Z intervals, each of which has a ΔR span: $\Delta R = (R_M - R_m)/Z$. We denote the j-th boundary value of Z different intervals to be R_j, i.e., $R_j = R_m + j\Delta R$.

We finally define each entry of a lookup table, $W(i,j)$, to memorize the number of data points in the inverted list L_i, whose squared distances to the centroid are less than R_j as follows:

$$W(i,j) = |\{x | d(x, c_i)^2 < R_j, x \in L_i\}|, \quad (4)$$



0	0	3	10	14	18	30	38	45	45
5	10	12	15	24	28	30	35	35	35
0	4	10	15	20	27	33	38	40	40

(a) Precomputation

0	0	3	10	14	18	30	38	45	45
5	10	12	15	24	28	30	35	35	35
0	4	10	15	20	27	33	38	40	40

(b) Runtime

Figure 1. An example of our lookup table W and shortlist computation. (a) The number of data points in L_2 whose distances to the centroid c_2 are less than R_5 is $W(2, 5) = 24$. (b) The number of data whose estimated distances are less than t_1 (red boxes) is approximately the sum of values on the red line. When $T = 50$, the binary search is performed in an order of $t_1 \to t_2 \to t_3$.

where $|\cdot|$ is the cardinality of the given set. Fig. 1(a) shows an example lookup table computed by our method.

The lookup table W has $O(MZ)$ memory complexity, where M and Z are independent of the dataset size. As a result, its memory overhead can be set to be much smaller than the size of the database, while providing high performance improvement over the on-the-fly computation. For example, the overhead of the lookup table takes 64 MB for our tested benchmark consisting of 1 billion data, when we use $M = 2^{14}$ and $Z = 1024$.

4.2.2 Shortlist Computation

We precomputed the query independent term r_x^2 of our residual-aware distance estimator (Eq. 2 in the lookup table W. The distances $h_{y,x}^2$ between the query y and centroids $q(x)$, the query-dependent term of our distance estimator, can only be computed during the runtime. For simplicity, we introduce h_i^2 to denote the squared distance between a query y and c_i, i.e., $h_i^2 = d(y, c_i)^2$.

The key idea of our new shortlist computation method is to consider all the inverted list jointly by aligning them with respect to the estimated distances to the query. Before presenting our shortlist computation method, we introduce a new function, $w(y, i, t)$, which counts the number of data points in the inverted list L_i whose estimated distance from the query y is less than t, as the following:

$$\begin{aligned} w(y,i,t) &= |\{x|\hat{d}(y,x)^2 < t, x \in L_i\}| \\ &= |\{x|h_{y,x}^2 + \alpha_K r_x^2 < t, x \in L_i\}| (\because \text{Eq. } 2) \\ &= |\{x|h_i^2 + \alpha_K r_x^2 < t, x \in L_i\}| (\because q(x) = c_i) \\ &= |\{x|\alpha_K r_x^2 < t - h_i^2, x \in L_i\}| \\ &= |\{x|r_x^2 < (t - h_i^2)/\alpha_K, x \in L_i\}|. \end{aligned} \quad (5)$$

Note that Eq. 5 has the same form as Eq. 4, when $(t - h_i^2)/\alpha_K$ is replaced with R_j. To utilize the lookup table $W(i, j)$, we need to compute the index j, and we can approximate $w(y, i, t)$ as follows:

$$w(y, i, t) = W(i, \left\lceil \frac{(t - h_i^2)/\alpha_K - R_m}{\Delta R} \right\rceil). \quad (6)$$

Based on this equation, we can compute the number of data within a particular distance t by considering r_x^2 and h_i^2 terms jointly through the lookup table W. The total number of data points in all the inverted lists that are within the distance t can be computed by $\sum_{i=1}^{K} w(y, i, t)$. For example, in Fig. 1(b) the number of data points which are within the distance t_1 is computed by summing numbers on the red line.

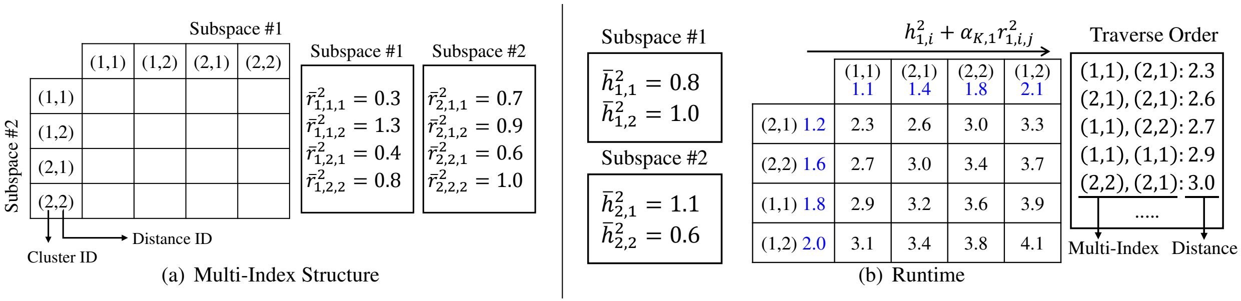
When a query y is given at runtime, we first compute and store h_i^2 for all the c_i. We then estimate the optimal threshold t of the estimated distance that meets a given shortlist size T. Since elements in each row of $W(i, \cdot)$ are arranged in the non-decreasing order, the column-wise sum in the table W has also the non-decreasing order. Therefore, $w(y, i, t)$ is also non-decreasing as we increase the value of t. Thanks to this simple property, we can use binary search to find an appropriate threshold t efficiently. The binary search for t is performed within the range of $[\min h_i^2 + \alpha_K R_m, \max h_i^2 + \alpha_K R_M]$. We stop the search when we found the largest t value that satisfies the inequality: $T \leq \sum_{i=1}^{K} w(y, i, t)$.

The final shortlist is constructed by collecting $w(y, i, t)$ data points from an inverted list L_i that have smaller estimated distances than the threshold found by the binary search (Fig. 1).

Our method performs a column-wise sum on the lookup table W and the binary search among Z intervals. As a result, our cost is $O(M \log Z)$. In practice $Z = 1024$ provides a good balance between the speed and accuracy (Sec. 5.2).

4.3. Inverted Multi-Index

The inverted multi-index (IMI) [1] supports exponentially increasing number of inverted lists. Although we can directly use the method described in Sec. 4.2 for the IMI, it can become inefficient in terms of memory and computation cost due to the large number of inverted lists. Thus, we propose a shortlist selection method tailored to the IMI with our residual-aware distance estimator.



	(1,1) 1.1	(2,1) 1.4	(2,2) 1.8	(1,2) 2.1
(2,1) 1.2	2.3	2.6	3.0	3.3
(2,2) 1.6	2.7	3.0	3.4	3.7
(1,1) 1.8	2.9	3.2	3.6	3.9
(1,2) 2.0	3.1	3.4	3.8	4.1

Figure 2. An example of our multi-index structure and shortlist computation. (a) An index in a subspace is a pair of a cluster ID and a quantized residual distance ID. The representative residual distances are pre-computed since they are independent from the query. (b) The distances from a query to centroids $h^2_{k,i}$ are computed. Based on $h^2_{k,i}$ the distances to indices in k^{th} subspace $\tilde{d}^2_{k,i,j} = h^2_{k,i} + \alpha_{K,k}\bar{r}^2_{k,i,j}$ computed (blue values). In this example, $\alpha_{K,k}$ is 1 for simplicity. The indices in each subspace are sorted. We traverse the table in order of estimated distance by using the multi-sequence algorithm [1].

4.3.1 Indexing Structure

IMI decomposes a vector x to $x = [x^1\ x^2]$, where $x^1, x^2 \in \mathbb{R}^{D/2}$ are in the 1^{st} and 2^{nd} subspaces. Cluster centroids c^1_i and c^2_i are defined in the 1^{st} and 2^{nd} subspaces ($i = 1, ..., M$), respectively. Let us redefine terms used for the inverted index (Sec. 4.2) to the k^{th} subspace: $q^k(x^k) = \text{argmin}_{c^k_i} d(x^k, c^k_i)$, $r_{x,k} = d(x^k, q^k(x^k))$, and $h_{k,i} = d(y^k, c^k_i)$.

For our method built on top of IMI, we partition i^{th} cluster in the k^{th} subspace, $X^k_i = \{x^k | q^k(x^k) = c^k_i\}$, according to the residual distances $r_{x,k}$. Each cluster X^k_i is then decomposed into P partitions $X^k_{i,1}$, ... , $X^k_{i,P}$ with residual distance boundaries, $R^k_{i,j}$:

$$X^k_{i,j} = \{x^k | R^k_{i,j-1} \le r^k_{x,i} < R^k_{i,j}, x^k \in X^k_i\}.$$

The residual distance boundaries $R^k_{i,j}$ are determined to divide the data equally into P partitions. Note that $R^k_{i,0}$ and $R^k_{i,P}$ are set to the min and max values in their subspaces, respectively. We use a pair of a cluster ID i and a distance ID j, (i, j), as the index of a subspace.

An inverted list in our multi-index is then defined by two indices:

$$L[(i_1, j_1), (i_2, j_2)] = \{x | x^1 \in X^1_{i_1,j_1}, x^2 \in X^2_{i_2,j_2}\}, \quad (7)$$

where, i_1 and i_2 are cluster IDs in 1^{st} and 2^{nd} subspaces, respectively. Also, j_1 and j_2 are the quantized distance IDs.

4.3.2 Shortlist Computation

Our residual-aware distance estimator Eq. 2 is extended to the inverted multi-index as the following:

$$\begin{aligned} \hat{d}(y, x)^2 \quad &= \quad d(y^1, q^1(x^1))^2 + \alpha_{K,1} r^2_{x,1} \\ &\quad + d(y^2, q^2(x^2))^2 + \alpha_{K,2} r^2_{x,2}, \end{aligned} \quad (8)$$

where, $\alpha_{K,k}$ is computed in the k^{th} subspace, in the same manner described in Sec. 4.1.

Storing $r^2_{x,k}$ values or computing them in the querying stage is impractical, as the case for the inverted list. We could also use our lookup table method for the multi-index. Nonetheless, this approach might generate a scalability issue for the multi-index, since our search space with the lookup table grows exponentially as we have more subspaces.

Instead, we propose to a representative residual distance for each partition for the multi-index. A representative residual distance, $\bar{r}_{k,i,j}$, for an index (i, j) in the k^{th} subspace regarding to a partition $X^k_{i,j}$ is defined by the average of residual distances of data within $X^k_{i,j}$ as the following:

$$\bar{r}_{k,i,j} = \frac{\sum r_{x,k}}{|X^k_{i,j}|}, \text{ for } x^k \in X^k_{i,j}. \quad (9)$$

We then derive a residual-aware distance estimator for our multi-index scheme for a query y and a data point $x \in L[(i_1, j_1), (i_2, j_2)]$ based on Eq. 8 and Eq. 9:

$$\tilde{d}(y, x)^2 = \underbrace{h^2_{1,i_1} + \alpha_{K,1}\bar{r}^2_{1,i_1,j_1}}_{\tilde{d}^2_{1,i_1,j_1}} + \underbrace{h^2_{2,i_2} + \alpha_{K,2}\bar{r}^2_{2,i_2,j_2}}_{\tilde{d}^2_{2,i_2,j_2}}. \quad (10)$$

The squared distance between y^k and an index (i, j) in k^{th} subspace is denoted by $\tilde{d}^2_{k,i,j}$ as indicated in the above equation.

When a query y is given at runtime, we first compute squared distances between the query y and cluster centroids $h^2_{1,i}$ and $h^2_{2,i}$. We then fetch precomputed values of $\alpha_{K,k}$ and $\bar{r}^2_{k,i,j}$ values, and compute the distances to all the indices as: $\tilde{d}^2_{k,i,j} = h^2_{k,i} + \alpha_{K,k}\bar{r}^2_{k,i,j}$ in each subspace with a minor computational cost; *i.e.* $2M$ addition operation, where M is the number of indices in each subspace. We then sort the indices in each subspace separately according to the computed distances $\tilde{d}^2_{k,i,j}$. Once the indices are sorted, we traverse the multi-index structure in the non-decreasing order of the estimated distances to select T search result candidates as the shortlist. We utilize the multi-sequence algorithm [1] based on a priority queue. Our multi-index structure and shortlist computation method is illustrated in Fig. 2.

Note that the representative residual distance method can also be applied to the inverted index (Sec. 4.2). Nonetheless, for the inverted index case, the lookup table with the binary search algorithm works better, even when we need to compute a large shortlist for the inverted index.

5. Evaluation

We use the following datasets:

- **SIFT-1M** and **SIFT-1B**: BigANN dataset [14]. 1 million and 1 billion of 128-dimensional SIFT features.
- **GIST384-1M** and **GIST384-80M**: Tiny Images [25]. 1 and 80 million of 384-dimensional GIST descriptors.
- **GIST960-1M**: BigANN dataset [14]. 1 million of 960-dimensional GIST descriptors.
- **VLAD2K-1M** and **VLAD8K-1M**: 1 million of 2048- and 8192-dimensional VLAD descriptors [15].
- **CNN-1M** and **CNN-11M**: 1 and 11 million of 4096-dimensional image features from the last fully connected layer (fc7) in the CNN [18].
- **S-VLAD2K-1B**: 1 billion of 2048-dimensional synthetic VLAD descriptors. Each VLAD descriptor is synthesized with 1000 randomly sampled SIFT features from **SIFT-1B**.

For all the datasets, we have 1000 queries disjoint from the retrieval databases.

5.1. Protocol

We evaluate the performance of different methods based on the accuracy of the shortlist as a function of shortlist size T. The accuracy is measured by recall, i.e. how many true neighbors are included in the shortlist.

We compare our method against the conventional shortlist computation method (Sec. 3) that is commonly used in [14, 1, 7]. We represent our proposed shortlist selection method as **Ours**. Each shortlist selection method is combined with three different inverted indexing schemes as follows:

- **II**: Inverted Index [14].
- **IMI**: Inverted Multi-Index [1].
- **OIMI**: Optimized Inverted Multi-Index [7].
- **BDH**: Bucket Distance Hashing [13]
- **II+Ours**: Our method combined with **II**.
- **IMI+Ours**: Our method combined with **IMI**.
- **OIMI+Ours**: Our method combined with **OIMI**.

We use 100K and 1M randomly sampled training data to train indexing methods for 1M and 1B dataset respectively. We use 500 randomly chosen samples (N_s in Sec. 4.1) from each dataset to train α_K values. For example, the trained α_{100} values for **GIST960-1M**, **VLAD2K-1M**, and **VLAD8K-1M** are 0.58, 0.80, and 0.92 respectively. We also use Z=1024 for **II+Ours** for all the experiments.

Time (ms)	GIST960	VLAD2K	VLAD8K	CNN
II+Ours	0.37, 0.63	0.64, 0.82	3.48, 3.78	1.40, 1.63
II	0.33, 0.57	0.60, 0.75	3.33, 3.50	1.34, 1.52
IMI+Ours	0.34, 1.61	0.47, 2.24	1.92, 2.75	0.77, 1.93
IMI	0.49, 2.79	0.78, 2.67	3.43, 3.54	1.51, 3.32
OIMI+Ours	0.45, 3.50	0.87, 3.96	10.9, 11.4	2.75, 6.25
OIMI	0.63, 4.15	0.99, 4.19	13.9, 15.0	4.30, 7.35
BDH	0.09, 3.43	0.14, 5.19	0.12, 4.42	0.08, 3.60

Table 1. This table shows the shortlist computation times. The first and second are times for the shortlist size $T = 800$ and $T = 51200$, respectively.

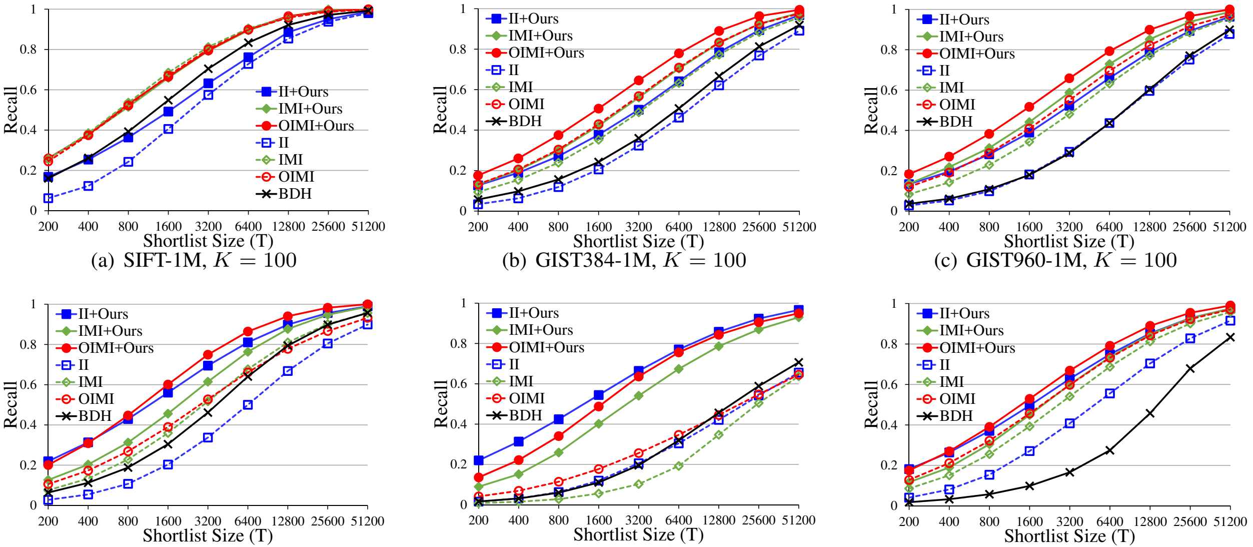
For **IMI+Ours** and **OIMI+Ours**, we use the number of partitions of each cluster to be two, i.e., P=2, and M=$M_{imi}/2$, where M_{imi} is the number of clusters used for IMI and OIMI, to set ours to have the same number of inverted lists with **IMI** and **OIMI** for a fair comparison. We also tested other settings such as $(P$=4$, M$=$M_{imi}/4)$, and found that these two settings show slightly varying performance across different benchmarks. For all the tests shown in this paper, we report results with $(P$=2$, M$=$M_{imi}/2)$. Note that when a parameter M is specified, **II** and **II+Ours** have M inverted lists, while **IMI**, **OIMI**, **IMI+Ours**, and **OIMI+Ours** have M^2 inverted lists.

All the methods are implemented in the C++ (Intel Compiler and its MKL is used for the faster performance). We conduct the experiments on a machine that consists of 2 Xeon E5-2690 CPUs and 256GB main memory. We use a single thread when measuring the computational time.

5.2. Results

Fig. 3 shows the accuracy of shortlists retrieved by the tested methods on six different benchmarks consisting of 1 million high-dimensional data, when the number of true neighbors K is 100. We use P=4 and $C = 2^{24}$ for **BDH**, and M=1024 for all the other methods. Our shortlist selection method **Ours** improved all the baselines **II**, **IMI**, and **OIMI** on **GIST384-1M**, **GIST960-1M**, **VLAD2K-1M**, **VLAD8K-1M**, and **CNN-1M** benchmarks, and provided comparable accuracy on **SIFT-1M**. For instance, our method collected 103%, 126%, and 90% more true neighbors compared to **II**, **IMI**, and **OIMI** on **VLAD8K-1M**, respectively, when T=12800(=1.28% of the benchmark size). Moreover, the performance gain by combining **Ours** becomes larger with higher dimensional data (**GIST384-1M** vs **GIST960-1M**, and **VLAD2K-1M** vs **VLAD8K-1M**). This result is achieved, mainly because our distance estimator works well even with high-dimensional cases, while employed quantization methods deteriorate with those cases.

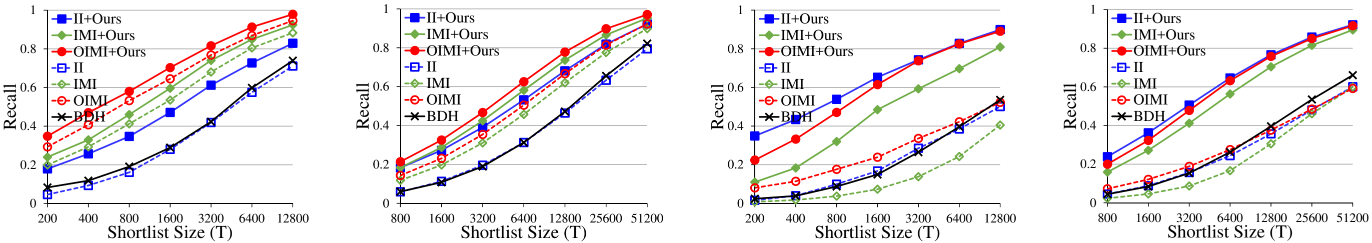
While our method shows higher accuracy over other techniques, we also analyze the computational time for



(a) SIFT-1M, $K = 100$ (b) GIST384-1M, $K = 100$ (c) GIST960-1M, $K = 100$

(d) VLAD2K-1M, $K = 100$ (e) VLAD8K-1M, $K = 100$ (f) CNN-1M, $K = 100$

Figure 3. Experimental results on SIFT-1M, GIST384-1M, GIST960-1M, VLAD2K-1M, VLAD8K-1M, and CNN-1M when the number of true neighbors $K = 100$.



(a) GIST960-1M, $K = 1$ (b) GIST960-1M, $K = 1000$ (c) VLAD8K-1M, $K = 1$ (d) VLAD8K-1M, $K = 1000$

Figure 4. Experimental results on GIST960-1M and VLAD8K-1M, and CNN-1M when the number of true neighbors $K = 1$ and 1000.

different methods. Table 1 reports the computation time to collect shortlists. **II+Ours** shows similar computation times with **II**. On the other hand, **IMI+Ours** are faster, 173%, 119%, 28%, and 72%, over **IMI** in **GIST960-1M**, **VLAD2K-1M**, **VLAD8K-1M**, and **CNN-11M**, respectively. Similarly, **OIMI+Ours** shows 118%, 6%, 32%, and 17% higher performance over **OIMI** in those datasets. The speedup comes mainly from the smaller number of centroids of our method combined with the multi-index.

We report the results with respect to the number of true neighbors K, since the desired number of neighbors can vary depending on applications. Fig. 4 shows the accuracy of shortlists on **GIST960-1M** and **VLAD8K-1M** dataset, when K=1 and K=1000. The accuracy improvement over the baseline is larger with larger K. This trend confirms the merits of our shortlist collection method based on the novel distance estimator against the baseline, which relies only on the distance between the query and the centroids.

Fig. 5(a) shows the results of **II** and **II+Ours** on **CNN-11M** with three different numbers of inverted lists M = 1024, 2048, and 8192, when K = 100. **II+Ours** consistently provides performance improvements over **II** for all the tested M. Moreover, **Ours+II** using 1024 inverted lists provides almost identical recall rates with **II** using 8192 inverted lists.

We now report results with large-scale datasets: **GIST384-80M** and two one billion descriptors of **SIFT-1B** and **S-VLAD2K-1B**. Unfortunately, we were unable to collect one billion images for the high-dimensional VLAD dataset. Instead, we construct the synthetic one, **S-VLAD2K-1B**, as described earlier. Table 2 show results with different methods. In those large-scale experiments, we used M=2^{12}, M=2^{14}, and M=2^{14} for **GIST384-80M**, **SIFT-1B**, and **S-VLAD2K-1B**, respectively. The results on **GIST384-80M** and **SIFT-1B** have similar trends to their corresponding 1M dataset. **II+Ours** identified more accurate shortlists over **II**, and **IMI+Ours** provided a higher or comparable accuracy with a faster performance compared to **IMI**.

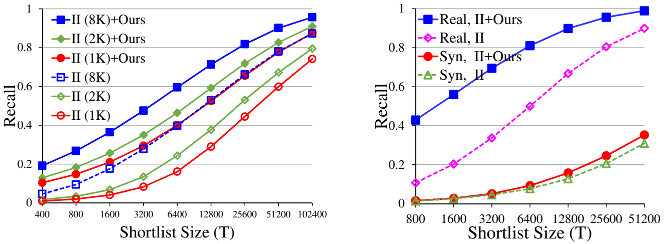
Note that our method works better with higher dimensional data as shown in Fig. 3. In a similar trend, our method tested with **S-VLAD2K-1B** outperformed the baselines, but its improvement looks marginal compared to results on **VLAD2K-1M**. We would like to point out that this

GIST384-80M							
S.List.Size T	1K	5K	10K	50K	100K	500K	1M
II+Ours	0.053 / 0.91	0.134 / 0.93	0.195 / 0.96	0.412 / 1.15	0.534 / 1.29	0.805 / 2.71	0.890 / 4.48
II	0.009 / 0.76	0.043 / 0.77	0.081 / 0.80	0.276 / 0.92	0.397 / 1.12	0.721 / 2.56	0.837 / 4.23
IMI+Ours	0.087 / 2.42	0.210 / 2.28	0.307 / 2.39	0.571 / 2.79	0.705 / 3.01	0.928 / 7.01	0.976 / 11.7
IMI	0.066 / 2.77	0.184 / 2.81	0.270 / 2.89	0.533 / 3.33	0.658 / 3.90	0.890 / 9.12	0.949 / 15.0

SIFT-1B							
S.List.Size T	1K	10K	50K	100K	500K	1M	10M
II+Ours	0.025 / 2.54	0.123 / 2.61	0.320 / 2.84	0.461 / 3.06	0.798 / 4.44	0.891 / 6.43	0.998 / 32.3
II	0.005 / 2.20	0.048 / 2.23	0.234 / 2.35	0.388 / 2.51	0.763 / 3.52	0.872 / 5.70	0.996 / 28.2
IMI+Ours	0.159 / 28.1	0.509 / 28.1	0.788 / 28.2	0.894 / 29.3	0.969 / 35.9	0.992 / 42.4	0.999 / 319
IMI	0.167 / 29.7	0.517 / 31.3	0.792 / 31.4	0.900 / 39.0	0.971 / 42.7	0.994 / 69.2	0.999 / 365

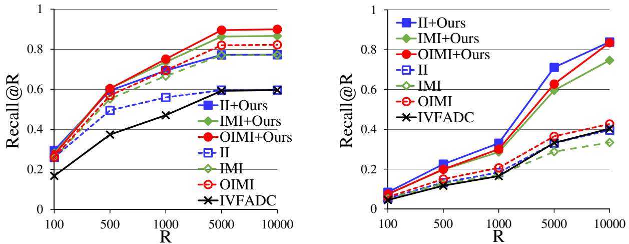
S-VLAD2K-1B							
S.List.Size T	1K	10K	50K	100K	500K	1M	10M
II+Ours	0.001 / 15.4	0.004 / 15.7	0.013 / 16.0	0.019 / 16.6	0.054 / 17.4	0.082 / 18.8	0.298 / 43.4
II	0.000 / 15.1	0.001 / 15.2	0.004 / 15.5	0.009 / 16.3	0.034 / 16.9	0.057 / 17.5	0.242 / 40.3
IMI+Ours	0.002 / 35.3	0.010 / 35.7	0.031 / 36.9	0.045 / 42.4	0.129 / 50.6	0.182 / 58.7	0.506 / 288
IMI	0.001 / 45.2	0.009 / 46.9	0.027 / 47.1	0.040 / 51.5	0.109 / 60.6	0.158 / 63.8	0.461 / 304

Table 2. Experimental results on GIST384-80M, SIFT-1B, and S-VLAD2K-1B, when $K = 100$. In each cell, the first and second values are a recall and computation time (in ms), respectively.



(a) CNN-11M (b) Real vs Synthetic VLAD

Figure 5. (a) This figure shows an experimental result on CNN-11M dataset with three different numbers of inverted lists, when $K = 100$. The number in brackets indicates the number of inverted lists M. Note that 1K is equal to 1024. (b) This figure shows the difference between real VLAD (VLAD2K-1M) and synthetic VLAD (S-VLAD2K-1M).



(a) GIST960-1M (b) VLAD8K-1M

Figure 6. Results with re-ranking when $K = 100$ and $T = 12800$.

is mainly due to the nature of the synthetic dataset, not a scalability issue of our method. In order to explain this, we provide an experimental support that shows the difference between real and synthetic data (Fig. 5(b)). As can be seen, the performance improvement using our method with a real 1 M dataset is much higher than that with a synthetic 1 M dataset. Based on this support, we expect that our method can provide higher improvements on a real dataset compared with those observed with **S-VLAD2K-1B**.

Results with re-ranking: We report the results with re-ranking in Fig. 6. In the experiments with re-ranking, we utilized 64 bits OPQ codes [6] for all tested methods except **IVFADC** [14] that uses PQ. Three methods based on the **II (II, II+Ours, IVFADC)** use residual vectors. The results confirm that for fixed encoding and shortlist size, shortlist accuracy is pivotal to end-to-end retrieval accuracy.

6. Conclusion

We have presented a novel shortlist selection algorithm for large-scale, high-dimensional approximate K-nearest neighbor search. The proposed method utilizes the residual-aware distance estimator that considers the residual distances of the data to their corresponding quantized centroids. In order to efficiently select a given size of the shortlist, we proposed effective pre-computation schemes for the inverted index and multi-index with a minor memory overhead. We have tested the proposed algorithm combined with the inverted index and inverted multi-index on large-scale benchmarks, and found that our method significantly improves the accuracy of shortlists over the prior methods with comparable or lower computational costs.

Acknowledgements

We would like to thank anonymous reviewers for constructive comments. S.-E. Yoon is a corresponding author of the paper. This work was supported by MSIP/IITP [R0126-16-1108], (MSIP) (No. 2013-067321, R0101-16-0176).

References

[1] A. Babenko and V. Lempitsky. The inverted multi-index. In *CVPR*, 2012. 1, 2, 4, 5, 6

[2] J. Brandt. Transform coding for fast approximate nearest neighbor search in high dimensions. In *CVPR*, 2010. 2

[3] T. Cai, J. Fan, and T. Jiang. Distributions of angles in random packing on spheres. *JMLR*, 2013. 3

[4] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, 2002. 2

[5] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM TOMS*, 3(3):209–226, 1977. 1

[6] T. Ge, K. He, Q. Ke, and J. Sun. Optimized product quantization for approximate nearest neighbor search. In *CVPR*, 2013. 2, 8

[7] T. Ge, K. He, Q. Ke, and J. Sun. Optimized product quantization. *IEEE TPAMI*, 2014. 2, 6

[8] Y. Gong and S. Lazebnik. Iterative quantization: a procrustean approach to learning binary codes. In *CVPR*, 2011. 2

[9] J. He, R. Radhakrishnan, S.-F. Chang, and C. Bauer. Compact hashing with joint optimization of search accuracy and time. In *CVPR*, 2011. 2

[10] J.-P. Heo, Y. Lee, J. He, S.-F. Chang, and S.-E. Yoon. Spherical hashing. In *CVPR*, 2012. 2

[11] J.-P. Heo, Z. Lin, and S.-E. Yoon. Distance encoded product quantization. In *CVPR*, 2014. 3

[12] P. Indyk and R. Motwani. Approximate nearest neighbors: toward removing the curse of dimensionality. In *STOC*, 1998. 2

[13] M. Iwamura, T. Sato, and K. Kise. What is the most efficient way to select nearest neighbor candidates for fast approximate nearest neighbor search? In *ICCV*, 2013. 2, 6

[14] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE TPAMI*, 2011. 1, 2, 6, 8

[15] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *CVPR*, pages 3304 –3311, 2010. 2, 6

[16] A. Joly and O. Buisson. Random maximum margin hashing. In *CVPR*, 2011. 2

[17] Y. Kalantidis and Y. Avrithis. Locally optimized product quantization for approximate nearest neighbor search. In *CVPR*, 2014. 2

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 2, 6

[19] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*, 2009. 2

[20] D. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004. 2

[21] D. Nistér and H. Stewénius. Scalable recognition with a vocabulary tree. In *CVPR*, 2006. 1

[22] M. Norouzi and D. J. Fleet. Cartesian k-means. In *CVPR*, 2013. 2

[23] A. Oliva and A. Torralba. Modeling the shape of the scene: a holistic representation of the spatial envelope. *IJCV*, 2001. 2

[24] C. Silpa-Anan, R. Hartley, S. Machines, and A. Canberra. Optimised kd-trees for fast image descriptor matching. In *CVPR*, 2008. 1

[25] A. Torralba, R. Fergus, and Y. Weiss. Small codes and large image databases for recognition. In *CVPR*, 2008. 6

[26] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, 2008. 2

[27] Y. Xia, K. He, F. Wen, and J. Sun. Joint inverted indexing. In *ICCV*, 2013. 2