Revisiting the Inverted Indices for Billion-Scale Approximate Nearest Neighbors

Dmitry Baranchuk
Yandex
Lomonosov Moscow State University
dmitry.baranchuk@graphics.cs.msu.ru

Artem Babenko
Yandex
Moscow Institute of Physics and Technology
artem.babenko@phystech.edu

Yury Malkov
The Institute of Applied Physics of the Russian Academy of Sciences
yurymalkov@mail.ru

Abstract

This work addresses the problem of billion-scale nearest neighbor search. The state-of-the-art retrieval systems for billion-scale databases are currently based on the inverted multi-index[2], the recently proposed generalization of the inverted index structure. The multi-index provides a very fine-grained partition of the feature space that allows extracting concise and accurate short-lists of candidates for the search queries.

In this paper, we argue that the potential of the simple inverted index was not fully exploited in previous works and advocate its usage both for the highly-entangled deep descriptors and relatively disentangled SIFT descriptors. We introduce a new retrieval system that is based on the inverted index and outperforms the multi-index by a large margin for the same memory consumption and construction complexity. For example, our system achieves the state-of-the-art recall rates up to six times faster on the dataset of one billion deep descriptors compared to the efficient implementation of the inverted multi-index from the FAISS library[14].

1. Introduction

The last decade memory-efficient billion-scale nearest neighbor search has become a significant research problem[13, 2, 9, 15, 6, 14], inspired by the needs of modern computer vision applications. In particular, since the number of images on the Internet grows enormously fast, the multimedia retrieval systems need scalable and efficient search algorithms to respond queries to the databases of billions of items in several milliseconds.

All the existing billion-scale systems avoid the infeasible exhaustive search via restricting the part of the database that is considered for a query. This restriction is performed with the help of an *indexing structure*. The indexing structures partition the feature space into a large number of disjoint regions, and the search process inspects only the points from the regions that are the closest to the particular query. The inspected points are organized in *short-lists* of candidates and the search systems calculate the distances between the query and all the candidates exhaustively. In scenarios, when the database does not fit in RAM, the compressed representations of the database points are used. The compressed representations are typically obtained with product quantization[12] that allows to compute the distances between the query and compressed points efficiently. The step of the distances calculation has a complexity that is linear in the number of candidates hence the short-lists provided by indexing structures should be concise.

The first indexing structure that was able to operate on the billion-scale datasets was introduced in [13]. It was based on the inverted index structure that splits the feature space into Voronoi regions for a set of K-Means centroids, learned on the dataset. This system was shown to achieve reasonable recall rates in several tens of milliseconds.

Later a generalization of the inverted index structure was proposed in [2]. This work introduced the inverted multi-index (IMI) that decomposes the feature space into several orthogonal subspaces and partitions each subspace into Voronoi regions independently. Then the Cartesian product of regions in each subspace formes the implicit partition of the whole feature space. Due to a huge number of regions, the IMI space partition is very fine-grained, and each region contains only a few data points. Therefore, IMI forms accurate and concise candidate lists while being memory and runtime efficient.

However, the structured nature of the regions in the IMI partition also has a negative impact on the final retrieval performance. In particular, it was shown in [6] that the majority of IMI regions contain no points and the effective number

1

of regions is much smaller than the theoretical one. For certain data distributions, this results in the fact that the search process spends much time visiting empty regions that produce no candidates. In fact, the reason for this deficiency is that the IMI learns K-Means codebooks independently for different subspaces while the distributions of the corresponding data subvectors are not statistically independent in practice. In particular, there are significant correlations between different subspaces of CNN-produced descriptors that are most relevant these days.

In this paper, we argue that the previous works underestimate the simple inverted index structure and advocate its use for all data types. To support our claim, we propose three decisive developments that allow the inverted index to outperform the existing state-of-the-art indexing structures by a huge margin.

1. We demonstrate how to increase the number of regions in the inverted index partition substantially. The efficient indexing and closest centroids retrieval remain possible with the approximate search of a high accuracy.

2. We describe how to use the product quantization [12] compression of the data points for the optimal runtime/memory efficiency in the case of a large inverted index codebook.

3. Instead of using the arbitrary point order in each region, we propose a new ordering procedure that boosts retrieval performance even further.

As a result, the proposed system achieves the state-of-the-art recall rates up to several times faster, compared to the advanced IMI implementation from the FAISS library[14] for the same memory consumption. The C++ implementation of our system is publicly available online[1].

The paper is structured as follows. We review related works on billion-scale indexing in Section 2. Section 3 describes the design of the proposed developments. The experiments demonstrating the advantage of our system are detailed in Section 4. Finally, Section 5 concludes the paper.

2. Related work

In this section we briefly review the previous methods that are related to our approach. Also here we introduce notation for the following sections.

Product quantization (PQ) is a lossy compression method for high-dimensional vectors [12]. Typically, PQ is used in scenarios when the large-scale datasets do not fit into the main memory. In a nutshell, PQ encodes each vector $x \in \mathbf{R}^D$ by a concatenation of M codewords from M

---

[1]https://github.com/dbaranchuk/ivf-hnsw

$\frac{D}{M}$-dimensional codebooks $R_1, \ldots, R_M$. Each codebook typically contains 256 codewords $R_m = \{r_1^m, \ldots, r_{256}^m\} \subset \mathbf{R}^D$ so that the codeword id could fit into one byte. In other words, PQ decomposes a vector x into M separate subvectors $[x_1, \ldots, x_M]$ and applies vector quantization (VQ) to each subvector x_m, while using a separate codebook R_m. Then the M-byte code for the vector x is a tuple of codewords indices $[i_1, \ldots, i_M]$ and the effective approximation is $x \approx [r_{i_1}^1, \ldots, r_{i_M}^M]$. As a nice property, PQ allows efficient computation of Euclidean distances between the uncompressed query and the large number of compressed vectors. The computation is performed via the ADC procedure [12] using lookup tables:

$$\|q - x\|^2 \approx \|q - [r_{i_1}^1, \ldots, r_{i_M}^M]\|^2 = \qquad (1)$$

$$\sum_{m=1}^{M} \|q_m - r_{i_m}^m\|^2$$

where q_m is the mth subvector of a query q. This sum can be calculated in M additions and lookups given that distances from query subvectors to codewords are precomputed and stored in lookup tables. Thanks to both high compression quality and computational efficiency PQ-based methods are currently the top choice for compact representations of large datasets. PQ gave rise to active research on high-dimensional vectors compression in computer vision and machine learning community[10, 18, 3, 5, 21, 22, 17, 8, 11].

IVFADC [13] is one of the first retrieval systems capable of dealing with billion-scale datasets efficiently. IVFADC uses the inverted index [19] to avoid exhaustive search and Product Quantization for database compression. The inverted index splits the feature space into K regions that are the Voronoi cells of the codebook $C = \{c_1, \ldots, c_K\}$. The codebook is typically obtained via standard K-means clustering. Then IVFADC encodes the displacements of each point from the centroid of a region it belongs to. The encoding is performed via Product Quantization with global codebooks shared by all regions.

The Inverted Multi-Index and Multi-D-ADC. The inverted multi-index (IMI) [2] generalizes the inverted index and is currently the state-of-the-art indexing approach for high-dimensional spaces and huge datasets. Instead of using the full-dimensional codebook, the IMI splits the feature space into several orthogonal subspaces (usually, two subspaces are considered) and constructs a separate codebook for each subspace. Thus, the inverted multi-index has two $\frac{D}{2}$-dimensional codebooks for different halves of the vector, each with K subspace centroids. The feature space partition then is produced as a Cartesian product of the corresponding subspace partitions. Thus for two subspaces the inverted multi-index effectively produces K^2 regions. Even for moderate values of K that is much bigger than the number of regions within the IVFADC system or other systems using inverted indices. Due to a very large number of re-

gions only a small fraction of the dataset should be visited to reach the correct nearest neighbor. [2] also describes the *multi-sequence* procedure that produces the sequence of regions that are the closest to the particular query. For dataset compression, [2] also uses Product Quantization with codebooks shared across all cells to encode the displacements of the vectors from region centroids. The described retrieval system is referred to as *Multi-D-ADC*.

The performance of indexing in the Multi-D-ADC scheme can be further improved by using the global data rotation that minimizes correlations between subspaces[9]. Another improvement[15] introduces the Multi-LOPQ system that uses local PQ codebooks for displacements compression with the Inverted Multi-Index structure.

Several other works consider the problem of the memory-efficient billion-scale search. [6] proposes the modification of the inverted multi-index that uses two non-orthogonal codebooks to produce region centroids. This modification was shown to achieve higher recall rates, but its typical runtimes exceed ten milliseconds. Another recent work [14] introduces a very efficient GPU-based approach that operates on billion-scale datasets. In this paper, we focus on the niche of CPU-based methods that operate with runtimes about one millisecond.

3. Boost Inverted Index performance

In this section we describe three essential ingredients that should be used by the inverted index and the IVFADC scheme. As will be shown experimentally, these ingredients allow IVFADC to outperform the existing state-of-the-art systems by a large margin.

Below we assume that the inverted index operates on a database $\{x_1, \ldots, x_N\} \subset \mathbf{R}^D$. The inverted index codebook $C = \{c_1, \ldots, c_K\} \subset \mathbf{R}^D$, obtained via K-Means clustering, determines the partition of the feature space into K Voronoi regions $X_i = \{x \mid i = \arg\min_k \|x - c_k\|\}$. For each database point, its displacement from the closest centroid is encoded into M bytes using PQ. The PQ codebooks are denoted by $R_1, \ldots, R_M$, each containing 256 codewords.

3.1. Ingredient 1: Large codebooks

In the original IVFADC scheme[13], the inverted index uses the exact search over the set of codebook centroids to determine the closest centroids for the particular query. Thus, the complexity of this step is linear in K, hence the inverted index cannot use the codebook larger than $2^{14}-2^{15}$ centroids while operating within several milliseconds. On the other hand, in terms of memory consumption, one could easily use a much larger codebook (say, $K = 2^{20}$) as its size is still much smaller than the compressed representation of the billion-scale database. The straightforward steps to use

larger codebooks with the approximate search of the closest centroids were not successful: [2] evaluated the scheme based on the inverted index with a very large codebook where the closest centroids were found via kd-tree[7]. It was shown that this scheme was not able to achieve recall rates higher than 0.9 for any reasonable short-list lengths due to inaccuracies of the closest centroids search. Another work[20] also employed approximate search of the IVFADC centroids but required several tens of milliseconds to accumulate reasonable number of candidates.

In this work, we show how to combine the IVFADC scheme with large codebooks without accuracy loss and with the total retrieval time about only one millisecond. For the approximate closest centroids search, we use the recent HNSW algorithm[16] from the nmslib library[1]. The algorithm is based on the proximity graph and allows to obtain a small top of the closest centroids with almost perfect accuracy in a submillisecond time. In the experiments, we observed that the usage of HNSW for the closest centroids search does not result in any noticeable loss compared to the exact search. Next, we describe the main stages of construction and usage of the inverted index that employs HNSW for the closest centroids search.

Large codebooks learning. The naive training of the inverted index codebook via K-Means would be prohibitively slow for large values of K due to computationally expensive assignment step. To make the learning process feasible, we employ the hierarchical approach: first, K-Means with a small number of centroids K_1 is trained on the subset of the training data, then all training data is distributed over obtained Voronoi regions. Finally, the data points in each of K_1 regions are clustered into $\frac{K}{K_1}$ clusters. The final inverted index codebook consists of the union of the centroids, obtained for each coarse region.

Another possible option is to use HNSW to perform the K-Means assignment step approximately. In the experiments, we did not observe any difference in the final IVFADC performance comparing to the hierarchical approach. All the results reported below were obtained with the hierarchically trained codebooks with $K_1 = 2^{13}$.

Indexing. On the indexing stage each database point should be assigned to the closest centroid. Typically, this assignment is performed via exhaustive search and has the complexity that is linear in K what is prohibitively slow for large K. Instead, we perform approximate indexing using HNSW, i.e. we find the closest centroid for each of n database points approximately. Note that such search could be easily parallelized. In the experiments, we observed that the indexing time of our scheme is on par with the indexing time of the state-of-the-art IMI implementations on the same machine and for the same number of threads.

Additional memory consumption. Here we analyze the additional memory consumption for the large codebook

and the HNSW structure. For the optimal performance the HNSW proximity graph should have about 32 edges for each item in its database, hence the total memory for all the edge lists equals $32 \cdot K \cdot \texttt{sizeof(int)}$ bytes. Given that the codebook itself requires $K \cdot D \cdot \texttt{sizeof(float)}$ bytes the overall memory consumption equals $4 \cdot K \cdot (D+32)$ (we assume 4-byte values for int and float). For typical values of D (e.g. 96 and 128 in DEEP1B and SIFT1B respectively) and a very large codebook of $K = 2^{20}$ centroids the additional memory consumption in the case of a database of one billion points is only about half a byte per point.

Note that the increase of the codebooks in the IMI is memory-inefficient as it requires $O(K^2)$ memory to maintain the indexing structure. This is an important limitation of the IMI scheme, while for the inverted index even K larger than 2^{20} are possible as the corresponding memory consumption scales linearly.

3.2. Ingredient 2: Efficient distance computation

Now we explain how large codebooks should be combined with displacements compression typically performed in IVFADC. Let us assume that a query q is traversing the region with a centroid c. Then the search process needs to evaluate the distance:

$$\|q - c - [r_1, \ldots, r_M]\|^2, \qquad (2)$$

where $[r_1, \ldots, r_M]$ is a PQ approximation of the particular database point displacement. Previous works have proposed two different ways to calculate this expression efficiently. (a) First, the original IVFADC scheme explicitly precalculated PQ lookup tables for the residual $(q-c)$ in each region before its traversal. (b) Another approach[4] transforms the distance expression(2) in the following way:

$$\|q - c - [r_1, \ldots, r_M]\|^2 = \|q - c\|^2 - 2 \sum_{m=1}^{M} \langle q_m, r_m \rangle +$$
$$2 \sum_{m=1}^{M} \langle c_m, r_m \rangle + \sum_{m=1}^{M} \|r_m\|^2 \qquad (3)$$

where q_m and c_m are the m-th subvectors of the query and the region centroid respectively. The advantage of this formulation is that the last two terms are query-independent, hence they could be precomputed offline and stored in lookup tables that allows to avoid the residual quantization in each region.

However, we argue that both approaches above can be efficiently used only for moderate codebooks in IVFADC. The approach (a) would be inefficient for large codebooks as for large K each region contains a small number of points (e.g. about thousand for $K=2^{20}$), hence many regions has to be visited to accumulate the reasonable number of candidates. As in each region the quantization of the residual vec-

tor $(q-c)$ has to be performed, the total number of quantizations would be large and the total computation time would be high. In the experiments below we show that this approach significantly slows down the proposed scheme. The limitation of the optimization (b) is the size of the lookup table for the third term in the sum (3). This table has size $256 \cdot K \cdot M \cdot \texttt{sizeof(float)}$ that is memory-inefficient for large codebooks. E.g. for $K=2^{20}$ and $M=16$ the lookup table takes 16Gb, which is infeasible for realistic setup.

Instead, we adopt the idea from the recent work[3] in order to compress the displacement vectors. Let us transform the distance expression (2) in the following way:

$$\|q - c - [r_1, \ldots, r_M]\|^2 = \|q - c\|^2 - \|c\|^2 -$$
$$2 \sum_{m=1}^{M} \langle q_m, r_m \rangle + \|c + [r_1, \ldots, r_M]\|^2 \qquad (4)$$

Now we describe how all terms in the sum (4) can be efficiently computed:

1. The term $\|q-c\|^2$ is known as it is a part of the closest centroids search result.
2. The term $\|c\|^2$ is constant for a given region and the squared norms of the region centroids could be precomputed and stored in the lookup table of K values.
3. The scalar products of the PQ codewords and the corresponding query subvectors $\langle q_m, r_m \rangle$ could be precomputed before the search, and only M lookup-add operations are needed to evaluate this term for each visited point.
4. The last term is the squared norm of the point effective approximation. This value is kept explicitly for each point in the database. Since in typical retrieval scenarios the database points are normalized, the values of these norms typically concentrate around one. Thereby, it is usually sufficient to cluster the norm values into 256 values and to keep the id of the norm cluster as an additional byte in the point code. In fact, the same optimization was used in the additive quantization work[3] to handle the norm term efficiently.

The proposed way to handle residuals codes requires only $M + 1$ lookup-add operations and does not require expensive region-level precomputations or large lookup tables. In terms of memory consumption, our approach requires one additional byte per database point. In the experiments below we show that for the same memory consumption the proposed approach is by far more efficient than a state-of-the-art IMI implementation.

3.3. Ingredient 3: Grouping and pruning

Now we describe another improvement of the IVFADC scheme that boosts its performance even further. In all the existing schemes like IVFADC or Multi-D-ADC, the points
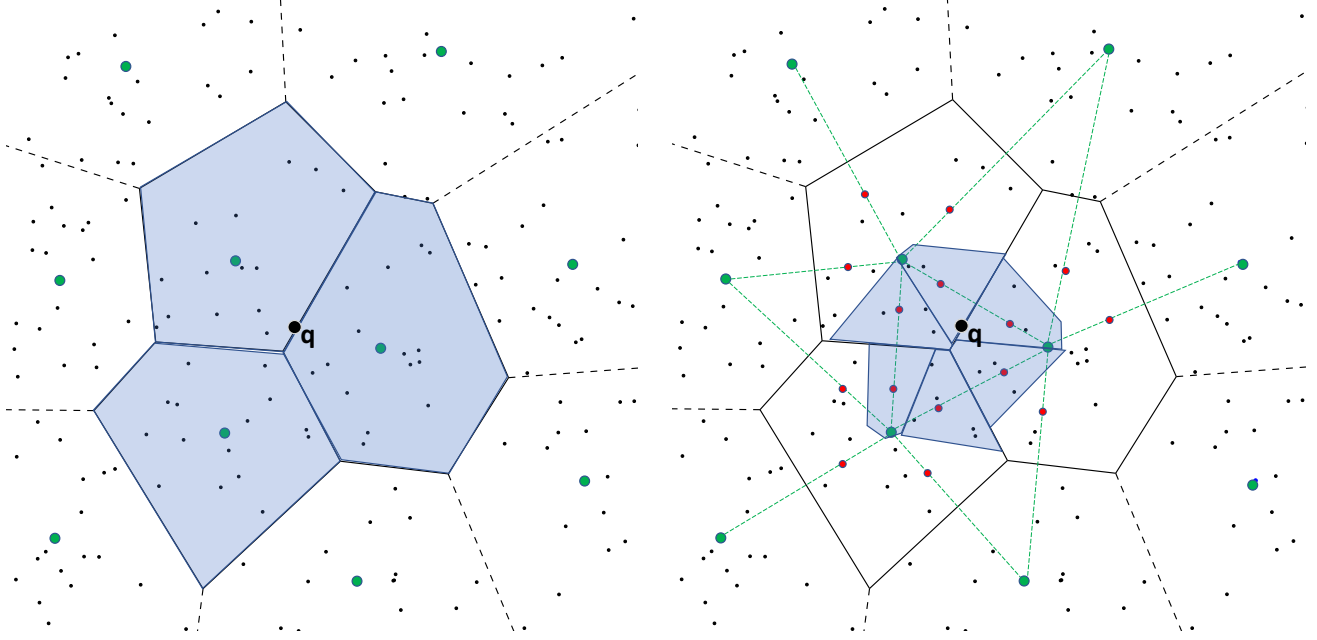
Figure 1. The indexing and the search process for the dataset of 200 two-dimensional points (small black dots) with the inverted index (left) and the inverted index augmented with grouping and pruning procedures (right). The large green points denote the region centroids, and for each centroid $L=5$ neighboring centroids are precomputed. For three regions in the center of the right plot, the region subcentroids are denoted by the red points. The fractions of the database traversed by the same query q with and without pruning are highlighted in blue. Here the query is set to visit only $\tau=40\%$ closest subregions.

in each region are kept in the arbitrary order, and the order of points has no influence on the search results. In this work, we exploit this degree of freedom and arrange region points in several groups what results in higher search efficiency. We refer to the proposed technique as *grouping* procedure.

In general, grouping organizes the points in each region into several groups such that the points in nearby locations belong to the same group. In other words, we want to split each inverted index region into a set of smaller *subregions*, corresponding to Voronoi cells of a set of *subcentroids*. The naive solution of this problem via K-Means clustering in each region would require storing full-dimensional subcentroids codebooks that would require too much memory. Instead, we propose an almost memory-free approach that constructs the subcentroids codebook in each region as a set of convex combinations of the region centroid and its neighboring centroids. We describe the grouping procedure formally below.

The model. The grouping procedure is performed independently for all the IVFADC regions so it is sufficient to describe it for the single region with the centroid c. We assume that the database points $\{x_1, \ldots, x_n\}$ belong to this region. Let us denote by $s_1, \ldots, s_L \in C$ the nearest centroids of the centroid c:

$$\{s_1, \ldots, s_L\} = \mathrm{NN}_L(c) \qquad (5)$$

where $NN_L(c)$ denotes the set of L nearest neighbors for c in the set of all centroids. The region subcentroids then

taken to be $\{c + \alpha(s_l - c)\}, l = 1, \ldots, L$, where α is a scalar parameter that is learnt from data as we describe below. Note that different α values are used in different regions. The points $\{x_1, \ldots, x_n\}$ are distributed over Voronoi subregions produced by this set of subcentroids. For each point x_i we determine the closest subcentroid

$$l_i = \arg\min_l \|x_i - (c + \alpha(s_l - c))\|^2 \qquad (6)$$

In the indexing structure the region points are stored in groups, i.e. all points from the same subregion are ordered continuously. In this scheme, it is sufficient to store only the subregion sizes to determine what group the particular point belongs to. After grouping, the displacements from the corresponding subcentroids

$$x_i - (c + \alpha(s_{l_i} - c)) \qquad (7)$$

are compressed with PQ, as in the original IVFADC. Note that the displacements to subcentroids typically have smaller norms than the displacements to the region centroid as in the IVFADC scheme. Hence they could be compressed more accurately with the same code length. This results in higher recall rates of the retrieval scheme as will be shown in the experimental section.

Distance estimation. Now we describe how to compute the distances to the compressed points after grouping. In fact, one has to calculate an expression of the form:

$$\|q - c - \alpha(s - c) - [r_1, \ldots, r_M]\|^2 \qquad (8)$$

where the $[r_1, \ldots, r_M]$ is the PQ approximation of the database point displacement. The expression (8) can be transformed in the following way:

$$\|q - c - \alpha(s - c) - [r_1, \ldots, r_M]\|^2 = (1 - \alpha)\|q - c\|^2 +$$
$$+ \alpha\|q - s\|^2 - 2 \sum_{m=1}^{M} \langle q_m, r_m \rangle + const(q) \qquad (9)$$

The first term in the sum above can be easily computed as the distance $\|q - c\|^2$ is known from the closest centroids search result. The distances $\|q - s\|^2$ are computed online before visiting the region points. Note that the sets of neighboring centroids for the close regions typically have large intersections. Note, we do not recalculate the distances $\|q - s\|^2$, which were computed earlier for previous regions, for efficiency. The scalar products between the query subvectors and PQ codewords $\langle q_m, r_m \rangle$ are precomputed before regions traversal. The last term is query-independent, and we also quantize it into 256 values and explicitly keep its quantized value as an additional byte in the point code, as described in the previous subsection. Note that the computation of distances to the neighboring centroids results in additional runtime costs. In the experiments below we show that these costs are completely justified by the improvement in the compression accuracy. The number of subregions L is set in such a way that the additional memory consumption is negligible compared to the compressed database size.

Subregions pruning. The use of the grouping technique described above also allows the search procedure to skip the least promising subregions during region traversal. This provides the total search speedup without loss in search accuracy. Below we refer to such subregions skipping as *pruning*. Let us describe pruning in more details. Consider traversing the particular region with a centroid c, the neighboring centroids $s_1, \ldots, s_L$ and the scaling factor α. The distances to the subcentroids can then be easily precomputed as follows:

$$\|q - c - \alpha(s_l - c)\|^2 = (1 - \alpha)\|q - c\|^2 +$$
$$+ \alpha\|q - s_l\|^2 + const(q), \; l = 1 \ldots L \qquad (10)$$

In the sum above the first and the second terms are computed as described in the previous paragraph while the last term is precomputed offline and stored explicitly for each neighboring centroid. If the search process is set to visit k inverted index regions, then kL distances to the subcentroids are calculated, and only a certain fraction τ of the closest subregions is visited. In practice, we observed that the search process could filter out up to half of the subregions without accuracy loss that provides additional search acceleration. Figure 1 schematically demonstrates the retrieval stage with and without pruning for the same query.

Learning the scaling factor α. Finally we describe how to learn the scaling factor α for the particular region with a

centroid c and the neighboring centroids $s_1, \ldots, s_L$. α is learnt on the hold-out learning set, and we assume that the region contains the learning points $x_1, \ldots, x_n$. We aim to solve the following minimization problem:

$$\min_{\alpha \in [0;1]} \sum_{i=1}^{n} \min_{l_i} \|x_i - c - \alpha(s_{l_i} - c)\|^2 \qquad (11)$$

In other words, we want to minimize the distances between the data points and the scaled subcentroids given that each point is assigned to the closest subcentroid. We additionally restrict α to belong to the $[0;1]$ segment so that each subcentroid is a convex combination of c and one of the neighboring centroid s.

The exact solution of the problem above requires joint optimization over the continuous variable α and the discrete variables l_i. Instead, we solve (11) approximately in two steps:

1. First, for each training point x_i we determine the optimal s_{l_i} value. This is performed by minimizing the auxiliary function that is the lower bound of the target function in (11):

$$\sum_{i=1}^{n} \min_{l_i, \alpha_i \in [0;1]} \|x_i - c - \alpha_i(s_{l_i} - c)\|^2 \qquad (12)$$

This problem is decomposable into n identical minimization subproblems for each learning point x_i:

$$\min_{\alpha_i \in [0;1], s_{l_i}} \|x_i - c - \alpha_i(s_{l_i} - c)\|^2 \qquad (13)$$

This subproblem is solved via exhaustive search over all possible s_{l_i}. For a fixed s_{l_i}, the minimization over α_i has a closed form solution and the corresponding minimum value of the target function (13) can be explicitly computed. Then the solution of the subproblem (13) for the learning point x_i is:

$$s_{l_i}^* = \arg\min_{s_{l_i}} \left\| x_i - c - \frac{(x_i - c)^T(s_{l_i} - c)}{\|s_{l_i} - c\|^2}(s_{l_i} - c) \right\|^2 \qquad (14)$$

2. Second, we minimize (11) over α with the values of $s_{l_i}^*$ obtained from the previous step. In this case the closed-form solution for the optimal value is given by:

$$\alpha = \frac{\sum_{i=1}^{n}(x_i - c)^T(s_{l_i}^* - c)}{\sum_{i=1}^{n} \|s_{l_i}^* - c\|^2} \qquad (15)$$

Discussion. The grouping and pruning procedures described above allow to increase the compression accuracy and the candidate lists quality. This results in a significant enhancement in the final system performance as will be shown in the experimental section. Note that these procedures are specific for the inverted index, and they cannot be exploited as efficiently in the inverted multi-index due to a very large number of regions in its space partition.

4. Experiments

In this section we present the experimental comparison of the proposed indexing structure and the corresponding retrieval system with the current state-of-the-art.

Datasets. We perform all the experiments on the publicly available datasets that are commonly used for billion-scale ANN search evaluation:

1. SIFT1B dataset[13] contains one billion of 128-dimensional SIFT descriptors as a base set, a hold-out learning set of 100 million vectors, and 10,000 query vectors with the precomputed groundtruth nearest neighbors.
2. DEEP1B dataset[6] contains one billion of 96-dimensional CNN-produced feature vectors of the natural images from the Web. The dataset also contains a learning set of 350 million descriptors and 10,000 queries with the groundtruth nearest neighbors for evaluation.

In most of the experiments the search accuracy is evaluated by the $Recall@R$ measure which is calculated as a rate of queries for which the true nearest neighbor is presented in the short-list of length R. All trainable parameters are obtained on the hold-out learning sets.

Indexing quality. In the first experiment we evaluate the ability of different indexing approaches to extract concise and accurate candidate lists. The candidates reranking is not performed here. We compare the following structures:

1. **Inverted Multi-Index (IMI)** [2]. We evaluate the IMI with codebooks of sizes $K = 2^{12}$ and $K = 2^{14}$ and consider the variant of the IMI with global rotation before dataspace decomposition [9] that is crucial for performance on datasets of deep descriptors. In all experiments we used the implementation from the FAISS library[14].
2. **Inverted Index**[19]. We use a large codebook of $K=2^{20}$ centroids computed as described in Section 3.1. The closest centroids for queries were obtained via HNSW method from the nmslib library[1].
3. **Inverted Index + Grouping + Pruning**. Here we augment the inverted index setup from above with the grouping and pruning procedures described in Section 3.3. The number of subregions is set to $L=64$, and the pruning process filters out 50% of the subregions during traversal.

The $Recall@R$ values for different values of R are demonstrated in Figure 2. Despite a much smaller number of regions, the inverted index produces more accurate short-lists compared to the IMI for the DEEP1B dataset. Note that the pruning procedure in the inverted index improves short-lists quality even further. The most practically important part of this plot corresponds to $R = 10^4 - 10^5$ and in this range the inverted index outperforms the IMI by up to ten percent points.

As expected, for the SIFT1B dataset, the IMI with $K=2^{14}$ produces noticeably better candidate lists than the
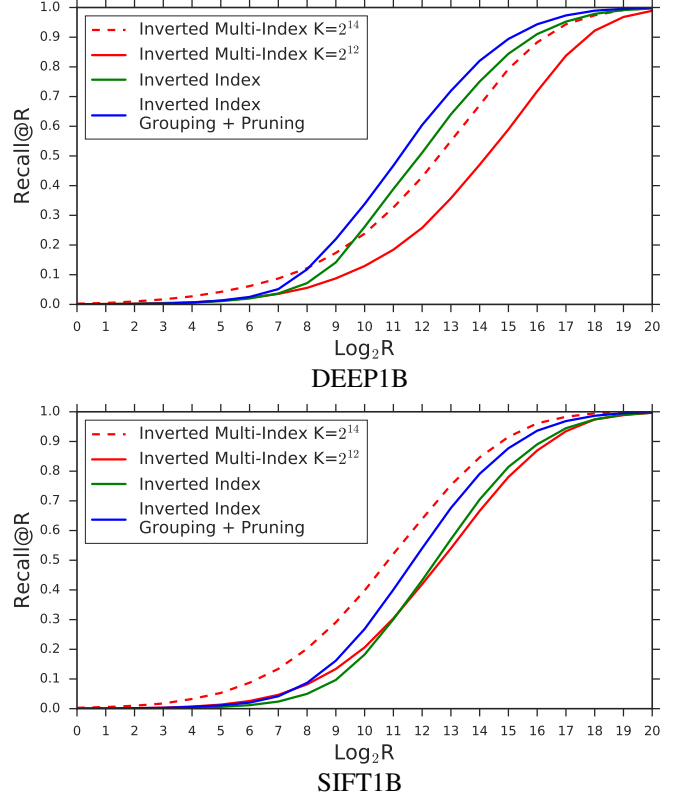

DEEP1B

SIFT1B

Figure 2. Recall as a function of the candidate list length for inverted multi-indices with $K = 2^{12}$, $K = 2^{14}$, inverted index with $K = 2^{20}$ with and without pruning. On DEEP1B (top plot) the inverted indices outperform the inverted multi-index for all reasonable values of R while for SIFT1B the IMI with larger codebooks provides the highest recall.

inverted indices. The reason is that SIFT vectors are histogram-based and the subvectors corresponding to the different halves of them describe disjoint image parts that typically have relatively weak statistical inter-dependency. However, as we show in the next experiment, the runtime cost of candidates extraction in the IMI is high due to the inefficiency of the multi-sequence algorithm.

ANN: indexing + reranking. As the most important experiment, we evaluate the performance of the retrieval systems built on top of the aforementioned indexing structures. All the systems operate in the compressed domain, i.e. the displacements of database points from their region centroids are PQ-compressed with code lengths equal to 8 or 16 bytes per point. In this experiment candidate lists are reranked based on the distances between the query and the compressed candidate points. The PQ codebooks are global and shared by all regions. We compare the following systems:

1. **O-Multi-D-ADC** is our main baseline system. It uses the inverted multi-index with global rotation and codebooks of sizes $K = 2^{12}$ and $K = 2^{14}$;
2. **IVFADC** is based on the inverted index with a codebook of size $K = 2^{20}$;

		SIFT1B								DEEP1B							
		8 bytes				16 bytes				8 bytes				16 bytes			
System	l	R@1	R@10	R@100	t	R@1	R@10	R@100	t	R@1	R@10	R@100	t	R@1	R@10	R@100	t
OMulti-D-ADC $2^{14} \times 2^{14}$	10000	0.174	0.511	0.751	2.17	0.329	0.711	0.781	2.54	0.196	0.413	0.574	1.58	0.320	0.557	0.615	1.84
	30000	0.177	0.541	0.843	4.13	0.348	0.794	0.904	5.26	0.210	0.457	0.689	2.24	0.359	0.671	0.776	2.83
	100000	0.178	0.550	0.884	10.6	0.355	0.831	0.971	14.7	0.218	0.482	0.762	6.06	0.382	0.749	0.903	8,09
OMulti-D-ADC $2^{12} \times 2^{12}$	10000	0.145	0.411	0.586	0.58	0.270	0.559	0.609	0.70	0.161	0.320	0.438	0.54	0.245	0.431	0.475	0.76
	30000	0.153	0.468	0.730	0.88	0.307	0.684	0.778	1.23	0.179	0.382	0.553	0.76	0.292	0.542	0.621	1.17
	100000	0.156	0.490	0.811	2,03	0.325	0.769	0.910	3.16	0.194	0.429	0.670	1.74	0.332	0.663	0.799	2.70
IVFADC	10000	0.146	0.423	0.602	0.47	0.292	0.577	0.623	0.53	0.214	0.447	0.629	0.42	0.349	0.612	0.675	0.53
	30000	0.158	0.480	0.750	0.83	0.331	0.710	0.798	1.04	0.228	0.492	0.734	0.78	0.388	0.719	0.822	0.97
	100000	0.165	0.501	0.828	2.15	0.351	0.786	0.918	2.59	0.233	0.507	0.782	2.01	0.405	0.773	0.916	2.52
IVFADC +Grouping	10000	0.167	0.454	0.610	0.53	0.305	0.587	0.624	0.61	0.226	0.470	0.643	0.46	0.369	0.627	0.679	0.57
	30000	0.184	0.521	0.764	0.98	0.349	0.729	0.801	1.15	0.241	0.519	0.757	0.88	0.411	0.736	0.827	1.03
	100000	0.189	0.555	0.856	2.60	0.371	0.816	0.926	3.22	0.249	0.542	0.814	2.36	0.430	0.803	0.925	2.86
IVFADC +Grouping +Pruning	10000	**0.176**	**0.492**	**0.693**	**0.62**	**0.330**	**0.664**	**0.716**	**0.67**	**0.234**	**0.496**	**0.699**	**0.51**	**0.389**	**0.679**	**0.747**	**0.63**
	30000	**0.187**	**0.542**	**0.813**	**1.16**	**0.361**	**0.774**	**0.861**	**1.25**	**0.245**	**0.531**	**0.786**	**0.97**	**0.421**	**0.768**	**0.871**	**1.18**
	100000	**0.190**	**0.560**	**0.873**	**3.01**	**0.373**	**0.829**	**0.948**	**3.75**	**0.252**	**0.550**	**0.829**	**2.73**	**0.437**	**0.820**	**0.949**	**3.26**

Table 1. The performance (recall for the top-1, top-10, and top-100 after reranking and runtime in milliseconds) of our system compared to the OMulti-D-ADC systems (based on the inverted multi-indices with $K{=}2^{12}, 2^{14}$). The results for two datasets, two compression levels, and different candidate list lengths are presented. The proposed system reaches substantially higher recall rates compared to OMulti-D-ADC, while being much more efficient. E.g. on deep descriptors our system achieves up to 26 percent points higher recall compared to OMulti-D-ADC for the same time budget.

3. **IVFADC+Grouping** additionally employs grouping procedure with $L{=}64$ subcentroids per region;

4. **IVFADC+Grouping+Pruning** employs both grouping and pruning procedures with $L{=}64$ subcentroids. The pruning is set to filter out 50% of the subregions.

We compute $R@1$, $R@10$ and $R@100$ for different numbers of candidates l for all systems on both datasets. All reported times are obtained on a single core of a 2.6GHz machine. The results are summarized in Table 1. We highlight several key observations from it:

1. The proposed system provides new state-of-the-art among CPU-based billion-scale methods both in terms of accuracy and search time. E.g., for a time budget of 0.5 milliseconds our system outperforms the OMulti-D-ADC by 7, 17 and 26 percent points of $R@1$, $R@10$ and $R@100$ respectively on the DEEP1B dataset and 8-byte codes. As for the runtime, our system reaches the previous state-of-the-art recall values much faster than OMulti-D-ADC, e.g. up to **six times faster** on the DEEP1B.

2. For SIFT descriptors, the OMulti-D-ADC with $K{=}2^{12}$ provides performance that is competitive with IVFADC without grouping and pruning procedures. **IVFADC + Grouping + Pruning** outperforms the OMulti-D-ADC by 5, 9 and 8 percent points of $R@1$, $R@10$ and $R@100$ respectively on SIFT1B and 16-byte codes.

3. OMulti-D-ADC with large codebooks $K{=}2^{14}$ produces the most accurate short-lists of candidates on SIFT1B. However, the cost of candidates extraction is much higher compared to IVFADC, as shown in Table 1. Overall, our system achieves practically the same recall rates on SIFT1B, while being several times faster.

4. Both grouping and pruning provide a substantial improvement in search accuracy, e.g. up to nine percent points in $R@100$. Note that grouping mostly increases $R@1$ values as it improves only encoding accuracy, whereas pruning also substantially improves $R@10$ and $R@100$ due to the higher quality of candidate lists.

In theory, the OMulti-D-ADC scheme with large codebooks could be accelerated by obtaining the closest subspace centroids approximately. However, in this case, one has to find several hundred closest items from a moderate codebook of size $K{=}2^{14}$, and we observed that in this setup the approximate search with HNSW takes almost the same time as brute-force. Moreover, such acceleration would not speed up the candidates accumulation that is quite slow in the multi-index due to a large number of empty regions.

Distance computation approaches. In Section 3 we discussed the motivation of using the proposed distance computation instead of the approach (a), which appears to be slow for the grouping scheme. It demands to compute a subcentroid and distance table for each subregion. Hence,
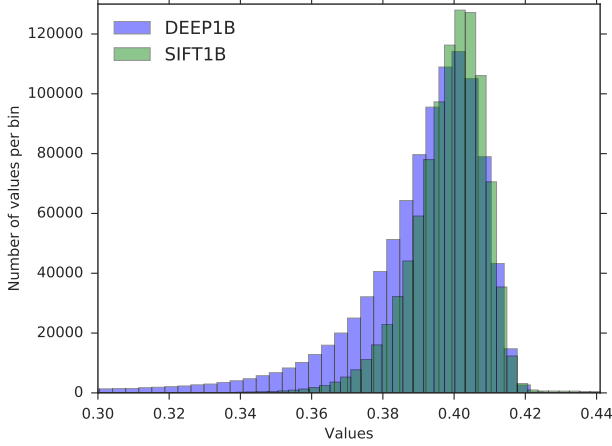
Figure 3. The scaling parameter value histograms for DEEP1B and SIFT1B. Most α values are concentrated around one point. Thus, for each dataset, individual α values can be replaced with the global one without considerable decrease of performance.

DEEP1B				
L	R@1	R@10	R@100	t
32	0.407	0.762	0.869	1.12
64	0.421	0.768	0.871	1.18
128	0.426	0.773	0.872	1.38
SIFT1B				
L	R@1	R@10	R@100	t
32	0.352	0.762	0.851	1.10
64	0.361	0.774	0.861	1.25
128	0.375	0.782	0.866	1.43

Table 2. The performance of the IVFADC + Grouping + Pruning for L=32/64/128 subcentroids per region. The results are presented for both datasets, l=30K and 16-byte codes.

OMulti-D-ADC with K=2^{14} has the memory consumption of 19.6 Gb, while IVFADC with K=2^{20} requires 20.0 Gb. The grouping procedure additionally requires about 300 Mb for L=64. Overall, the increase is only $2-4\%$ what is negligible for the most realistic setups.

5. Conclusion

In this work, we have proposed and evaluated a new system for billion-scale nearest neighbor search. The system expands the well-known inverted index structure and makes no assumption about database points distribution what makes it a universal tool for datasets with any data statistics. The advantage of the scheme is demonstrated on two billion-scale publicly available datasets.

at search time, $L \times N$ distance tables, where N is a number of visited regions, are computed, when our approach requires one inner product table computation. For clarity, for 16-byte codes on SIFT1B, the grouping scheme using the approach (a) takes 12.8 ms to process 10K points, when the proposed one requires only 0.67 ms. However, the IVFADC without grouping does not have such gap in performance, as it traverses L times less regions. E.g, for the same setup, our approach traverses 128 regions faster by 1 ms.

Scaling parameter. In the grouping scheme we learn individual scaling parameters α for each region. We provide α value histograms for both datasets in Figure 4. It appears to be interesting what if we learn the global α for all regions. In practice, there is very negligible decrease of R@1, so the global α can be used as well. However, in general, individual scaling parameters is a more accurate choice. Moreover, it does not slow down the search procedure, requires only 4Mb extra memory for K=2^{20}, and allows to work with each group independently.

Number of subregions L. Here we compare the performance of the grouping scheme for L=32/64/128 subcentroids per region. In Table 2 we provide the evaluation of the IVFADC + Grouping + Pruning for l=30K and 16-byte codes. The number of subregions L=64 and L=128 provide the best performance in terms of accuracy and search time for DEEP1B and SIFT1B respectively.

Memory consumption. We also analyze the total memory consumption for all the systems, operating on a database of N=10^9 points. In addition to $4 \cdot N$ bytes for the point ids and $M \cdot N$ bytes for the compressed representations, OMulti-D-ADC consumes $4 \cdot K^2$ bytes to store the region boundaries. On the other hand, IVFADC requires N bytes for the quantized norm terms and $4 \cdot K \cdot (D + 32)$ bytes for the HNSW graph and the codebook. E.g, for M=16

References

[1] NMSLIB library. https://github.com/searchivarius/nmslib. 3, 7

[2] A. Babenko and V. Lempitsky. The inverted multi-index. In *CVPR*, 2012. 1, 2, 3, 7

[3] A. Babenko and V. Lempitsky. Additive quantization for extreme vector compression. In *CVPR*, 2014. 2, 4

[4] A. Babenko and V. S. Lempitsky. Improving bilayer product quantization for billion-scale approximate nearest neighbors in high dimensions. *arXiv preprint arXiv:1404.1831*, 2014. 4

[5] A. Babenko and V. S. Lempitsky. Tree quantization for large-scale similarity search and classification. In *CVPR*, 2015. 2

[6] A. Babenko and V. S. Lempitsky. Efficient indexing of billion-scale datasets of deep descriptors. In *CVPR*, 2016. 1, 3, 7

[7] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9), 1975. 3

[8] M. Douze, H. Jégou, and F. Perronnin. Polysemous codes. In *ECCV*, 2016. 2

[9] T. Ge, K. He, Q. Ke, and J. Sun. Optimized product quantization. Technical report, 2013. 1, 3, 7

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

[11] H. Jain, P. Pérez, R. Gribonval, J. Zepeda, and H. Jégou. Approximate search with quantized sparse representations. In *ECCV*, 2016. 2

[12] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *TPAMI*, 33(1), 2011. 1, 2

[13] H. Jegou, R. Tavenard, M. Douze, and L. Amsaleg. Searching in one billion vectors: Re-rank with source coding. In *ICASSP*, 2011. 1, 2, 3, 7

[14] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*, 2017. 1, 2, 3, 7

[15] Y. Kalantidis and Y. Avrithis. Locally optimized product quantization for approximate nearest neighbor search. In *in Proceedings of International Conference on Computer Vision and Pattern Recognition (CVPR 2014)*. IEEE, 2014. 1, 3

[16] Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *arXiv preprint arXiv:1603.09320*, 2016. 3

[17] J. Martinez, J. Clement, H. H. Hoos, and J. J. Little. Revisiting additive quantization. In *ECCV*, 2016. 2

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

[19] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *ICCV*, 2003. 2, 7

[20] J. Wang, J. Wang, G. Zeng, R. Gan, S. Li, and B. Guo. Fast neighborhood graph search using cartesian concatenation. In *ICCV*, 2013. 3

[21] T. Zhang, C. Du, and J. Wang. Composite quantization for approximate nearest neighbor search. In *ICML*, 2014. 2

[22] T. Zhang, G.-J. Qi, J. Tang, and J. Wang. Sparse composite quantization. In *CVPR*, 2015. 2