



Hierarchical Link and Code: Efficient Similarity Search for Billion-Scale Image Sets

Kaixiang Yang¹  Hongya Wang^{1,2}  Ming Du¹ Zhizheng Wang¹ Zongyuan Tan¹ Yingyuan Xiao³

¹College of Computer Science and Technology, Donghua University, Shanghai, China

²Shanghai Key Laboratory of Computer Software Evaluating and Testing, Shanghai, China

³School of CSE, Tianjin University of Technology, China

Abstract

Similarity search is an indispensable component in many computer vision applications. To index billions of images on a single commodity server, Douze et al. introduced L&C that works on operating points considering 64–128 bytes per vector. While the idea is inspiring, we observe that L&C still suffers the accuracy saturation problem, which it is aimed to solve.

To this end, we propose a simple yet effective two-layer graph index structure, together with dual residual encoding, to attain higher accuracy. Particularly, we partition vectors into multiple clusters and build the top-layer graph using the corresponding centroids. For each cluster, a subgraph is created with compact codes of the first-level vector residuals. Such an index structure provides better graph search precision as well as saves quite a few bytes for compression. We employ the second-level residual quantization to re-rank the candidates obtained through graph traversal, which is more efficient than regression-from-neighbors adopted by L&C. Comprehensive experiments show that our proposal obtains over 30% higher recall@1 than the state-of-the-arts, and achieves up to 7.7x and 6.1x speedup over L&C on Deep1B and Sift1B, respectively.

CCS Concepts

• Information systems → Top-k retrieval in databases;

1. Introduction

Nearest neighbor search is a fundamental problem in many computer science domains such as computer vision, massive data processing and information retrieval [SDI06]. For example, it is a key component of large-scale image search [LCL04], and classification tasks with a large number of classes [DSHJ18].

In the last few years, two promising indexing paradigms for similarity search has drawn much attention in both academic and industry fields. The first one called product quantization (PQ) [JDS10] focuses on compact codes based on various quantization methods, by which image descriptors consisting of a few hundred or even thousand components are compressed employing only 8-32 bytes per descriptor.

In contrast, the graph-based similarity search paradigm offers high accuracy and efficiency, paying little attention to the memory constraint. For example, the successful approach hierarchical navigable small worlds (HNSW) by Malkov et al. [MY18] can easily achieve 95% recall with an order of magnitude speedup over other search methods [KGr]. Such outstanding performance, however,

does not come for free – it needs both the original vectors and a full graph structure to be stored in the main memory, which severely limits the scalability.

Douze et al. take the first step to conciliate these two trends by proposing Link&Code (L&C) [DSJ18], which represents vectors in the compressed domain and builds a search graph using only the compact codes of vectors. But our preliminary experiments indicate that L&C still suffers the accuracy saturation problem.

To address this issue, we propose a simple yet effective solution that fulfills all design goals of L&C and offers much higher accuracy and efficiency. Specifically, we employ a two-layer graph structure, instead of a single giant one, to organize all vectors. Such a hierarchical structure improves the graph search accuracy significantly using the same number of links as L&C. The other benefit of this design is that we only need 2 bytes to keep track of one link identifier if the sizes of all clusters are limited to 2^{16} , which is easy to enforce. Considering that L&C requires 4 bytes for one link, it saves us quite a few bytes for the compact representation of vectors.

To sum up, the contributions of this paper are:

- Through preliminary experiments we show that a hierarchical structure of small graphs provides better accuracy than a single giant one. Interestingly, it takes only 1/2 of the space cost

† Corresponding author: hywang@dhu.edu.cn

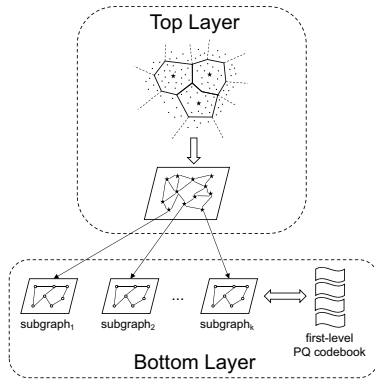


Figure 1: Overview of the two-layer graph structure of HiL&C.

required by L&C for storing link identifiers and saves quite a number of bytes for compression.

- We demonstrate that the dual residual encoding scheme offers much less reconstruction error, suggesting better approximation of original vectors is attained than the regression-from-neighbors strategy adopted by L&C.
- We introduce a hierarchical graph-based similarity search method with dual residual quantization (HiL&C), which takes full advantage of the precious memory budget. Extensive experiments show that our proposal achieves the state-of-the-art performance. To be specific, HiL&C attains 90%+ 1-recall@10 on two billion-sized benchmarks at the cost of 10ms per query in the single thread setting.

The paper is organized as follows. After a brief review of related works in Section 2, Section 3 presents the comparison of different design alternatives. We introduce our approach in Section 4 and evaluated it in Section 5, Then we conclude.

2. Related Work

Consider a set of n vectors $\mathcal{X} = \{x_1, \dots, x_n\} \subset \mathbb{R}^d$ and a distance measure $d: \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$, we focus on the problem of finding the nearest neighbors $\mathcal{N}_{\mathcal{X}}(y) \subset \mathcal{X}$ of a query $y \in \mathbb{R}^d$, that is, identifying a set of elements $\{x\}$ that minimize the distances $d(y, x)$. We routinely consider the case $d = \ell_2$, which is of high interest in computer vision applications.

Quantization-based methods. Product Quantization (PQ) [JD-S10] partitions the original high-dimensional vectors $x \in \mathbb{R}^d$ into m $\frac{d}{m}$ -dimensional subvectors $x = [x^1, \dots, x^m]$. Then PQ encodes these m $\frac{d}{m}$ -dimensional subvectors using m different subquantizers $q_i(x)$, each of which is associated with a codebook c_i . Therefore the final codebook C is the Cartesian product of the m sub-codebooks

$$C = c_1 \times \dots \times c_m$$

Each codebook includes s codewords (centroids), where s is typically set to 256 in order to fit a codeword into 1 byte. Thus, a compressed vector occupies only $m \log(s)$ bits. To process a query, all vectors are reproduced on-the-fly by consulting the codebooks and their distances to the query are evaluated to get the best answers.

The idea of re-rank with source coding is proposed in [JT-DA11] to refine the hypotheses of a query with an imprecise post-verification scheme, i.e., a first-level quantizer produces an approximate version of the vector, and an additional code refines this approximation.

Graph-based methods. Graph-based methods are currently the most efficient similarity search method, not considering the main memory constraint. Malkov *et al.* [MY18] introduced an hierarchical navigable small world graphs (HNSW), one of the most accomplished graph-based search algorithms. The main idea of HNSW is distributing vectors into multiple layers and introducing the long-range links to speedup the search procedure. Empirical study shows that HNSW exhibits $O(\log n)$ search complexity, which is rather appealing in practice. A recent proposal named NSG outperforms HNSW by constructing the so-called monotonic relative neighbor graph [FXWC19]. Quite a number of graph-based algorithms are proposed in the last few years and we would like to refer readers to [ZPZ*19, RZL20] for more details.

Link&Code. HNSW requires to store both the original vectors and the graph index in main memory, which jeopardize the scalability. In contrast, PQ-based methods supports billion-scale datasets but suffer the accuracy saturation problem. Douze *et al.* [DSJ18] introduced Link and Code (L&C) to offer a compromise between approaches considering very short codes (8-32 bytes) and those not considering the memory constraint. Specifically, L&C encodes the vectors using PQ-based methods and organizes them with a single HNSW graph. It is demonstrated that L&C beats the state-of-the-art on operating points considering 64-128 bytes per vectors.

3. Motivations

This section presents several studies that have guided the design of our approach introduced in Section 4. First, we focus on how the size of dataset affects the performance when the number of neighbors per vector on the base layer of HNSW are fixed and relatively small. Then we demonstrates the superiority of the hierarchical graph structure consisting of a number of small subgraphs over L&C. This leads us to favor dual residual quantization over quantized regression method adopted by L&C.

3.1. The impact of dataset size on accuracy

Due to the limited memory budget, L&C can only use a relatively small number of links pointing to the corresponding approximate neighbors for each point. We first examine the impact of dataset cardinality on the performance of L&C for a fixed number of links, which is set to 7 by default on the base layer of graph structure. All these evaluations are performed on Sift10K, Sift1M, Sift10M and Sift100M datasets, where Sift10K consists of the first 10K descriptors of Sift1B dataset, and so on and so forth.

We select standard parameter settings for vector quantization and regression coefficient encoding, where OPQ32 (32 bytes) is used for the first-level vector approximation and the regression coefficient takes 8 bytes per descriptor. Figure 2 reports the accuracy as a function of the search time for different datasets. The plot shows that 1) the accuracy increases as more search time is taken but tends to saturate because of the existence of reconstruction error; 2)

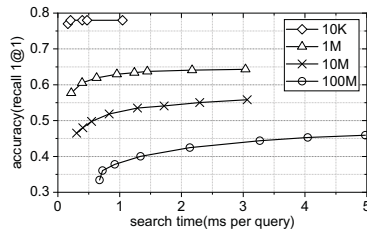


Figure 2: Accuracy vs. search time. 10K means the first 10 kilo descriptors of SIFT1B dataset, and so on and so forth.

smaller the dataset is, higher the accuracy will be. Since the memory footprint for quantization is identical for all evaluations, one can see that the accuracy is negatively correlated with the size of datasets for a fix number of links per vector. This motivates us to consider a set of small subgraphs rather than a big one.

The other benefit of using small subgraphs is that we could reduce the memory cost for storing graph links. In the original implementation of L&C, each neighbor identifier occupies 4 bytes to support billion-sized address space. Suppose we somehow could partition the big dataset into a set of small ones, of which the cardinality is less than 2^{16} . Then, only 2 bytes are enough to store each neighbor identifier, reducing the space cost by half. This saves us quite a few bytes for longer compact representation of vectors given the limited memory budget.

3.2. A big graph or a set of small subgraphs?

The previous experiment shows that using a set of small graphs might be promising in improving the accuracy and reducing the memory occupation for graph links. Inspired by IVFPQ [JTDA11], we design a hierarchical structure as illustrated in Figure 1. The top layer is a standard HNSW graph, of which vertices are the centroids obtained by employing the K-means algorithm over the whole dataset. Please note that these centroids are not quantized, i.e., the original centroid vectors are used to build the top layer graph.

Suppose we have \mathcal{K} clusters in hand, then we use L&C to index vectors in each cluster and all these \mathcal{K} L&C graphs constitute the bottom layer. A pointer is associated with each centroid in the top layer, following which one can visit its corresponding L&C graph. We call such an index structure *the hierarchical graph* (HG).

We use Sift100M to evaluate the performance of the standard L&C and HG. Particularly, a single big graph is constructed following the L&C index building strategy. For HG, we first use K-means to partition Sift100M into 5000 clusters, enforcing the size of each cluster is less than 2^{16} . A HNSW graph is built using these 5000 centroids, and then L&C is applied for each clusters to build 5000 L&C subgraphs. For both L&C and HG the parameter settings are identical, where the numbers of links are set to 7 and OPQ32 (32 bytes) is used for vector approximation.

To answer a query, L&C traverses the graph, evaluates the distances between query y and reconstructed vectors in the candidate set, and outputs the best results after the refinement stage. Since HG owns a two-layer index structure, it starts the search procedure

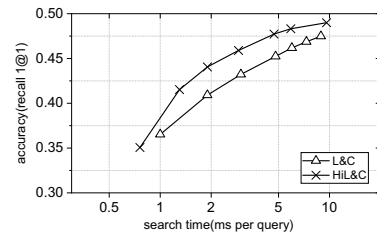


Figure 3: L&C vs. HG on SIFT100M: accuracy as a function of the search time.

by traversing the top-layer HNSW graph first, and identify k^* centroids that are closest to y . Then, the corresponding k^* L&C subgraphs are searched and the results from each subgraph are merged into a temporary candidate set, in which the closest vectors to y are chosen and output as the final answers.

We adjust the number of visited L&C subgraphs, i.e., k^* , to make the search times are identical for both L&C and HG. Figure 3 reports the accuracy as a function of search time for both methods. As one can see, HG provides much higher accuracy than L&C with the same time cost. The reasons are 1) the k^* clusters already cover most nearest neighbors of y , and 2) recall increases as the dataset cardinality decreases for a fixed number of links as discussed in Section 3.1.

4. Hierarchical link and code with dual residual encoding

In this section, we introduce our approach named *hierarchical link and code with dual residual encoding* (HiL&C). It uses a hierarchical graph structure and two-level residual quantization to build an index that scales to billion-sized datasets for efficient similarity search. After describing an overview of our indexing structure and search procedure, we present the detailed indexing and search algorithms of HiL&C. Finally, we conduct an analysis to discuss the trade-off between connectivity and coding for fixed memory budget.

4.1. Overview of the index structure

Hierarchical graph-based structure. Motivated by the discussion in Section 3.2, we employ a two-layer HSNW index structure as illustrated in Figure 1, except that we modify it so that it works with our data partitioning and dual residual encoding strategies. To be specific, each vector is stored in a cluster centroid plus compact code format, but the Add and Query operation are performed using asymmetric distance computation [JDS10]: the query vector to insert is not quantized, only the vectors already having been indexed are.

Vector approximation. All vectors are first partitioned into \mathcal{K} clusters via the K-means algorithm, and the \mathcal{K} corresponding centroids are inserted into the top-layer graph and stored in the original vector format. Please note that the number of clusters is rather small compared with the dataset cardinality, and thus is not a dominating factor of the index size. After subtracting each vector from its corresponding centroid, all first-level residuals are compressed with a

coding method independent of the structure. Formally, it is a quantizer, which maps any vector $x \in \mathbb{R}^d \mapsto q(x) \in \mathcal{C}$, where \mathcal{C} is a finite set subset of \mathbb{R}^d meaning that $q(x)$ is stored as a code. Although vector residuals are assigned to different bottom-layer subgraphs, they share the same first-level codebook for coding and decoding.

Candidate refinement. Recall that HiL&C adopts a two-stage search strategy similar to [DSJ18, JTDA11]. To re-rank a short-list of potential neighbor candidates, we compute the second-level residual for each vector and compress them using the other quantizer. To answer a query more precisely, the short-list of candidates are re-ranked using the vector estimation reconstructed on-the-fly from their second-level compact code. This trades a little extra computation per vector for better accuracy.

It is worth noting that HiL&C is a more generalized indexing framework for similarity search than L&C. Actually, if we set $\mathcal{K} = 1$ and replace the second-level residual encoding with the regression method for candidate refinement, HiL&C will reduce to L&C.

4.2. Algorithm description

This subsection presents the details of the indexing and query processing algorithms of HiL&C.

The algorithm for building a HiL&C index.

1. Learning \mathcal{K} clusters on a training dataset using K-Means and insert all centroids into the top-layer HNSW graph. The value of \mathcal{K} is chosen judiciously to make the sizes of all clusters are smaller than 2^{16} . For example, \mathcal{K} is set to around 40000 for the billion-scale datasets in our experiment setup. Each $x \in \mathcal{X}$ is assigned to its closest centroid $q_c(x)$.
2. For all vectors, the first-level residuals $r_1(x)$ are computed by $r_1(x) = x - q_c(x)$. After learning the first-level PQ quantizer $q_{r_1}(\cdot)$ with a sample set of $r_1(x)$, we insert $r_1(x)$ into its corresponding HNSW subgraph, where $r_1(x)$ is stored in the compact form of $q_{r_1}(r_1(x))$. Please note that we use the *internal ID* of x rather than the *global one* to store the neighbor identifier in the subgraph. This design saves two bytes per link. To map an internal ID to a global one during query processing, we maintain a lookup table for each subgraph, which occupies extra 4 bytes per vector.
3. To re-rank the short-list of potential candidates, we learn a second-level PQ quantizer $q_{r_2}(\cdot)$ with a sample of second-level residuals r_2 , which is computed by $r_2(x) = x - q_c(x) - q_{r_1}(r_1(x))$. Similarly, the codebook is shared by all $r_2(x)$.

The algorithm for similarity search.

1. To get the nearest neighbors of a query vector y , HiL&C starts the search by traversing the top-layer HSNW graph and return k^* closest centroids to y . The residual $r_1(y) = y - q_c(y)$ is computed for each of the k^* subgraphs. k^* is an important parameter by tuning which we can trade speed for accuracy.
2. For each of the k^* subgraphs, we perform the graph-based similarity search using $r_1(y)$ and get t best answers. The first-level residuals $r_1(x)$ of the results are reconstructed on-the-fly by first

bytes/vector	R_1	R_2	recall@1	@10	@100
16	16	0	0.416	0.834	0.887
	8	8	0.408	0.735	0.833
32	32	0	0.593	0.886	0.891
	16	16	0.624	0.874	0.886
64	64	0	0.749	0.897	0.897
	32	32	0.783	0.890	0.891

Table 1: Trade-offs for allocating bytes to quantizers for the first-level and second-level residuals on SIFT1M.

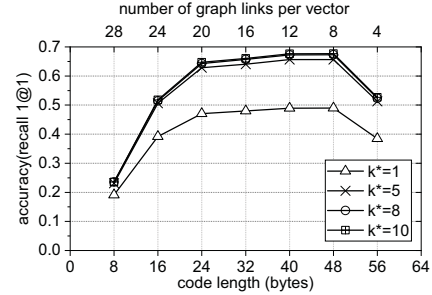


Figure 4: Performance on Deep1M by varying the number of links for a fixed memory budget of 64 bytes.

consulting the lookup table to map the internal ID into the global one, and then approximated using the corresponding compact code.

3. The short-list of potential candidates consists of all $k^* * t$ results obtained in Step 2. The second-level residual $r_2(x)$ are used to re-rank these candidates. Specifically, the distance between y and x in the short-list is computed as $d(x, y) = \|y - q_c(x) - q_{r_1}(r_1(x)) - q_{r_2}(r_2(x))\|$. We select the best results based on $d(x, y)$ as the final answers.

4.3. Memory allocation trade-offs

In this subsection, we analyze HiL&C when imposing a fixed memory budget per vector. Four factors contribute to the marginal memory fingerprint: (a) the number of graph links per vector (2 bytes per link); (b) the code used for the first-level residual approximation, for instance OPQ32 (32 bytes); (c) the R_2 bytes used by the refinement method to re-rank the short-list of candidates; (d) the lookup table mapping internal ID to the global ID (4 bytes per vector).

Since the memory occupation for the lookup table is fixed, we only focus on the compromise among the first three factors next.

Linking vs Coding. We first study the impact of the number of links on the performance of HiL&C. Note that increasing the number of links means one has to reduce the number of bytes allocated to the compression codec. Figure 4 illustrates the trade-off using a simple example on Deep1M with $R_2 = 0$. We consider all possible setups reaching the same budget of 64 bytes (the lookup table is not included), and report results for several choices of k^* , which controls the total number of distance evaluations. t is set to 150 by default.

One can observe a clear trade-off enforced by the memory constraint. The search is ineffective with too few links, as the search

graphs are too sparse to offer high accuracy. On the contrary, the precision is also crippled by the large quantization error, when the memory budget allocated to compression is not enough. Examining more subgraphs shifts the curves upwards, meaning improved accuracy at the cost of more time consumption.

First-level approximation vs refinement codec. We now fix the number of links to 6 and consider the compromise between the number of bytes allocated to the first-level and second-level residuals under fixed memory constraint. Recall that the first-level residual is used to generate a short-list of potential candidates and the second-level quantization is used for the re-rank procedure. We denote the number of bytes allocated for them by R_1 and R_2 , respectively. The number of subgraphs examined are fixed to 5.

Table 1 shows that the effectiveness of refinement procedure depends heavily on the amount of memory budget. When the total number of bytes per vector is very small, say 16, allocating bytes to the second-level residual codec hurts recall at all ranks listed, i.e., recall@1, @10 and @100. We have investigated the reason for this observation, and discovered that 1) 16 bytes are essentially not enough to obtain precise approximation of the first-level residuals, considering the size of the dataset; 2) the reconstruction error of the first approximation increases dramatically when decreasing R_1 from 16 to 8.

The picture is totally different if the memory budget is more sufficient, which is the operating point we are interested in. For example, in the case of 64 bytes per vector, allocating these bytes evenly to the two-level residual codecs improves recall@1 significantly but hardly affects recall@10 and @100, compared with $R_2 = 0$.

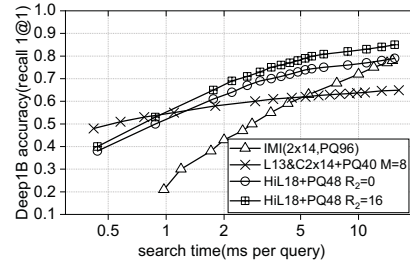
5. Experiments and analysis

In this section we present the experimental comparison of HiL&C with several state-of-the-art algorithms. All experiments are carried out on a server with E5-2620 v4@2.10GHz CPU, 256GB memory and 2TB mechanical hard drive. The accuracy is measured as the fraction of cases where the actual nearest neighbor of the query is returned at rank 1 or before some other rank, say recall@10. Following the methodology in [DSJ18], the search time is obtained with a single thread and given in milliseconds per query.

5.1. Baselines and algorithm implementation

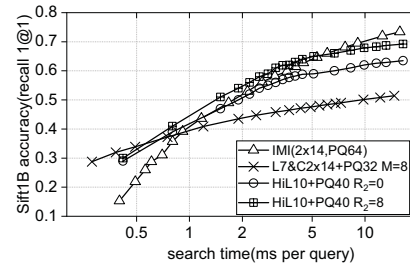
We choose Inverted Multi-Index [BL14b] (IMI) and L&C as two baseline methods because most recent works on large-scale indexing build upon IMI [BL16, BL14a, DJP16, KA14], and L&C outperforms IMI for most operating points as reported in [DSJ18]. We use the implementation of Faiss (<https://github.com/facebookresearch/faiss>) (in CPU mode) as the IMI and L&C baselines. All results are obtained using the optimal parameters selected by automatic hyperparameter tuning for them. HiL&C is implemented using the same code base of Faiss as L&C. OPQ is used to facilitate the encoding in both levels. By default, we set $\mathcal{K} = 40000$ and $t = 150$ for billion-sized datasets.

The indexing cost is an important factor for real applications. It takes much less time for HiL&C to build the index compared with L&C (20 vs. 28 hours on SIFT1B and 28 vs. 34 hours on DEEP1B).



(a) The performance on Deep1B for recall@1

Figure 5: Performance comparison of HiL&C, L&C and IMI on Deep1B.



(a) The performance on Sift1B for recall@1

Figure 6: Performance comparison of HiL&C, L&C and IMI on Sift1B.

The main reason is that building multiple small graphs is cheaper than constructing a giant one.

5.2. Empirical evaluation on billion-sized image sets

We perform all the experiments on the two publicly available billion-scale datasets, which are widely adopted by the computer vision community:

1. SIFT1B [JTDA11] contains 1 billion 128-dimensional SIFT vectors, where each vector requires 512 bytes to store.
2. DEEP1B [BL16] consists of 1 billion 96-dimensional feature vectors extracted by a CNN, where each vector occupies 384 bytes.

Both datasets come with a set of 10,000 query vectors with pre-computed ground-truth, as well as a set of unrelated training vectors to learn the codebooks for the quantizers. IMI codebooks are trained using 2 million vectors, and the regression codebooks of L&C are trained using 250k vectors and 10 iterations. The codebooks of HiL&C are learned using the same training set as L&C.

For encoding, all three methods use 104 bytes per vector for Deep1B and 72 bytes per vector for SIFT1B. Please note HiL&C requires only two bytes for each link, and maintains an additional lookup table (4 bytes per vector) for ID mapping.

Figure 5 compares the performance in terms of search time vs accuracy for different algorithms. As depicted in Figure 5 (a), HiL&C is much faster than L&C and IMI for almost all operating points.

SIFT1B					
	R@1	R@10	R@100	tims(ms)	bytes
Multi-LOPQ [KA14]	0.430	0.761	0.782	8	16
OMulti-D-OADC-L [BL14b]	0.421	0.755	0.782	7	16
FBPQ [BL14a]	0.179	0.523	0.757	1.9	16
	0.186	0.556	0.894	9.7	16
PolySemous [DJP16]	0.330	/	0.856	2.77	16
Link&Code [DSJ18]	0.461	0.608	0.613	2.10	72
HiL&C	0.542	0.694	0.697	2.06	72

Deep1B					
	R@1	R@10	R@100	tims(ms)	bytes
GNO-IMI [BL16]	0.450	0.8	/	20	16
Polysemous [DJP16]	0.456	/	/	3.66	20
Link&Code [DSJ18]	0.668	0.826	0.830	3.50	104
HiL&C	0.767	0.832	0.833	3.69	104

Table 2: Performance evaluation on two billion-sized datasets

For example, HiL&C achieves 6.1x and 3.1x speedup at 65% recall@1 compared with L&C and IMI, respectively. Moreover, HiL&C attains much higher accuracy, e.g., it provides around 85% recall@1 using 10ms per query whereas L&C has already saturated at 65%, indicating a 30%+ gain in accuracy.

Recall@10 is an important metric to evaluate if one would like to pay extra random access to the original data stored on the external memory. IMI is inferior to the other two algorithms due to its low selectivity as discussed in [DSJ18].

Figure 6 compares three algorithms on Sift1B. For most operating points, HiL&C delivers much higher accuracy than L&C and IMI. To be specific, HiL&C is 7.7x and 1.12x faster than L&C and IMI to attain a recall@1 of 51% (the saturation point of L&C), respectively. Around 39% improvement in recall@1 is achieved by HiL&C compared with L&C at the operating point of 10ms per query. For recall@10, HiL&C also demonstrates the superiority over the others. We do not report the results for recall@100 since all algorithms exhibit the similar trends as recall@10.

5.3. Comparison with other competing algorithms

Table 2 lists the comparison of HiL&C with other results reported in the literature. Note that HiL&C and L&C uses more memory than others since the design goal of both methods is optimizing the compromise between memory and accuracy. Our proposal is very competitive when one is interested in high accuracy. The other algorithms are either much time-consuming, or significantly less precise. Considering recall@1, HiL&C outperforms the state-of-the-arts by a large margin with respect to the accuracy/speed trade-off.

Providing more memory with other methods would increase the accuracy, but would also invariably increase the search time. Considering the increasing popularity of servers with 256G+ main memory, our approach offers an appealing and practical solution for most real-life computer vision applications.

6. Conclusion

In this paper, we introduced a simple yet effective approach for efficient approximate nearest search on billion-scale datasets on a single commodity server. The proposed method, HiL&C, adopts the hierarchical graph index structure and dual residual encoding

to take full advantage of the limited memory budget. The search efficiency and quantization error are both improved thanks to the delicate design choices. Empirical study shows that HiL&C outperforms the state-of-the-arts significantly.

Acknowledgements

This work is supported by the Fundamental Research Funds for the Central Universities under grant number (No: 2232021A-08), NSF of Xinjiang Key Laboratory under grant number (No:2019D04024).

References

- [BL14a] BABENKO A., LEMPITSKY V.: Improving bilayer product quantization for billion-scale approximate nearest neighbors in high dimensions. *arXiv preprint arXiv:1404.1831* (2014). 5, 6
- [BL14b] BABENKO A., LEMPITSKY V.: The inverted multi-index. *IEEE transactions on pattern analysis and machine intelligence* 37, 6 (2014), 1247–1260. 5, 6
- [BL16] BABENKO A., LEMPITSKY V.: Efficient indexing of billion-scale datasets of deep descriptors. In *CVPR* (2016), pp. 2055–2063. 5, 6
- [DJP16] DOUZE M., JÉGOU H., PERRONNIN F.: Polysemous codes. In *ECCV* (2016), Springer, pp. 785–801. 5, 6
- [DSHJ18] DOUZE M., SZLAM A., HARIHARAN B., JÉGOU H.: Low-shot learning with large-scale diffusion. In *CVPR* (2018), pp. 3349–3358. 1
- [DSJ18] DOUZE M., SABLAYROLLES A., JÉGOU H.: Link and code: Fast indexing with graphs and compact regression codes. In *CVPR* (2018), pp. 3646–3654. 1, 2, 4, 5, 6
- [FXWC19] FU C., XIANG C., WANG C., CAI D.: Fast approximate nearest neighbor search with the navigating spreading-out graph. *VLDB* 12, 5 (2019), 461–474. 2
- [JDS10] JÉGOU H., DOUZE M., SCHMID C.: Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 33, 1 (2010), 117–128. 1, 2, 3
- [JTDA11] JÉGOU H., TAVENARD R., DOUZE M., AMSALEG L.: Searching in one billion vectors: re-rank with source coding. In *ICASSP* (2011), IEEE, pp. 861–864. 2, 3, 4, 5
- [KA14] KALANTIDIS Y., AVRITHIS Y.: Locally optimized product quantization for approximate nearest neighbor search. In *CVPR* (2014), pp. 2321–2328. 5, 6
- [KGr] KGraph Project—<https://github.com/aaalgo/kgraph>. URL: <https://github.com/aaalgo/kgraph>. 1
- [LCL04] LV Q., CHARIKAR M., LI K.: Image similarity search with compact data structures. In *CIKM* (2004), pp. 208–217. 1
- [MY18] MALKOV Y. A., YASHUNIN D. A.: Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* (2018). 1, 2
- [RZL20] REN J., ZHANG M., LI D.: HM-ANN: efficient billion-point nearest neighbor search on heterogeneous memory. In *NeurIPS* (2020). 2
- [SDI06] SHAKHAROVICH G., DARRELL T., INDYK P.: *Nearest-neighbor methods in learning and vision: theory and practice (neural information processing)*. The MIT press, 2006. 1
- [ZPZ*19] ZHAO K., PAN P., ZHENG Y., ZHANG Y., WANG C., ZHANG Y., XU Y., JIN R.: Large-scale visual search with binary distributed graph at Alibaba. In *CIKM* (2019), pp. 2567–2575. 2