Scalability of the NV-tree: Three Experiments

Laurent Amsaleg, Björn Thor Jónsson, Herwig Lejsek

Scalability of the NV-tree: Three Experiments

Laurent Amsaleg[1], Björn Þór Jónsson[2], and Herwig Lejsek[3]

[1] CNRS–IRISA, Rennes, France
[2] IT University of Copenhagen, Denmark
[3] Videntifier Technologies
laurent.amsaleg@irisa.fr

Abstract. The NV-tree is a scalable approximate high-dimensional indexing method specifically designed for large-scale visual instance search. In this paper, we report on three experiments designed to evaluate the performance of the NV-tree. Two of these experiments embed standard benchmarks within collections of up to 28.5 billion features, representing the largest single-server collection ever reported in the literature. The results show that indeed the NV-tree performs very well for visual instance search applications over large-scale collections.

1 Introduction

Visual instance search (VIS) is the task of retrieving from a media collection the items that contain an actual instance of a visual query. Various real world applications require such fine-grained recognition capabilities, including forensics and copyright enforcement. A common theme in these applications is that the media collection is very large, calling for scalable indexing methods.

Due to the fine-grained nature of VIS, this domain will always require many local features for each media item. Query processing will then boil down to running multiple k-NN queries and consolidating the result into a single reply. Currently, SIFT features are the state-of-the-art for the VIS domain when considering extremely large image datasets: hundreds of local features are typically generated for each media item and result consolidation can be done via simple voting schemes. The scalability problem is compounded due to the multiple features generated: a VIS system handling tens of millions of images, for example, may need to manage tens of billions of local features.

At industry scale, VIS applications have the following requirements:

- The visual query typically results in multiple query features. Approximate indexing schemes are thus applicable to VIS applications, and in fact they can often tolerate fairly low recall.
- Each k-NN query is actually looking only for very specific features; in the case of copy detection applications typically only one feature. The appropriate quality metric is thus Recall@k.
- Due to the scale of the feature collections, data may not fit in memory. The high-dimensional index must thus support disk-based processing.

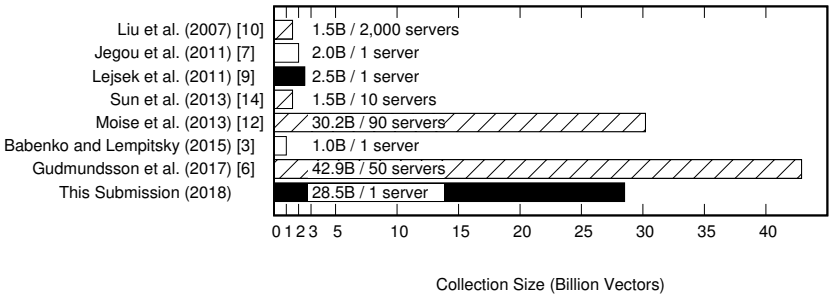


Fig. 1. The largest experimental collections reported in the literature. Black bars represent results using the NV-tree. Shaded bars represent multi-server configurations.

– VIS applications are typically part of a larger pipeline, so query processing costs must be small and, no less importantly, predictable.

In previous work, we have proposed the NV-tree [8, 9], a scalable approximate high-dimensional specifically designed for large-scale VIS applications. The NV-tree builds upon a combination of projections of data points to lines and partitioning of the projected space. By repeating the process of projecting and partitioning, data is separated into small neighborhoods which can be easily fetched from disk with a single read. By construction, the NV-tree thus guarantees query processing cost of at most one disk read per index per query feature.

Fig. 1 shows a summary of all works reported in the literature with collections of at least 1B features. As the figure shows, the NV-tree has already been used for the largest reported single-server installation, with 2.5B features. The only two other systems that have been applied at this scale are Product Quantisation (PQ) [7] and Inverted Multi-Index (IMI) [2, 3], while only installations based on Hadoop [12] and Spark [6] have used larger collections. The figure also shows that the largest collections reported in this paper contain 28.5B features; an order of magnitude more than the other single-server approaches have managed.

In this paper, we report on three experiments designed to evaluate the suitability of the NV-tree for VIS applications:

– In Section 3 we report on two image-based VIS benchmarks, embedded within collections of up to 28.5B distracting features, the largest single-server collection reported in the literature.
– In Section 4 we report on a single-query benchmark from our previous work, again embedded within collections of up to 28.5B features.
– In Section 5 we compare the NV-tree to PQ and IMI using the relatively small SIFT1B benchmark. While this is not a VIS benchmark, we use it because it represents the largest collection used by those systems. We then analyse how PQ and IMI would perform with the 28.5B collection.

The results show that the NV-tree does very well for the large-scale VIS applications. With the small-scale non-VIS benchmark, the NV-tree has better

performance but lower result quality. Further, our analysis shows that query processing performance of PQ and IMI would suffer with the 28.5B collection.

The NV-tree is proven technology, already in use at Videntifier Technologies, one of the main players in the forensics arena with technology deployed at such clients as Interpol. Their search engine targets fine-grained VIS for investigations that, e.g., aim to dismantle child abuse networks. The search engine can identify very fine-grained details in still images and videos from a collection of 150 thousand hours of video, typically scanning videos at 40x real-time speed, and while allowing dynamic insertions of about 700 hours of video material every day.

2 The NV-tree

The NV-tree [8, 9] is a disk-based high-dimensional index. It builds upon a combination of projections of data points to lines and partitioning of the projected space. By repeating the process of projecting and partitioning, data is separated into small partitions which can be easily fetched from disk with a single read, and which are likely to contain all the close neighbors in the collection. We briefly describe the NV-tree creation process, its search procedure, its dynamic insert process and and then enumerate some salient properties of the NV-tree.

2.1 Index Creation

Overall, an NV-tree is a tree index consisting of: a) a hierarchy of small *inner nodes*, which guide the feature search to the appropriate leaf node; and b) larger *leaf nodes*, which contain references to actual features. The leaf nodes are further organised into *leaf-groups* that are disk I/O units, as described below.

When tree construction starts, all features from the collection are first projected onto a single projection line through the high-dimensional space ([8] discusses projection line selection strategies). The projected values are then partitioned in 4 to 8 partitions based on their position on the projection line. Information about the partitions, such as the partition borders along the projection line, forms the first inner node of the tree—the root of the tree. To build the subsequent levels of the NV-tree, this process of projecting and partitioning is repeated recursively for each and every partition, using a new projection line for each partition, thus creating the hierarchy of smaller and smaller partitions represented by the inner nodes.

At the upper levels of the tree, with large partitions, the partitioning strategy assigns equal distance between partition boundaries at each level of the tree. The partitioning strategy changes when the features in the partition fit within 6×6 leaf nodes of 4 KB each. In this case, all the features from that partition are partitioned into a *leaf-group* made of (up to) 6 inner nodes, each containing (up to) 6 leaves. In this leaf-group, partitioning is done according to an equal cardinality criterion (instead of an equal distance criterion). Finally, for each leaf node, projection along a final random line gives the order of the feature

identifiers and the ordered identifiers are written to disk. It is important to note that the features themselves are *not* stored; only their identifiers.

Indexing a collection of high-dimensional features with an NV-tree thus creates a tree of nodes keeping track of information about projection lines and partition boundaries. All the branches of the tree end with leaf-groups with (up to) 36 leaf nodes, which in turn store the feature identifiers.

2.2 Nearest Neighbor Retrieval

During query processing, the search first traverses the hierarchy of inner nodes of the NV-tree. At each level of the tree, the query feature is projected to the projection line associated with the current node. The search is then directed to the sub-partition with center-point closest to the projection of the query feature until the search reaches a leaf-group, which is then fully fetched into RAM, possibly causing one single disk I/O. Within that leaf-group, the two nodes with center-point closest to the projection of the query feature are identified. The best two leaves from each of these two nodes are then scanned in order to form the final set of approximate nearest neighbors, with their rank depending on their proximity to the last projection of the query feature. The details of this process can be found in [9].

While the NV-tree is stored on disk, the hierarchy of inner nodes is read into memory once query processing starts, and remains fixed in memory. The larger leaf nodes, on the other hand, are read dynamically into memory as they are referenced. If the NV-tree fits into memory, the leaf nodes remain in memory and disk processing is avoided, but otherwise the buffer manager of the operating system may remove some leaf nodes from memory.

2.3 Properties of NV-trees

The experiments and analysis of [9] show that the NV-tree indexing scheme has the following properties:

- *Scalar Quantization:* The NV-tree uses random projections to turn multi-dimensional features into single-dimensional values indexed by B^+-trees. As only feature identifiers are stored, the NV-tree requires 6 bytes per feature.
- *Single Read Performance Guarantee:* In the NV-tree, leaf-groups have a fixed size. Therefore, the NV-tree guarantees query processing time of a single read regardless of the size of the feature collection. Larger collections need deeper NV-trees but intermediate nodes fit easily in memory and tree traversal cost is negligible.
- *Compact Data Structure:* The NV-tree stores in its index the identifiers of the features, not the features themselves. This amounts to about 6 bytes of storage per features on average. The NV-tree is thus a very compact data structure. Compactness is desirable as it maximizes the chances of fitting the tree in memory, thus avoiding disk operations.

– *Consolidated Result:* Random projections produce numerous false positives that can be almost all eliminated by an ensemble approach. Aggregating the results from a few NV-trees, which are built independently over the same collection, dramatically improve result quality.

3 Experiment 1: Image Benchmarks at Scale

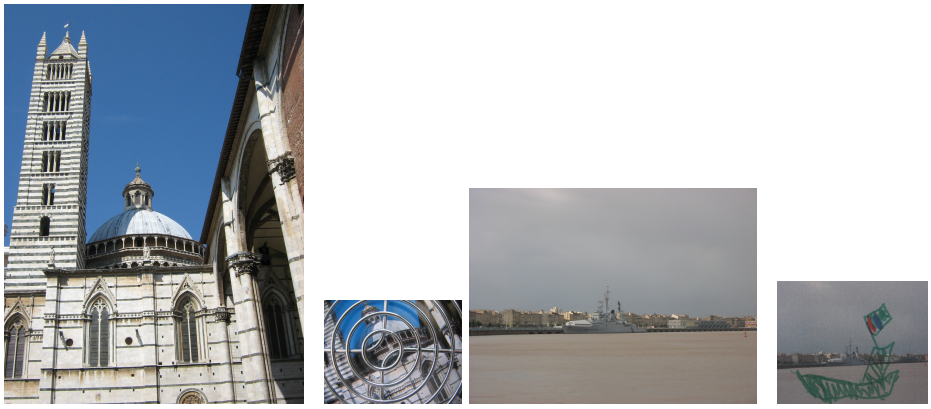
In this first experiment, we have adopted a traditional fine-grained quasi-copy paradigm, implemented using SIFT features. We report on two benchmarks from the literature, embedded in collections of up to 28.5 billion "distracting" features. The resulting feature collection is so large (3.7 Tera bytes) that no other high-dimensional indexing scheme from the litterature could handle it. The NV-tree could therefore not be compared to any other scheme from the litterature. This applies to the results presented in this section, as well as the ones reported in Section 4.

3.1 Image Datasets and Ground Truth

The image retrieval benchmarks are the "49k" benchmark [1] and the "Copydays" benchmark [5]. These benchmarks apply predefined image transformations to a particular collection of images to obtain a set of query images. We then "drown" the original images, used to create the transformed quasi-copies, within a large collection of random images which play the role of "distracting" the search. The transformed query images are then evaluated against the indexed collection and the location of the original image in the final result list is noted. When the first image in the ranked result list is the original image, the answer is considered correct; if the first image in the ranked list is not the original image, then the system is said to fail, even if that image turns out to be second in the ranked list. For each transformation, 100% success means that all the ground truth images were at the top of the corresponding ranked lists in the result set.

All pictures used in our experiments were resized such that their longer edge is 512 pixels long. We did this to ensure that the number of SIFT features [11] computed over each image was about one thousand on average. All the original images used to create the quasi-copies of the 49k and Copydays benchmarks were resized accordingly. Queries and their counterparts are therefore consistent with respect to the distracting image collection within which they are drowned.

49k Image Benchmark For this benchmark, one thousand images were randomly selected from Flickr, resulting in a very diverse collection. For each image, the Stirmark software [13] was used to generate 49 different transformations, summarized in [1]. Overall, this process generates 49,000 quasi-copy query images, hence the name "49k". Note that some of these quasi-copies are quite dissimilar from their original counterpart. For example, the CONV_2 transform tends to be extremely dark, to the point where very few SIFT features can be computed from the transformed images, making this quasi-copy very hard to



(a) (b) (c) (d)

Fig. 2. Examples from Copydays: (a) and (c) are two original images, while (b) and (d) are two strong variants used as queries. Size ratio is preserved.

find. The MEDIAN_9, NOISE_5 and PSNR_50 are also quite different, making the identification of their original counterparts challenging, especially because the SIFT features of the quasi-copies are either at different scales or at different locations in the images, since the visual noise produces very different local DoG extrema. Finally, it is worth noting that crops are inherently challenging since they dramatically reduce the number of SIFT features that can be computed for the quasi-copies, which in turn strongly decreases the number of possible matches. This is a truly difficult instance search problem.

Copydays Image Benchmark This benchmark is a publicly available collection [5] used in several publications. We use this benchmark for our experiments because it contains quasi-copies that are more severely distorted than the ones in the 49k benchmark, making the original images much harder to find. Copydays contains 157 original images. Three families of transformations have been applied, as summarized in [5], resulting in 3,055 quasi-copies in total. Some of the 229 manual transformations are particularly difficult to find since they generate quasi-copies that are visually extremely different from their original counterparts. For example, Fig. 2 shows two original images along with one strong (manually created) variant.

Distracting Image Collections To evaluate the result quality produced by the NV-tree, we inserted the original images of the 49k and Copydays image sets into a larger set of images randomly downloaded from Flickr between 2009 and 2011. The downloading process rejected images smaller than 100x100 pixels and also used MD5 signatures to reject exact duplicates of any previously downloaded images. We have gathered almost 30 million such images, and we varied the

Table 1. Distracting image collections used in Experiment 1.

Collection	Images	SIFTs	Disk Size
300M	334,268	305,443,749	40.3 GB
3B	2,970,596	3,040,856,472	401 GB
28.5B	28,969,271	28,484,904,924	3.7 TB

number of distracting images in order to study the impact of collection size on the result quality of the indexing scheme. We report on experiments with three different data collections, all including the original images from the 49k and Copydays image sets, but having a different number of distracting images, resulting in roughly 300M, 3B and 28.5B features (see Table 1).

3.2 Result Quality

49k Image Benchmark Fig. 3 shows the result quality when querying three NV-trees with the 49 transformations, varying the distracting image collections. Fig. 3 reads as follows: 100% of the ground truth images are found for 8.16% of the queries when the distracting collection is the 300M set. This means that all 1,000 ground truth images were ranked #1 in the result set for 4 of the 49 transformations. Continuing with the 300M distracting set, then 43 transformations are found from 90% to 100% of the time, or 87.17% of the queries.

Moving to larger distracting sets, 6.12% of the transformations are always found within the 3B set while this percentage goes down to 4.08% with the 28.5B set. In turn, more transformations are found between 90% and 100% of the times as depicted by the large gray area which grows to 91.84% with 3B and 93.88% with 28.5B. Overall, image retrieval works extremely well as almost all the ground truth images are found when queried with images belonging to the 49k image benchmark, with all three distracting collection sizes.

A few comments are in order, however. First, crops are always found, validating instance search. Second, one image variant is almost never found, no matter which distracting image collection is used; this is the "CONV_2" variant, which produces almost completely black images where next to no detail remains. It is common that the computation of the SIFT features on these images produces no features at all, or only one or two features, making the results random. Third, for some transformations, 100% recall is not reached. For example, for the collection that contains 28.5B features, "JPEG_15" gives 98.5%, "RML_10" gives 98.4% and "MEDIAN_9" gives 96.5%. A detailed analysis of the result lists shows that in *all these cases*, the ground truth images are ranked from #2 (the most frequent situation) to #5. This is an extremely good result, especially because our success criterion—considering only rank #1—is very strict.

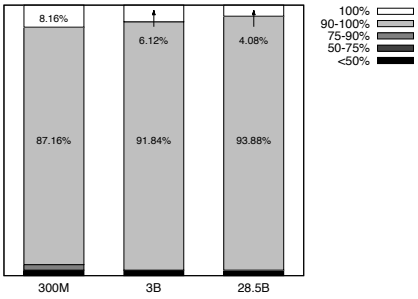


Fig. 3. Result quality for 49k.

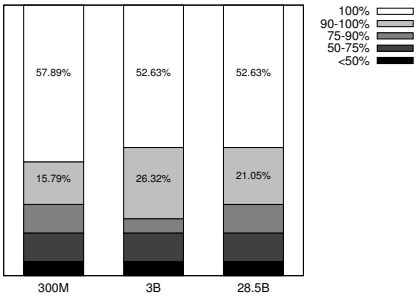


Fig. 4. Result quality for CopyDays.

Copydays Image Benchmark Fig. 4 shows the result quality for the Copydays image benchmark. Overall, the results are excellent for all but the most difficult variants. The NV-tree is able to identify the correct images most of the time, even from quite strongly distorted queries. It is not surprising to observe that quality drops with extremely compressed images (a person can sometimes hardly find any similarity between a JPEG 3% compressed image and its original version) and with some of the strong variants. Note that sometimes such attacked query images create only a handful of features, so there are too few matches for the original to rank #1—it is lost in the noise.

3.3 Retrieval Performance

We ran experiments using a Dell r710 machine that has two Intel X5650 2.67 Ghz CPUs. Each CPU has 12 MB of L3 cache that is shared by 6 actual and 6 virtual cores. The RAM consists of 18x8 GB 800 Mhz RDIMMs chips for a total of 144 GB. That machine is connected to a NAS 3070 storage system from NetApp, offering about 100TB of magnetic disk space in a RAID configuration. We ran the experiments using a single core, using three NV-trees which are probed one after the other; no parallelism is enforced in our experiments to simlipfy interpretation of the results.

When the three NV-trees fit entirely in main memory, which is the case for the 300M and the 3B collections, answering each query feature is extremely fast. A detailed analysis shows that, on average, 2,500 query features can be processed per second. On average, therefore, identifying 100 near neighbors of a single query feature takes about 0.4 milliseconds per NV-tree. In turn, as there are about 1,000 query features per image, it takes about 400 milliseconds to identify the images that are the most similar to the query image.

When using the 28.5B collection, however, no index fits entirely in main memory. In this case, the system must therefore read data from disks for almost every query feature; as each query feature is likely to access a different part of the index, no main memory buffering policy is effective. Detailed analysis shows that about 50 query features could be processed per second in this case, which is

50 times slower than for the cases where the index fit in RAM. In the case of the 28.5B collection, it is thus possible to return the answer to a single query feature in 22.47 milliseconds on average, while an image takes about 2.25 seconds.

Note that since some queries have very few features while others have more features, the retrieval time varies significantly. As pointed out earlier, however, the construction process of the NV-tree is such that the search requires only a single disk read. Because three NV-trees are used, no more than three disk reads are thus performed per feature search, and only 3,000 features are considered on average, out of the 28.5B, which is about 0.00001% of the collection.

4 Experiment 2: Single Feature Recall at Scale

This section again analyses the ability of the NV-tree index structure to cope with truly large-scale data collections, reporting results with up to 28.5B features, this time focusing on the recall of single query features.

4.1 Experimental Setup

In this experiment, we use the ground truth defined in [8]. A sequential scan was used to determine the 1,000 nearest neighbors of 500,000 query features, all coming from a collection of 180 million SIFT features. Analyzing the resulting 500M neighbors, we identified 248,212 features as being contrasted enough to be considered the true nearest neighbors of the query features. Contrast here is directly derived from criterion of [11]: a neighbor is considered a true neighbor if it is significantly closer than neighbor number one hundred.

We then embed these 248,212 features within feature sets of varying cardinalities to distract the search. These sets of distracting features have been created by extracting SIFT features from up to 30 million images randomly downloaded from Flickr. The images are ignored here, however, as we focus on individual query features. The resulting distractor sets are shown in Table 2.

This experiment focuses on recall, i.e., how many of these 248,212 ground truth features are found using the original set of 500,000 queries when varying the number of distractors, but we also report on retrieval performance. We are not aware of any other experiment ever published where recall measurements are obtained from searching the nearest neighbors of individual query features lost within 28.5B distracting features.

4.2 Quality of Nearest Neighbor Retrieval

Fig. 5 shows the recall for the different distractor collections. The x-axis shows the size of the distractor collections (note the logarithmic scale), while the y-axis shows the impact on recall of using a varying number of NV-trees. Up to three NV-trees were used against all the collections. We also considered using up to six NV-trees to improve recall; as such experiments are complicated and time consuming, however, we used only two moderate size datasets for this purpose.

Table 2. Distracting feature collections for Experiment 2.

Collection	SIFT Features	Feature Collection Size	NV-tree Size
30M	28,799,690	3.8 GB	180 MB
180M	179,443,881	23.6 GB	1 GB
300M	305,443,749	40.3 GB	1.9 GB
2.5B	2,485,568,191	328 GB	14 GB
3B	3,040,856,472	401 GB	17 GB
28.5B	28,484,904,924	3.7 TB	162 GB

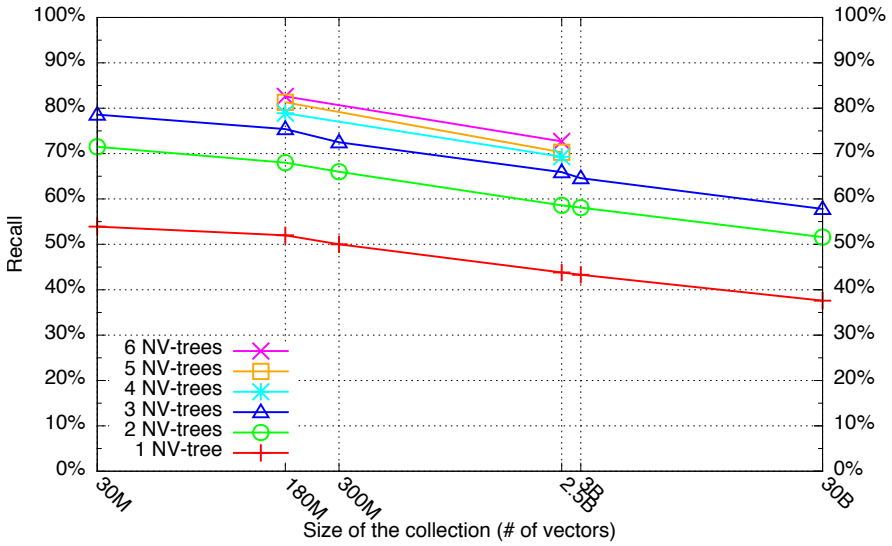


Fig. 5. Recall, varying collection sizes, varying number of NV-trees.

Fig. 5 shows that when using a single NV-tree, recall is relatively low. Close to 54% of the 248,212 ground truth features are found when they are lost in the 30M collection. This percentage then slowly decreases as the distracting collection grows, to about 38% when challenged by the 28.5B collection.

Using additional indices dramatically improves performance, however. With the 30M collection, recall jumps to 72% with two NV-trees and 79% using three NV-trees. At the other end of the figure, with the 28.5B collection, recall is lower as before but remains remarkably good given the size of the distracting collection: 52% with two NV-trees and 58% with three NV-trees.

Using more than three NV-trees provides a slight recall improvement, but not as dramatic as going from a single NV-tree to two and three. Multiplying the number of NV-trees is therefore not a worthy option, since it increases the pressure on storage and main memory and increases the retrieval cost.

4.3 Retrieval Performance

We now turn to the retrieval performance. We ran this experiment on the same system as in the previous experiment, and therefore the retrieval performance was identical. Recall that the main memory of our server was 144 GB, which means that all the leaves of three NV-trees can fit into memory for all collections except the 28.5B collection. When the various indices entirely fit in main memory, answering each query feature is extremely fast, as before: it takes a fraction of a millisecond to process one feature against one NV-tree. When using the 28.5B collection, on the other hand, main memory can not fit even one NV-tree, and the response time is therefore larger, but still only about 22.5 ms per query.

5 Experiment 3: The Small-Scale SIFT1B Benchmark

In this experiment we examine the SIFT1B benchmark, as this is the largest benchmark for which there are results for the primary competitors from the literature: Product Quantization (PQ) [7] and the Inverted Multi-Index (IMI) [2, 3]. Note, however, that we consider SIFT1B to be a relatively small benchmark. As the NV-tree index was designed primarily for very large collections, this analysis favors its competitors significantly.

Previous work has directly compared the NV-tree to many approaches, including LSH, median rank aggregation and the Spill-Tree, showing that NV-tree significantly outperforms those approaches already at a much smaller scale [8]. But it is insightful to determine the disk space requirements and expected disk read performance if the more recent state-of-the-art techniques were used to index the 28.5B features used in the previous two experiments. We make this comparison at the end of this section.

5.1 Experimental Setup

SIFT1B is a collection of exactly one billion 128-dimensional SIFT features extracted from natural images [7], which is publicly available and has been used in many publications, including [3]. The dataset comes with pre-calculated ground truth, where the exact 1,000 nearest neighbors for each of the 10,000 queries are provided; these neighbors were identified using a (long!) sequential scan computing euclidean distances. The measure used to compare systems indexing this SIFT1B dataset is the Recall@R, which for varying values of R determines the average rate of queries for which the 1-nearest neighbor is ranked in the top R positions. The SIFT1B data set was indexed by the NV-tree and the queries run. We then compare those results to the results reported in [3], which were obtained using fairly similar hardware. In our analysis, we focus on the Recall@1 measure, and observe that all collections fit easily in main memory, as we use the same server as before.

Table 3. Performance Comparison Using SIFT1B. Results for PQ and IMI variants are reproduced from [3].

Indexing Method	Bytes / Feature	Quality (Recall@1)	Features Read	Retrieval Time	Index Size
NV-tree (3 indices)	18	0.076	3 K	1.2 ms	18 GB
PQ	12	0.112	∼8 million	155 ms	12 GB
Multi-Index	12	0.158	10 K	2 ms	13 GB
	12	0.165	100 K	11 ms	13 GB
- OMulti-D-OADC-Local	12	0.268	10 K	6 ms	15 GB
	12	0.286	100 K	50 ms	15 GB
- 16 bytes per feature &	20	0.421	10 K	7 ms	23 GB
OMulti-D-OADC-Local	20	0.467	100 K	66 ms	23 GB

5.2 Results

Table 3 shows the comparison of the three methods: PQ, IMI, and NV-tree. For IMI, three variants are shown, where the latter apply some additional optimizations [3]. As the table shows, IMI improves on PQ in all aspects: result quality, features read, and retrieval time. Optimizations to IMI then further improve quality but at a very significant cost of retrieval time.

Table 3 also shows that while the NV-tree returns worse results, it performs better, both in terms of features read and retrieval time. By construction, the NV-tree scans contents from exactly four leafs in order to build its result, and was asked to return only $k = 1,000$ neighbors from each tree, for a total of 3,000 features scanned. As the NV-tree only stores the identifier of the feature, no re-ranking is performed after retrieval, unlike IMI and PQ, which may help explain the reduced result quality. Furthermore, unlike the other approaches, only one disk access is required per index even in the large-scale scenario of the following experiment.

This comparison shows that the NV-tree and IMI offer quite different trade-offs between retrieval performance and quality. IMI requires more work to i) discover which cells to read, ii) read the cells, and iii) post-process the results. At a small scale, however, such as with this small SIFT1B collection, IMI does indeed offer a good trade-off between quality and retrieval time. As the scale of the collections grows, however, this trade-off becomes less and less viable.

5.3 Scalability Analysis

It is insightful to determine the disk space requirements and expected disk read performance if PQ and IMI were used to index the 28.5B features used in the

previous experiments. PQ uses at least 32 bytes per feature, so indexing 28.5B features would require close to 1TB of memory. Product quantization must scan a substantial number of cells in order to return high quality results—typically 16 cells per query feature, with each cell requiring at least one random disk read—which amounts to a few hundred thousands points at least; clearly impossible in a disk-based setting.

IMI improves on PQ in a main-memory situation, and can obtain reasonable quality by examining only between 10 thousand and 100 thousand data points. Their most compact proposal uses only 12 bytes per feature (4 for the identifier and 8 bytes for information used for improving quality). With these settings, indexing the 28.5B collection in memory would require at least 320 GB of main memory (ignoring all overheads). In a disk-based setting, however, the key question is how many cells would be read, as each cell requires a random disk read. Unfortunately, [3] gives no information on the cell size distribution. As they use $2^{14} \times 2^{14}$ cells, however, each cell would contain on average just over one hundred data points, and 95 cells would need to be read on average to retrieve 10,000 candidates, and about 950 to retrieve 100,000 candidates. Results from [4] indicated that only about half of these cells are empty, so about 50 cells would need to be read on average to retrieve 10,000 candidates, and almost 500 cells to retrieve 100,000 candidates. Note that we have used the settings for a 1B feature collection in this analysis; it is of course not clear that those settings would yield sufficient result quality with 28.5B features. What is clear, however, is that the response time of IMI would be unacceptable at this scale in an industry setting.

In contrast, as discussed below, while we typically index each collection using three NV-trees, only a single disk read is required per NV-tree and fewer than one thousand feature identifiers are typically considered per tree when constructing the approximate answer, independent of the scale of the collection.

6 Conclusion

The NV-tree is a scalable approximate high-dimensional indexing method specifically designed for visual instance search (VIS) at large scale. In this paper, we have reported on three experiments designed to evaluate the suitability of the NV-tree for VIS applications. Two of these experiments embed VIS benchmarks from the literature within collections of up to 28.5B high-dimensional features, which is the largest single-server collection reported in the literature. The results show that indeed the NV-tree performs very well for VIS applications over large-scale collections. With the small-scale non-VIS benchmark, the NV-tree has better performance but lower result quality, but our analysis shows that query processing performance of PQ and IMI would suffer with the 28.5B collection.

The NV-tree index is a proven technology, as it is already in use at Videntifier Technologies, one of the main players in the forensics arena with technology deployed at such clients as Interpol. Their search engine targets fine-grained VIS for investigations that, e.g., aim to dismantle child abuse networks. The search engine can identify very fine-grained details in still images and videos

from a collection of 150 thousand hours of video, typically scanning videos at 40x real-time speed, and about 700 hours of video material can be dynamically added to the index every day.

References

1. Amsaleg, L.: A database perspective on large scale high-dimensional indexing. Habilitation à diriger des recherches, Université de Rennes 1 (2014)
2. Babenko, A., Lempitsky, V.S.: The inverted multi-index. In: Proc. CVPR. Providence, RI, USA (2012)
3. Babenko, A., Lempitsky, V.S.: The inverted multi-index. IEEE Transactions on Pattern Analysis and Machine Intelligence 37(6), 1247–1260 (2015)
4. Babenko, A., Lempitsky, V.S.: Efficient indexing of billion-scale datasets of deep descriptors. In: Proc. CVPR. Las Vegas, NV, USA (2016)
5. Douze, M., Jégou, H., Sandhawalia, H., Amsaleg, L., Schmid, C.: Evaluation of gist descriptors for web-scale image search. In: Proc. CIVR. Santorini, Greece (2009)
6. Guðmundsson, G.Þ., Amsaleg, L., Jónsson, B.Þ., Franklin, M.J.: Towards engineering a web-scale multimedia service: A case study using Spark. In: Proc MMSys. Taipei, Taiwan (2017)
7. Jégou, H., Tavenard, R., Douze, M., Amsaleg, L.: Searching in one billion vectors: re-rank with source coding. In: Proc. ICASSP. Prague, Czech Republic (2011)
8. Lejsek, H., Ásmundsson, F.H., Jónsson, B.Þ., Amsaleg, L.: NV-tree: An efficient disk-based index for approximate search in very large high-dimensional collections. IEEE Transactions on Pattern Analysis and Machine Intelligence 31(5), 869 – 883 (2009)
9. Lejsek, H., Jónsson, B.Þ., Amsaleg, L.: NV-Tree: Nearest neighbours at the billion scale. In: Proc. ACM ICMR. Trento, Italy (2011)
10. Liu, T., Moore, A., Gray, A., Yang, K.: An investigation of practical approximate nearest neighbor algorithms. In: Proc. NIPS. Vancouver, BC, Canada (2004)
11. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. International Journal on Computer Vision 60(2), 91–110 (2004)
12. Moise, D., Shestakov, D., Guðmundsson, G.Þ., Amsaleg, L.: Indexing and searching 100M images with Map-Reduce. In: Proc. ACM ICMR. Dallas, TX, USA (2013)
13. Petitcolas, F.A.P., Steinebach, M., Raynal, F., Dittmann, J., Fontaine, C., Fates, N.: A public automated web-based evaluation service for watermarking schemes: StirMark benchmark. In: Proc. Electronic Imaging, Security and Watermarking of Multimedia Contents III. San Jose, CA, USA (2001)
14. Sun, X., Wang, C., Xu, C., Zhang, L.: Indexing billions of images for sketch-based retrieval. In: Proc. ACM Multimedia. Barcelona, Spain (2013)