Suitability of Nearest Neighbour Indexes for Multimedia Relevance Feedback

Omar Shahbaz Khan[1][0000−0001−9720−3645],
Martin Aumüller[2][0000−0002−7212−6476], and
Björn Þór Jónsson[1][0000−0003−0889−3491]

[1] Reykjavik University, Reykjavik, Iceland,
{omark,bjorn}@ru.is
[2] IT University of Copenhagen, Copenhagen, Denmark,
maau@itu.dk

Abstract. User relevance feedback (URF) is emerging as an important component of the multimedia analytics toolbox. State-of-the-art URF systems employ high-dimensional vectors of semantic features and train linear-SVM classifiers in each round of interaction. In a round, they present the user with the most confident media items, which lie furthest from the SVM plane. Due to the scale of current media collections, URF systems must be supported by a high-dimensional index. Usually, these indexes are designed for nearest-neighbour point queries, and it is not known how well they support the URF process. In this paper, we study the performance of four state-of-the-art high-dimensional indexes in the URF context. We analyse the quality of query results, compared to a sequential analysis of the collection, over a range of classifiers, showing that result quality depends (i) heavily on the quality of the SVM classifier and (ii) the index structure itself. We also consider a search-oriented workload, where the goal is to find the first relevant item for a task. The results show that the indexes perform similarly overall, despite differences in their paths to the solution. Interestingly, worse recall can lead to better application-specific performance.

Keywords: High-Dimensional Indexing · Interactive Learning · User Relevance Feedback · Multimedia Retrieval

1 Introduction

In user relevance feedback (URF), the goal is to train interactive classifiers to satisfy specific information needs based on direct feedback from the user. When interacting with a multimedia collection, the user is presented in each interaction round with a set of items from the collection and asked to judge some items as relevant and some items as non-relevant for a specific task. At the start of the URF process, the items are typically randomly sampled from the collection or retrieved using a query, but once the initial classifier has been trained the items are usually selected from the top items returned by the current version of the classifier. This interactive process continues until the user's information need is satisfied or they determine that the collection holds no items of interest. As URF allows users to express and refine fuzzy information needs, it is an important component of the multimedia analytics toolbox.

A state-of-the-art multimedia URF system, such as [12], is implemented as follows. First, the multimedia items are described by semantic labels, which are produced by advanced deep-learning models and compressed using an index-based compression technique. Second, such a system builds linear-SVM classifiers, which are known to work well with few examples, and presents users in each interaction round with the multimedia items that (a) are furthest away from the resulting hyperplane, and (b) have not been seen before in the process. And third, a high-dimensional index is used to speed up the retrieval and ensure a stable response time. While the state-of-the-art has shown URF to work at scale using these elements, such furthest neighbour queries from an SVM-based hyperplane have not been studied much in the literature. In particular, to the best of our knowledge these evaluations have not been carried out in the context of URF. This paper therefore opens an investigation into the suitable choice of such a high-dimensional index for URF over multimedia collections.

Indexing high-dimensional data to support similarity queries such as finding (approximate) nearest or furthest neighbors suffer the *curse of dimensionality.* In general, this means that there are no *sublinear* time algorithms that solve these tasks *exactly* on *arbitrary data*, and a linear scan through the dataset is the best one can hope for. However, if a small loss in accuracy can be accommodated or if the data is "favorable", a large collection of scalable solutions for finding nearest neighbors is available. A solution is either provided with strong theoretical guarantees, such as *hashing-based* approaches [9] with their theoretical time guarantees, or with strong empirical evidence on the quality of the query result, such as *cluster-based* [11], *graph-based* [10, 14], or *tree-based* [5] approaches. The ANN benchmarking effort [1] summarizes these approaches and demonstrates that, in practice, nearest neighbor search tasks on million-scale datasets can be solved several orders of magnitude faster than a linear scan with little loss in accuracy.

1.1 Problem Definition

We formulate the SVM-based hyperplane setting used in URF as follows. First, the distance between a point $p \in \mathbb{R}^d$ and a hyperplane $q \in \mathbb{R}^{d+1}$ is

$$d_{P2H}(p, q) = \frac{q_{d+1} + \sum_{i=1}^{d} p_i q_i}{\sqrt{\sum_{i=1}^{d} q_i^2}} \ .$$

Second, the problem of finding furthest points from a given hyperplane in the positive direction is equivalent to finding the points in the dataset with largest (positive) distance to the hyperplane. To avoid an exhaustive scan through the dataset, the task is to build an *index* data structure over the point set $S \subseteq R^d$ that supports these furthest point queries.

Point-to-Hyperplane queries are challenging because d_{P2H} is not a distance measure in the strict sense. However, we can reduce the problem to an inner product space as follows: Append a 1 to each point in the dataset, so that both points and hyperplanes have $d+1$ dimensions, and notice that—since the hyperplane is fixed at query time—finding the furthest neighbors according to d_{P2H} is equivalent to finding vector p that *maximizes* the *inner product* (MIPS) $d'(p, q) = \sum_{i=1}^{d+1} p_i q_i$. Solving maximum inner product search has seen significant progress in the research community and arises in particular in recommender systems [2]. The standard approach involves asymmetric transformations of data and query points [2, 20]. After such a transformation, finding points that maximize the inner product becomes equivalent to finding nearest neighbors in the transformed space, which is usually Euclidean distance. However, these transformations usually lower the contrast between points. For example, Huang et al. [8] consider transformations for hashing-based *closest point to hyperplane* queries and experimentally show that finding *furthest neighbors* instead of *nearest neighbors* (under slightly different transformations) provides empirical speed-ups. In the context of graphs, Morozov and Babenko [16] show that the transformed vectors produce *worse graph indexes* than using the inner product directly.

1.2 Contributions

The current state-of-the-art large-scale URF approach, Exquisitor [12], uses the high-dimensional ANN index eCP (extended Cluster Pruning) [15]. The stated reasons for this choice are its comprehensibility, time guarantees, and ability to work with hyperplane queries using MIPS. Based on practical advances in nearest neighbor search, we evaluate the suitability of state-of-the-art high-dimensional indexing approaches for URF over multimedia collections. In particular, we inspect three diverse approaches—Annoy (Approximate Nearest Neighbor Oh Yeah) [5], IVF (Inverted File Index on k-means clustering) [11], and HNSW (Hierarchical Navigable Small World) [14]—that perform well on million-scale nearest neighbor search with regards to supporting maximum inner product queries. We evaluate these indexes along with the eCP index using an automated evaluation protocol, based on the Lifelog Search Challenge 2019 dataset [7], to simulate URF sessions with the goal of finding one relevant item. All source code for the URF evaluation is made available on GitHub[1] to provide the research community with an experimental pipeline to compare different high-dimensional indexes for URF. In the process, we make the following contributions:

- All evaluated indexes show adequate quality towards hyperplane queries in terms of recall, with HNSW achieving the best overall performance.
- Indexes that introduce variety, due to build quality or search approach, are better at solving URF tasks (eCP and Annoy). Thus, high recall does not directly translate to being the best at solving the actual URF tasks. This relates to hyperplane queries being refined throughout a URF session, so the quality of initial queries may not be well defined for the task.

[1] https://github.com/Ok2610/urf-indexing-eval

– Finally, we analyze the effects of the approximation of each index. We find that eCP and IVF have a more comprehensible parameter for this than HNSW and Annoy, and eCP's setting has the least variability leading to better time estimates.

2 User Relevance Feedback

Analytical tasks for multimedia focus on discovering knowledge from the media items that reside within the ever growing multimedia collections of today. To truly uncover this knowledge it is essential to explore and search through the contents of such collections in real time. While long-running machine tasks may be capable of categorising and summarising parts of the collection for a task, typically as new knowledge is discovered the goal of the task can shift. In such situations, user relevance feedback is preferred as it allows the user to shift the classifier based on the new knowledge [22].

The most common approach to URF is to present a suggestion set S to the user, using the current classifier C. From S the user labels p items as positives and n items as negatives, which are then used to update C. There are multiple ways to determine which items to include in S. In URF the most confident items of C are presented. This is beneficial when the intention is not solely on creating a strong classifier, as the information need may not be entirely clear and may be susceptible to change throughout the session. With URF, the user may explore the collection or be more search-oriented, depending on how much refinement is put towards the classifier.

URF for content-based retrieval has been around for several decades [18, 23, 4, 17], but as multimedia collections started to rapidly expand, it became cumbersome at such scale due to the response time. Even prior to the scale issues, it was difficult to have explainable classifiers due to the feature representations of the multimedia items. Comprehension is important for the user to better understand the effects of their actions on the classifier. With improvements in deep learning, machines have become much better at discovering semantic features in multimedia contents [6, 3], making it a preferred choice of feature for URF applications. Semantic features extracted through deep nets result in sparse high-dimensional vectors. The current state of the art large-scale URF approach, Exquisitor, uses the high-dimensional feature vectors in a compressed representation together with the clustering-based ANN index eCP. The compressed representation selects the top f features and stores them in a space-efficient representation. Note that the compressed representation does not transform the feature space to ensure it remains comprehensible. The eCP index has been modified such that it works with this compressed representation. Furthermore, the index handles hyperplane queries from Exquisitor's Linear-SVM classifier [12]. In addition to the basic URF scheme, Exquisitor also employs incremental retrieval, which continues a search within the index, in case not enough items are found to be returned. This is linked to the search expansion parameter b in the eCP index, which is the number of clusters it needs to retrieve. Exquisitor has been shown capable of working with YFCC100M, a multimedia collection with nearly 100 million items, achieving subsecond response time with modest computing resources [12].

3 High-Dimensional Indexing

There exists a plethora of different approaches for solving nearest neighbor search queries. The most successful approaches can be categorized into *clustering-based*, *graph-based*, *hashing-based*, and *tree-based* approaches. For our practical evaluation, we pick approaches from each category which have not been considered for URF. We exclude locality-sensitive hashing based approaches because they were part of an earlier evaluation [12] and were shown to be inferior to using the eCP index. We review the considered approaches next.

Cluster-based approaches (IVF [11], eCP [15]). Given a dataset $S \subseteq \mathbb{R}^d$ and two parameters k and ℓ, run a clustering algorithm such as k-means to find k centroids. By associating each point with its closest centroid, the space is partitioned into k parts. The data structure that stores the centroids and the associated lists is referred to as an inverted file index (IVF). To find nearest neighbors to a query $q \in \mathbb{R}^d$, inspect the points associated to the ℓ closest centroids to q, possibly indexing the centroids for large k. The eCP index uses the same approach but uses the k initial random points as centroids to find a balanced space partition. Furthermore, eCP builds a hierarchy using the centroids.

Graph-based approaches (HNSW [14]). Given a dataset $S \subseteq \mathbb{R}^d$ and parameters k, ℓ, the goal is to build a graph $G = (V, E)$, where each point is represented by a vertex and edges exist between a point and a "diverse" set of at most k close points. Let us assume that such a graph G is given. To find the nearest neighbors of a query point q, HNSW uses a hierarchy of graphs to find a good entry point into the bottom-layer graph that indexes all points. Given such a start point, carry out a greedy hill climbing. In each round, consider the currently closest point to the query not considered before. Inspect the neighborhood and compute the distances to the query point. After each round, trim the list of current closest points (inspected and non-inspected) to ℓ, which is usually called the beam width. Terminate if all ℓ points have been considered. (Note that this is not a bound on the number of distance computations, since considered points might be trimmed.) To build the graph, order all the points and insert them one-by-one using the search algorithm¸ often with a smaller ℓ' than used for the queries. From the points inspected in this search, a pruned set of k points is chosen as neighbors of the inserted point (pruning might be necessary for its neighbors if the degree bound k is not met). There exist many other graph-based indexes that change details of this construction [10, 21].

Tree-based approaches (Annoy [5]). Annoy builds a collection of trees based on random projections. Given a set of points S and two parameters k, ℓ, the data structure works as follows. First, a node in a tree is described by a hyperplane a that splits up a point set $S' \subseteq S$. For example, one can find the median inner product of the data points with a and split S' into two balanced subsets based on that. At the root, the whole dataset is taken into consideration, and a leaf is created as soon as the number of points at a node is below a certain threshold. Instead of a single tree, k trees are created to boost the quality of the results. Given a query point and a collection of k trees, carry out root-to-leaf-traversals in each tree for the query. Traverse the trees upwards until ℓ (unique) points are found and return the closest among these points as the result of the search.

4 Evaluation

Evaluating user relevance feedback approaches is best done through real user tests. As we are attempting to gauge the performance of hyperplane queries on different indexes in a preliminary URF setup, however, we do not need real users at this stage. Instead we employ an automated evaluation protocol, based on real-life applications [13]. The evaluation protocol consists of tasks where the objective is to find the first relevant item within the tasks relevant item set, R_t. Typically, in an automated URF session, a set number of positives and negatives are considered from the suggestion set S in each round. Based on the labelling policy used in the protocol, positives and negatives are added or replaced from their respective global sets to train the linear-SVM classifier. Positives and negatives are labelled by comparing the distances between the combined maximum feature vector of R_t and the feature vector of items in S.

Based on [13], we design the evaluation protocol with the following parameter choices; rd number of rounds in each URF session, k number of items to retrieve, s the number of suggestions to consider, p the number of positives to label and n the number of negatives to label, P and N the positive and negative sets used to train the linear SVM. rd is set to 50 to simulate a long-running session. s is set to 25, which typically would also be the value for k. Items seen in previous rounds of a URF session should not be presented to the user again, however, and in situations where all k items are seen the session ends. When the underlying URF approach is able to remove previously seen items internally to avoid returning them again, then we would set $k = s$. Since this is not the case for the tested indexes, however, the indexes are asked to return a higher number $k = 1000$ of items, from which the previously seen items are then pruned. The labelling policy of the evaluation protocol is *AccRep*, where in each round p positives and n negatives are added to P and N. If better items exists in the remaining suggestions for either P or N, they replace the weakest items in the sets [13]. In our evaluation $p = 5$ and $n = 15$ leading to 5-250 positives and 15-750 negatives for each session, with each session presenting a total of 1250 $(s \cdot rd)$ items.

Dataset. The dataset used in the evaluation is from the Lifelog Search Challenge 2019. Lifelogging is the principal of recording your daily life with as much data as possible, i.e. logging biometric data, taking images throughout the day, food logs, and more. Pure lifeloggers often walk around with a miniature camera on their person that takes an image at a set interval, and thus they accumulate a large multimedia collection. The Lifelog Search Challenge (LSC) is a live interactive

Index	Build Parameters	Search Parameter
eCP	L: 3, $cSize$: 100	b: 64
Annoy	n_{trees}: 100	$search_k$: 10000
HNSW	M: 48, ef_C: 500	ef_S: $\max(10, k)$
IVF	n_{list}: 417	n_{probe}: 64

Table 1: Build and search parameters for each index.

retrieval challenge, where tasks are defined over a snippet of the extremely large collection. The LSC 2019 dataset represents one lifelogger's daily life across 1 month, consisting of 41,666 images. There are 24 retrieval tasks defined over this collection [7]. Semantic feature labels have been extracted using a deep net, with the top 7 features being used and the rest set to zero. At LSC multiple multimedia retrieval systems attempt to solve the tasks one by one within a time limit. A task consists of a text query, but unlike regular search challenges where the entire text is given at once, in LSC the text query is presented in parts. Every 30 seconds new information is added to the presented text of the current task. In the evaluation protocol this aspect is reflected when filters are used, but as our focus here is on hyperplane queries, this aspect is ignored.

Index Parameters. We use Annoy v1.17.2, HNSW from hnswlib v0.7.0 and IVF from Faiss v1.7.4. The build parameters for each index used in the experiments can be seen in Table 1. The choice of the build parameters for Annoy and HNSW are based on their settings for datasets of relevant sizes from ANN-benchmarks [1]. The eCP index it has two build parameters; L the level of its hierarchy and $cSize$ the number of items in each cluster. Note that the latter is a soft enforcement, so there is still a chance of clusters being larger or smaller. We aim to have clusters with 100 items in a 3-level hierarchy where L_l has $\sqrt{L_{l+1}}$ clusters, with $L = 3$ having all clusters. The IVF index has a single build parameter n_{list} which specifies the number of clusters to divide the items into. We select this to be similar to eCP's clusters at $L = 3$. Each index has a runtime approximation parameter that is set according to their recommendations. The effects of this parameter is crucial to understand in terms of distance computations performed, as it indicates whether an index is able to provide stable response time guarantees. All indexes were built using Euclidean distance.

4.1 Experiment 1: Hyperplane Queries

In the first experiment we investigate the recall of each index when encountering hyperplane queries. The hyperplane queries for this experiment are generated by running the evaluation protocol on the LSC dataset, leading to 1200 hyperplane queries in total (24 tasks with URF sessions of 50 rounds). The groundtruth for this experiment are the top 1000 items for each hyperplane under maximum inner product similarity obtained by an exact linear scan. Here, recall is the fraction of the k items returned by the implementation that belong to the true

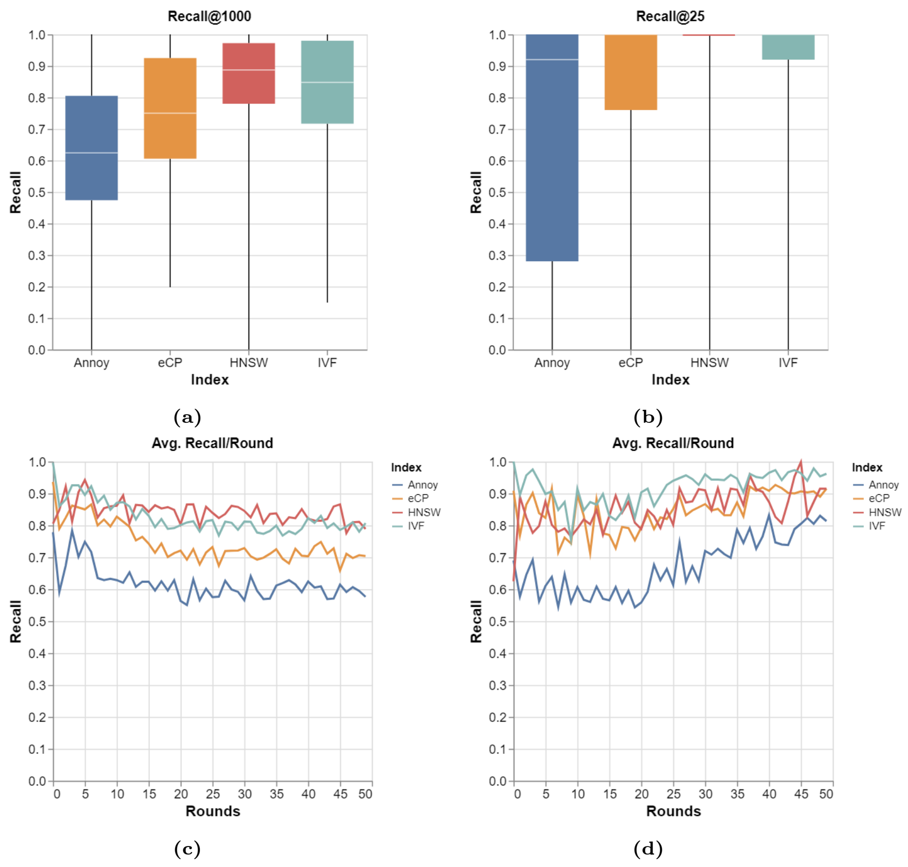

Fig. 1: Recall distribution @1000(a,c) and @25(b,d) for hyperplane queries and average recall per round. Annoy (blue), eCP (orange), HNSW (red), IVF (teal).

top k items (groundtruth) with largest inner product for a given hyperplane. The results from this experiment are depicted in Figure 1. Figure 1a shows the recall distribution for the top 1000 items. The best-performing index is HNSW with a consistent distribution above 75% recall. The IVF index is close but has a slightly lower recall overall. eCP fares worse than IVF, which shows the extra effort in constructing the clusters is beneficial for recall, but not by much. Annoy has a similar distribution to eCP, but generally 10% lower recall on average. Figure 1b shows the recall distribution for the top 25 items, or the items that would actually be presented to the user. Here we see an increase in recall overall for all the indexes. HNSW remains at top, achieving seemingly 100% apart from some outliers. IVF is slightly worse with eCP following. Annoy has a much wider distribution, but a high median above 90% recall. Thus, the items the user sees are high-recall items for the hyperplane queries regardless of the index.[2] Figures 1c and 1d depict the average recall across all tasks per round in the URF session. As more rounds pass in a session, the hyperplane queries should become more descriptive. For the top 1000 items all indexes follow a similar pattern of starting with a high recall that falls during the initial rounds, and then settles at a lower average recall. For the top 25 items the pattern starts similarly with a drop off, but as the hyperplane queries become more refined the recall for all indexes increase. Given that the top 25 items are the ones the user sees, this behavior is desired.[3]

4.2 Experiment 2: User Relevance Feedback

From the previous experiment, it is clear that HNSW performs best in the case of recall for hyperplane queries while Annoy performs worst. When it comes to URF tasks, however, the theoretical top items for a query may not necessarily be items of interest, especially in early rounds when the hyperplane still needs to be refined. We consider two scenarios. In Scenario 1, the indexes are provided with the hyperplanes from using an exact linear scan in the evaluation protocol. In Scenario 2, the linear-SVM is trained from the individual results of an index.

Table 2 shows the round where each index managed to find the first relevant item in the top 25 in Scenario 1, where Scan represents the results from the linear scan. Scan solves the most tasks (14 out of 24), while the indexes solve 10 tasks each except for Annoy which solves 11. With Annoy solving 1 more task it shows that high recall for majority of hyperplane queries in a URF session is not always necessary. Looking at the solved tasks, none of the indexes complete a task that is not also solved by Scan, and they either solve it in the same round or 1 round after it. Notable exceptions, are tasks 11, 20 and 23 where Scan solves them 5-11 rounds earlier, and task 14 where eCP and HNSW solve the task 7 rounds prior to Scan. From these results we see how the restriction on the indexes require more rounds to solve a task, while task 14 shows that having the entire collection available can also introduce noise.

We now turn to Scenario 2. As the indexes employ a structure and approximations on the collection, a suggestion set from the same hyperplane can differ, and from that point they have different hyperplane queries throughout the session. This is depicted in Table 3 where each index has run the evaluation protocol using hyperplane queries generated from their own suggestions. By using queries defined through their own sessions they solve more and different tasks, and for the tasks they solve in common there are larger gaps between the rounds. Here, Annoy solves 15 tasks and the others solve 14. It is also worth noting that Annoy, eCP and IVF solve some tasks that Scan could not, which again indicates the larger scope of the full scan encountering noise. We further test the assumption of

[2] This experiment was also conducted using Annoy, HNSW, and IVF built using inner product instead of Euclidean distance. In all cases, average recall @1000 was lower, while for HNSW recall @25 was improved.

[3] Similar results are observed when (roughly) targeting a certain number of distance computations across all indexes.

Task	Scan	Annoy	eCP	HNSW	IVF
0	**33**	34	34	34	34
1	-	-	-	-	-
2	-	-	-	-	-
3	-	-	-	-	-
4	-	-	-	-	-
5	-	-	-	-	-
6	**12**	**12**	**12**	**12**	**12**
7	-	-	-	-	-
8	**23**	24	24	24	**23**
9	**2**	**2**	**2**	**2**	**2**
10	-	-	-	-	-
11	**25**	32	32	32	32
12	**19**	20	**19**	20	**19**
13	**32**	-	-	-	-
14	11	11	**4**	**4**	12
15	10	10	10	10	10
16	**36**	37	**36**	37	37
17	-	-	-	-	-
18	**3**	-	-	-	-
19	**23**	-	-	-	-
20	**20**	25	-	-	-
21	-	-	-	-	-
22	-	-	-	-	-
23	**4**	15	15	15	15
Solved	14	11	10	10	10
Best	12	2	5	3	4

Table 2: First round in the URF session where a relevant item for the task was found using the hyperplanes generated for the entire dataset D.

Task	Annoy	eCP	HNSW	IVF
0	-	-	**33**	47
1	**25**	-	-	-
2	-	-	-	-
3	-	**35**	-	-
4	**33**	-	-	43
5	-	-	-	-
6	20	**6**	22	12
7	-	-	-	-
8	9	17	18	**7**
9	**2**	**2**	**2**	**2**
10	-	-	-	-
11	**20**	40	32	43
12	39	-	**17**	20
13	33	27	**24**	-
14	18	15	**11**	17
15	**6**	8	7	10
16	29	11	37	**9**
17	-	-	-	-
18	12	5	8	**3**
19	**9**	18	23	30
20	16	18	17	**15**
21	-	-	-	-
22	-	**32**	-	-
23	17	8	**4**	14
Solved	15	14	14	14
Best	6	4	6	5

Table 3: First round in the URF session where a relevant item for the task is found using the hyperplanes generated from each index.

hyperplanes from an index' own session being best, by running the hyperplanes from one index with another index. These results show that while they solve the same tasks, the rounds for many of the tasks differ, ranging from a few rounds to 20+. HNSW with IVF's hyperplanes show the most similar performance. We have omitted the table for these results to not exceed the article length.

Overall, from these results we have shown that all indexes, with their recommended settings, are capable of dealing with URF tasks. There is no clear indication for which index is best; while Annoy solves the most tasks, there are still tasks that are solved faster with the other indexes. A point of interest now is the approximation parameter for each index which determines the number of items each index considers or the number of distance computations taking place. With the recommended settings Annoy has the lowest number of average distance computations with ∼4300, while HNSW has the highest with ∼9400.

Task	Annoy	eCP	HNSW	IVF
0	-	-	-	**47**
1	**33**	-	-	-
2	-	-	-	-
3	-	**35**	-	-
4	-	-	-	**43**
5	-	-	-	-
6	26	**6**	16	12
7	-	-	-	-
8	11	17	-	**7**
9	**2**	**2**	**2**	**2**
10	-	-	**47**	-
11	**23**	40	27	43
12	31	-	**15**	20
13	31	**27**	34	-
14	18	15	**11**	17
15	**6**	8	7	10
16	-	11	46	**9**
17	-	-	-	-
18	12	5	8	**3**
19	**7**	18	12	30
20	**15**	18	17	**15**
21	-	-	-	-
22	-	**32**	-	-
23	-	**8**	34	14
Solved	12	14	13	14
Best	6	6	4	7

Table 4: Results from similar search scope (∼6400).

Task	Annoy	eCP	HNSW	IVF
0	**20**	-	-	31
1	29	**4**	-	-
2	-	-	-	-
3	-	-	-	**37**
4	42	**33**	-	-
5	**45**	-	-	-
6	17	**8**	48	15
7	-	-	-	-
8	20	**7**	36	**7**
9	**2**	**2**	**2**	**2**
10	-	**41**	-	-
11	27	**15**	26	-
12	22	**11**	30	23
13	27	33	**26**	-
14	22	15	**11**	17
15	7	8	**6**	10
16	**7**	-	-	9
17	-	-	-	-
18	7	8	7	**3**
19	10	22	**8**	23
20	**16**	24	22	20
21	-	-	-	-
22	-	**38**	44	-
23	30	**5**	8	**5**
Solved	17	16	13	13
Best	4	10	5	4

Table 5: Results from reduced search scope (∼3200).

Task	Annoy	eCP	HNSW	IVF
0	-	-	-	-
1	-	46	-	**31**
2	-	**47**	-	-
3	-	-	-	-
4	**18**	31	-	-
5	-	-	-	-
6	40	15	25	**14**
7	-	-	-	-
8	-	38	-	**10**
9	**2**	**2**	**2**	**2**
10	-	-	**41**	-
11	22	33	**10**	42
12	21	32	45	**20**
13	31	31	-	-
14	14	**10**	11	15
15	**6**	**6**	10	10
16	29	16	**13**	32
17	-	-	-	-
18	8	8	7	**3**
19	10	27	**8**	16
20	18	28	**16**	23
21	-	-	-	-
22	35	**30**	43	-
23	16	7	8	**5**
Solved	14	17	13	13
Best	6	4	6	7

Table 6: Results from reduced search scope (∼1600).

The approximation parameter is what introduces the quality/time trade-off and is often set based on the specific use case. The transparency of this parameter is better for some indexes than others. For IVF and eCP it is the b and n_{probe} parameter, which is how many clusters to consider during the search. In eCP b is used for each level in its hierarchy. Annoy uses the $search_k$ parameter, which is the number of binary trees it will search. HNSW uses the ef_S parameter, which is the number of candidates to consider while retrieving the top k items.

In Table 4 we compare the performance using settings for each index that result in around 6400 distance computations. For this HNSW's $ef_S = 700$ and for Annoy's $search_k = 14000$, while IVF and eCP remain the same ($b/n_{probe} = 64$), so the only changes worth noting in the table are for Annoy and HNSW. With these settings they both solve fewer tasks, 2 for Annoy and 1 for HNSW, and some tasks where they were the best require more round, leading to other indexes solving them faster or in the same round. For Annoy and HNSW to have the

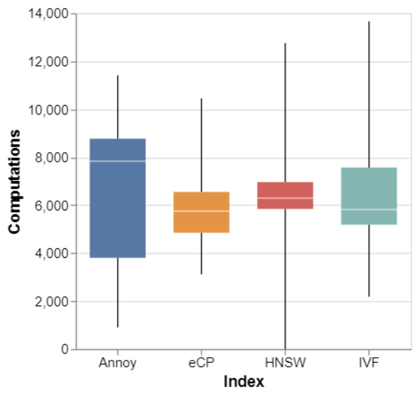

Fig. 2: Distribution of distance computations for the indexes, when the approximation was aimed to be ∼6400.

same distance computations, we had to increase Annoy's search parameter while reducing HNSW's. To investigate the effects of a lower scope further, we run the evaluation protocol again with distance computations roughly around 3200 and 1600 for all indexes, depicted on Table 5 and 6 respectively. When reducing the scope eCP solves more tasks and is seemingly faster than with the higher scope. Annoy improves in terms of tasks solved with 17 at scope 3200, but at 1600 it solves 14. HNSW solves the same number of tasks for both reduced scopes, but not always the same tasks, which hints that certain tasks are better with a lower scope for HNSW and some are better with a larger one. IVF has similar behavior as HNSW when reducing scope. It should also be noted that the lowest scope runs for IVF and HNSW also lead to tasks finishing before the 50 rounds as all items returned had been seen in previous rounds. Fortunately, the relevant item was found in a previous round for those tasks, but in case this occurs earlier, some notion of incremental retrieval that can expand the search within the index is needed. This feature exists in eCP when used with Exquisitor.

5 Discussion

In this section we discuss the insights gained from the experiments and the role of approximation parameters for each index. In Figure 2 the distribution of distance computations is depicted for each index, where the average distance computation was around 6400. Annoy with $search_k = 14000$ has the highest variability, fluctuating between 4000 and 8000 distance computations. eCP, HNSW and IVF are more consistent[4] and closer to the average target.

Our experiments highlight that each index is able to solve URF tasks on the small LSC dataset, with eCP and Annoy being better than HNSW and IVF.

[4] The 0-valued outliers for HNSW stem from URF sessions stopping early, as everything returned has already been seen, while the actual minimum was around 4700.

The LSC dataset contains many near-duplicate images, as the dataset depicts the daily life of one person. To get a better picture with a more general dataset and similar tasks, we conducted an experiment on the dataset from Video Browser Showdown 2020 [19] (VBS) consisting of 1 million images. Solving tasks is more difficult in VBS as there are more scenarios to cover, and typically filters are applied to help with the task. With pure URF both Annoy and eCP manage to solve 2-3 tasks out of the 12, while HNSW and IVF solve 1. Similarly to the LSC collection, eCP and Annoy performed best with a lower scope ($\sim$3200). However, given that Annoy still fluctuates between 1000 and 8000 distance computations, eCP remains the better overall choice. HNSW and IVF at similar scope did not manage to solve any task, and HNSW even had multiple sessions stopping early due to all returned items being seen. This is the danger of a small search scope, and is why having an easy to comprehend and reliable approximation parameter is extremely beneficial. With eCP and IVF one can reliably ask for additional b/n_{probe} clusters, knowing the computation time will be roughly the same. However, for Annoy and HNSW this is more difficult.

6 Conclusion

In this paper we investigated the performance of multiple state-of-the-art ANN indexes in user relevance feedback (URF) settings dealing with hyperplane queries. We evaluated 4 indexes, the tree-based approach Annoy, the graph-based approach HNSW, and the cluster-based approaches IVF and eCP. Each of these high-dimensional indexes use some form of approximation that introduces a quality/time trade-off. In interactive URF sessions, fast and reliable response time is crucial. Through our experiments using an automated evaluation protocol simulating URF sessions, we find that each index is able to solve such tasks. We also discovered that a lower setting for the approximation parameter, which reduce the search space, can improve results. However, if it is set too low, the index may not find any new items to present the user. Out of the four, eCP and Annoy perform best overall. We further analyze the approximation parameters of the indexes and find eCP's parameter to be more comprehensible and reliable. The other indexes are still viable for URF on a small scale collection, but it is harder to predict their performance when used at scale. Following up with real user tests and conducting experiments on even larger collections is warranted, to better verify these findings.

Acknowledgements: This work was supported by Icelandic Research Fund grant 239772-051 and by the Innovation Fund Denmark for the project DIREC (9142-00001B).

References

1. Aumüller, M., Bernhardsson, E., Faithfull, A.J.: ANN-Benchmarks: A Benchmarking Tool for Approximate Nearest Neighbor Algorithms. Inf. Syst. **87** (2020)

2. Bachrach, Y., Finkelstein, Y., Gilad-Bachrach, R., Katzir, L., Koenigstein, N., Nice, N., Paquet, U.: Speeding up the Xbox Recommender System using a Euclidean Transformation for Inner-Product Spaces. In: RecSys. pp. 257–264. ACM (2014)
3. Barraco, M., Cornia, M., Cascianelli, S., Baraldi, L., Cucchiara, R.: The Unreasonable Effectiveness of CLIP Features for Image Captioning: An Experimental Analysis. In: CVPR. pp. 4662–4670 (2022)
4. Bartolini, I., Ciaccia, P., Waas, F.: FeedbackBypass: A New Approach to Interactive Similarity Query Processing. In: VLDB. pp. 201–210 (2001)
5. Bernhardsson, E.: Annoy, https://github.com/spotify/annoy
6. Dubey, S.R.: A Decade Survey of Content Based Image Retrieval Using Deep Learning. IEEE TCSVT **32**(5), 2687–2704 (2021)
7. Gurrin, C., Schoeffmann, K., Joho, H., Leibetseder, A., Zhou, L., Duane, A., Nguyen, D., et al.: Comparing Approaches to Interactive Lifelog Search at the Lifelog Search Challenge (LSC2018). ITE TMTA **7**(2), 46–59 (2019)
8. Huang, Q., Lei, Y., Tung, A.K.H.: Point-to-Hyperplane Nearest Neighbor Search Beyond the Unit Hypersphere. In: SIGMOD. pp. 777–789. ACM (2021)
9. Indyk, P., Motwani, R.: Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In: STOC'98. pp. 604–613 (1998)
10. Iwasaki, M., Miyazaki, D.: Optimization of Indexing Based on k-Nearest Neighbor Graph for Proximity Search in High-dimensional Data. ArXiv e-prints (Oct 2018)
11. Johnson, J., Douze, M., Jégou, H.: Billion-Scale Similarity Search with GPUs. IEEE TBD **7**(3), 535–547 (2021)
12. Khan, O.S., Jónsson, B.Þ., Rudinac, S., Zahálka, J., Ragnarsdóttir, H., Þorleiksdóttir, Þ., Guðmundsson, G.Þ., Amsaleg, L., Worring, M.: Interactive Learning for Multimedia at Large. In: ECIR. pp. 495–510. ECIR'20, Springer (2020)
13. Khan, O.S., Jónsson, B.Þ., Zahálka, J., Rudinac, S., Worring, M.: Impact of Interaction Strategies on User Relevance Feedback. In: ICMR. p. 590–598. ICMR'21, ACM (2021)
14. Malkov, Y.A., Yashunin, D.A.: Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. IEEE TPAMI **42**(4), 824–836 (2020)
15. Moise, D., Shestakov, D., Gudmundsson, G., Amsaleg, L.: Indexing and Searching 100M Images with Map-Reduce. In: ICMR. pp. 17–24. ICMR'13, ACM (2013)
16. Morozov, S., Babenko, A.: Non-metric Similarity Graphs for Maximum Inner Product Search. In: NeurIPS. pp. 4726–4735 (2018)
17. Rui, Y., Huang, T.: Optimizing learning in image retrieval. In: CVPR. vol. 1, pp. 236–243 (2000)
18. Rui, Y., Huang, T.S., Ortega, M., Mehrotra, S.: Relevance Feedback: A Power Tool for Interactive Content-Based Image Retrieval. IEEE TCSVT **8**, 644–655 (1998)
19. Schoeffmann, K.: A User-Centric Media Retrieval Competition: The Video Browser Showdown 2012-2014. IEEE MultiMedia **21**(4), 8–13 (2014)
20. Shrivastava, A., Li, P.: Asymmetric LSH (ALSH) for Sublinear Time Maximum Inner Product Search (MIPS). In: NeurIPS. pp. 2321–2329 (2014)
21. Subramanya, S.J., Devvrit, F., Simhadri, H.V., Krishnaswamy, R., Kadekodi, R.: DiskANN: Fast Accurate Billion-point Nearest Neighbor Search on a Single Node. In: NeurIPS. pp. 13748–13758 (2019)
22. Zahálka, J., Worring, M.: Towards Interactive, Intelligent, and Integrated Multimedia analytics. In: IEEE VAST. pp. 3–12. IEEE (2014)
23. Zhou, X., Huang, T.: Relevance Feedback in Image Retrieval: A Comprehensive Review. Multimedia Systems **8**, 536–544 (2003)