

U-HNSW: An Efficient Graph-based Solution to ANNS Under Universal L_p Metrics

Huayi Wang
Georgia Institute of Technology
Atlanta, GA, USA
huayiwang@gatech.edu

Jingfan Meng
Georgia Institute of Technology
Atlanta, GA, USA
jmeng40@gatech.edu

Jun Xu
Georgia Institute of Technology
Atlanta, GA, USA
jx@cc.gatech.edu

Abstract

Approximate nearest neighbor search under universal L_p metrics (ANNS- $U-L_p$) is an important and challenging research problem, as it requires answering queries under all possible p ($0.5 \leq p \leq 2$) values simultaneously without building an index for each possible p value. The state-of-the-art solution, called MLSH, is a Locality-Sensitive Hashing (LSH)-based ANNS method with barely acceptable query performance. In contrast, graph-based ANNS methods, which offer significantly improved query efficiency on the ANNS- L_p problem (with a fixed p -value), cannot be naively extended to the ANNS- $U-L_p$ problem. In this paper, we propose U-HNSW, the first graph-based method for ANNS- $U-L_p$. Our scheme uses HNSW graph indexes built on two base metrics (L_1 and L_2) to generate promising NN candidates, and then verifies these candidates with an early-termination strategy. Experimental results show that U-HNSW not only achieves up to 2670 times shorter query time than the original MLSH implementation running on a RAM disk, but also outperforms the original HNSW on the ANNS- L_p problem, except for a few special p values.

Keywords

Approximate nearest neighbor search, High-dimensional similarity search, L_p distance, Graph-based index

1 Introduction

Approximate nearest neighbor search (ANNS) is a fundamental algorithmic problem arising in many areas of computer science, such as large language models [8, 28], computer vision [16], and information retrieval [20]. In these applications, data items in a dataset $\mathcal{D}$ are represented by vectors in a high-dimensional (say d -dimensional) space. Given a query point $\vec{q}$, the goal of ANNS is to find one or more points in $\mathcal{D}$ that are among the closest to $\vec{q}$ according to a certain distance metric, such as L_2 distance.

1.1 ANNS- $U-L_p$: Application and Challenges

In this work, we attack a fundamentally different variant of the ANNS problem. In this problem, we need to answer queries over a dataset $\mathcal{D}$ that lies in a d -dimensional space. However, unlike traditional ANNS formulations that assume a fixed distance metric, each query in our problem specifies its own distance metric parameter p . Specifically, the i -th query is a tuple $(\vec{q}_i, p_i)$, where $\vec{q}_i$ is the query vector and p_i determines the L_p metric used to measure distances. The objective is to find $\vec{x} \in \mathcal{D}$ that is closest to $\vec{q}_i$ under the L_{p_i} distance. Because the value of p_i can vary from one query to the next, we refer to this problem as ANNS under universal L_p metrics, or ANNS- $U-L_p$ for short.

While the L_2 metric is undoubtedly the most widely used and influential distance measure in ANNS [9, 19, 22, 23], other L_p

metrics are also important, as they can often yield more insightful and discriminative results in data mining, image retrieval, and genomic data search [13, 17, 29]. Accordingly, the ANNS problem under a general L_p metric (ANNS- L_p) has been studied extensively in the literature [7, 14]. However, an ANNS- L_p query engine built for a fixed value of p is often inadequate for many modern ML applications, where the most discriminative choice of p can vary across datasets or tasks and must be tuned per application [29]. For example, in K -NN classification, [29] shows that the optimal p differs across datasets: $p = 0.5$ achieves the highest accuracy on a physical signal dataset, $p = 0.6$ is optimal on an image dataset, while $p = 1.0$ is optimal on a text dataset. Since the best p cannot be known in advance and must be determined empirically, an efficient query engine that supports arbitrary L_p metrics without rebuilding the index is essential. An efficient query engine for ANNS under universal L_p metrics (ANNS- $U-L_p$) would be ideal for such applications, as it enables adaptive exploration of the dataset $\mathcal{D}$ under different L_p metrics—allowing each application to identify the most suitable, task-specific p value dynamically.

The ANNS- $U-L_p$ problem is particularly challenging because it demands high query efficiency and accuracy across a continuum of L_p metrics. A straightforward approach is to construct, for each possible p_i value, a dedicated ANNS index (e.g., using p -stable Locality-Sensitive Hashing [7]) to handle ANNS- L_{p_i} queries. However, this naive solution is impractical, as it would require building and maintaining a large number of separate indexes for the full range of p values in $[0.5, 2]$, even after discretization. For example, a coarse discretization of p at intervals of 0.1 (as used in prior work [21, 29]) would require 16 separate indexes.

1.2 Existing Solutions

LazyLSH [29] represents the first practical attempt to address the ANNS- $U-L_p$ problem without building multiple indexes. It is built upon the p -stable LSH framework [7] and aims to reuse a single index constructed for the L_1 metric (called the *base metric* in [29]) to answer queries under various L_p metrics for $0.5 \leq p \leq 1$. The key insight is that two points $\vec{x}$ and $\vec{y}$ that are close under the L_1 metric are also not too far under the L_p metric, provided that p is not too far from 1. However, as p deviates further from 1 (e.g., toward 0.5), this correlation between L_1 -closeness and L_p -closeness diminishes, leading to a noticeable degradation in query accuracy, as observed in [21].

Multi-Metric LSH (MLSH) [21], the state-of-the-art solution for the ANNS- $U-L_p$ problem, addresses this limitation by introducing an additional LSH index for the $L_{0.5}$ metric “at the other end.” Specifically, given an ANNS- $U-L_p$ query $(\vec{q}_i, p_i)$, MLSH selects which index to use based on the value of p_i : when p_i is closer to 0.5 than to 1, it relies on the $L_{0.5}$ index instead of the L_1 index to find the nearest neighbors (NNs) of $\vec{q}_i$. However, the overall query performance of MLSH remains constrained by the inherent

inefficiency of the LSH framework, which is known to incur relatively long query times in practice [19, 23].

1.3 Our Graph-based Solution to ANNS-U- L_p

For ANNS- L_p with a fixed p , graph-based methods such as Hierarchical Navigable Small World (HNSW) [23] have been shown to deliver substantially better query efficiency than LSH-based approaches. As a result, graph-based ANNS methods have become one of the most widely adopted approaches in modern industrial vector database systems [2, 4]. We are hence motivated to explore whether a graph-based ANNS- L_p solution can be extended to handle the ANNS-U- L_p problem. However, a naive approach—constructing a separate graph index for every possible p value—clearly fails to scale. Moreover, building even a single high-quality graph index is computationally expensive and time-consuming, as noted in [19]. These challenges likely explain why, despite the appeal of graph-based methods, no prior work has successfully extended them to support efficient ANNS under universal L_p metrics—until this work.

Another challenge posed by ANNS-U- L_p is that, even for the much simpler ANNS- L_p problem (with a fixed p), directly applying HNSW is much less efficient (except for a few special p values) than its well-known performance for ANNS- L_2 . This is because the dominant cost in HNSW query processing arises from query-to-data (Q2D) distance computations, and the algorithm needs to evaluate Q2D distances for a large number of intermediate points. When the target metric is a general L_p , each such Q2D computation is substantially more expensive than under L_2 (as analyzed in Section 2.1), which can significantly increase the query time.

To address the above challenges, we propose Universal HNSW (U-HNSW)—the first graph-based method that extends the widely adopted HNSW algorithm for solving the ANNS-U- L_p problem. Unlike a naive approach that simply replaces the LSH indexes in MLSH with HNSW graphs, U-HNSW is enabled by two key observations: 1) L_1 or L_2 distance is over an order of magnitude faster to compute than general L_p via SIMD as described in Section 2.1; 2) L_1 or L_2 metric ordering is a highly effective filter for L_p nearest neighbors, so a small candidate set of only $t = 300$ points already captures the true top- K L_p NNs with recall close to 1 (for $K = 50$), as we detail in Section 3.1. Specifically, U-HNSW maintains only two graph indexes: G_1 , an HNSW index built under the L_1 metric, and G_2 , an HNSW index built under the L_2 metric. During query processing, given a query $(\vec{q}, p)$, U-HNSW first selects the base index—either G_1 or G_2 —whose underlying metric is closer to the query metric L_p (with a slight weighting adjustment). It then searches for ANNs of $\vec{q}$, in the selected base metric (L_1 or L_2), using the selected graph (G_1 or G_2). This step is computationally efficient because Q2D distance computations in the base metric (L_1 or L_2) is relatively inexpensive (observation 1). Next, each candidate point returned from the base-metric search—ordered by its base distance (e.g., L_1)—is vetted by computing its exact Q2D distance under the query metric L_p . This step is also computationally efficient thanks to observation 2. To further improve query efficiency, U-HNSW performs these relatively expensive L_p distance computations progressively and terminates once the estimated recall is found to exceed the target threshold as described in Section 3.1. As demonstrated in Section 4, U-HNSW not only achieves a substantial performance improvement over MLSH, but also significantly outperforms the

original HNSW when applied to fixed-metric ANNS- L_p problems, except for a few special p values (e.g., $p = 0.5, 1.5$).

In summary, this work makes three main contributions. First, we present U-HNSW, the first cost-effective practical graph-based solution to the ANNS-U- L_p problem. Second, we develop an adaptive candidate checking framework that reduces the number of expensive L_p distance computations, allowing U-HNSW to outperform its baseline HNSW even for fixed-metric ANNS- L_p tasks. Third, through extensive experiments, we show that U-HNSW not only achieves up to 2670 times shorter query times than the original MLSH implementation running on a RAM disk (up to 15 times shorter than the idealized MLSH), but also outperforms the original HNSW on the ANNS- L_p problem, except for a few special p values.

2 Background and Related Work

In this section, we first define the problem of ANNS under universal L_p metrics (ANNS-U- L_p) and explain how the computational cost of L_p distance varies significantly with p , in Section 2.1. Finally, we describe HNSW, which serves as the baseline for our U-HNSW scheme, in Section 2.2.

2.1 ANNS under Universal L_p Metrics

Given a query tuple $(\vec{q}, p)$, the goal of ANNS-U- L_p is to retrieve K points with a high recall value (say 90%) for the top- K nearest neighbors of the query point $\vec{q}$ in $\mathcal{D}$ according to the L_p metric, namely, with at least a specified fraction of the returned points belonging to the true top- K nearest neighbors, in as little query time as possible. In the sequel, unless specifically mentioned, we assume that the given p value ranges from 0.5 to 2, which covers the commonly evaluated range of p values used in previous ANNS-U- L_p works [21, 29]. We follow the convention adopted in the ANNS literature [10, 11, 14, 19, 21, 23, 29] of focusing on $p \leq 2$, since L_p metrics with $p > 2$ are rarely used in practical ANNS applications. When the dimension d is not small, the computation time of this L_p distance can vary significantly for different values of p , as follows.

- **$p = 1$ or 2 (fastest).** The computation of the L_1 or L_2 metric involves only basic arithmetic operations (add, subtract, multiply), which can be performed in a computationally efficient manner using SIMD instructions such as `_mm512_sub_ps` and `_mm512_add_ps`. As shown in [25], the L_1 or L_2 metric is the fastest to compute across different dimensionalities.
- **$p = 0.5$ or 1.5 (quite fast).** In addition to basic arithmetic operations, the computation of the $L_{0.5}$ or $L_{1.5}$ metric involves the square root operation, which can also be computed efficiently with SIMD instructions such as `_mm512_sqrt_ps`. As shown in [25], the computation of the $L_{0.5}$ or $L_{1.5}$ metric is slightly slower than that of the L_1 or L_2 metric across different d values.
- **Other p values (slow).** For L_p metrics with other p values, power operations are needed in their computations, which are not SIMD-friendly and are much more expensive computationally than the aforementioned basic and square root operations. Since all such p values require power operations, their computation times are nearly identical to each other. As a result, computation time of a general L_p metric is more than an order of magnitude slower than that of the L_1 or L_2 metric as shown in [25].

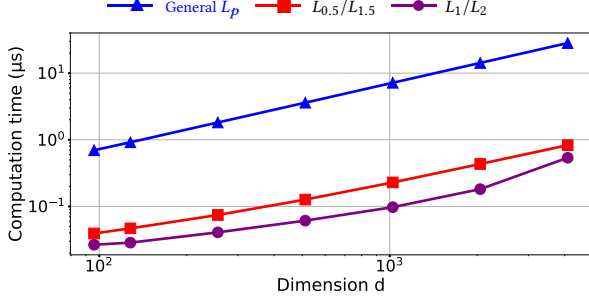


Figure 1: Average computation times (in microseconds) of each L_p distance instance on our workstation described in Section 4.1 (both axes in log scale). L_1 and L_2 metric computations are faster than general L_p metric computation by more than one order of magnitude.

2.2 Hierarchical Navigable Small World Graph

Hierarchical Navigable Small World (HNSW) graph [23] is an efficient and widely recognized graph-based solution to ANNS- L_p for a fixed p value ($0.5 \leq p \leq 2$). HNSW organizes the dataset into a multi-layer proximity graph, where each node represents a data point. The graph is built incrementally by inserting data points one by one: each new point is connected to its M closest neighbors in the graph (where M is a key construction parameter), and the quality of these connections is controlled by efConstruction, the size of the dynamic candidate list maintained during insertion. Larger M and efConstruction yield higher-quality graphs at the cost of longer construction time. If two nodes are close to each other in the L_p metric, there is an edge connecting them in the proximity graph. Given a query point $\vec{q}$, HNSW uses the following greedy search strategy to find the top- K NNs: starting from a pre-defined entry point, HNSW traverses the proximity graph along these edges. When reaching a previously unvisited node $\vec{v}$, HNSW adds $\vec{v}$ to the NN candidate set C , if $\vec{v}$ has a smaller Q2D distance than the farthest point in C . The size of C is controlled by efSearch, the key parameter of HNSW that governs the trade-off between recall and query time: a larger efSearch enlarges C , leading to higher recall but at the cost of more Q2D distance computations and hence longer query time. The top- K NNs are then returned as the K points in C with the smallest Q2D L_p distances to $\vec{q}$.

Now we explain why conventional HNSW must build a separate graph index for every possible p -value (L_p metric) in order to answer ANNS- $U-L_p$ queries. As just explained, the HNSW index is constructed by connecting data points (by proximity edges) based on their closeness (proximity) under a given L_p metric. Let p_1 and p_2 be two distinct p values that are not very close to each other. Then the set of proximity edges in the graph index built for the L_{p_1} metric can differ significantly from that in the graph index built for the L_{p_2} metric. Since the “soul” of a graph index is these proximity edges, using the index built for L_{p_1} to search for NNs under L_{p_2} cannot guarantee high query accuracy. For this reason, HNSW and other graph-based methods [9, 22, 23] all specify a single target distance metric (usually the L_2 metric) and evaluate performance only under that metric.

2.3 Product Quantization ANNS Methods

Besides LSH-based and graph-based methods, product quantization (PQ) based methods [10, 11, 15] are another widely used family of ANNS methods. These methods quantize (cluster) all data points in $\mathcal{D}$ to a small set of designated points (centroids) and the search is performed over only those points in $\mathcal{D}$ that are quantized to the same centroid (and nearby centroids) as the query point is. Most of the PQ-based solutions focus exclusively on the ANNS- L_2 problem. It is not clear how to extend them for solving ANNS- $U-L_p$. For example, RaBitQ [10] does not work for any ANNS- L_p with $p \neq 2$, since its conversion of L_2 to inner product distance does not work for other L_p . A few other PQ methods such as Optimized Product Quantization (OPQ) [11] could in principle be adapted for ANNS- L_p . However, like graph-based methods, the set of centroids in a PQ index built for L_{p_1} can differ significantly from that built for L_{p_2} , when p_1 and p_2 are not close to each other. Hence, to answer ANNS- $U-L_p$ queries, OPQ would require building a separate index for each possible p value—like that described in Section 1.1. For these reasons, we do not compare with PQ-based methods in our evaluation.

3 Scheme Description

In this section, we present our novel solution U-HNSW for the ANNS- $U-L_p$ problem. We first describe the overall procedure of the algorithm, including candidate generation and verification in Section 3.1. Then, we explain the rationale behind using graph indexes on the L_1 and L_2 metrics to generate the candidate set for universal L_p metrics and analyze how to set parameters to minimize the number of expensive L_p distance computations in Section 3.2. In this section, we focus on the general case where $p \notin \{1, 2\}$; when $p = 1$ (or $p = 2$), the query reduces to standard ANNS under the L_1 (or L_2) metric, and we can directly search the corresponding graph index G_1 (or G_2) for K -NNs.

3.1 Algorithm Description

As mentioned earlier, U-HNSW extends HNSW for ANNS- $U-L_p$. To this end, U-HNSW builds and uses two HNSW indexes, namely G_1 for the L_1 metric and G_2 for the L_2 metric. Algorithm 1 shows how U-HNSW processes an ANNS- $U-L_p$ query $(\vec{q}, p)$, to arrive at the K NNs of $\vec{q}$ under the L_p metric. It consists of the following two steps.

Candidate generation. U-HNSW first selects a graph index according to the p value specified in the query as follows. If $p \leq 1.4$, the graph index G_1 is selected; otherwise, the graph index G_2 is used. We use 1.4 as the default cutoff value, which approximately equals the crossover point as shown in Figure 2. The performance is not sensitive to small perturbations around this value. Then, U-HNSW runs the standard KnnSearch procedure in HNSW to return a candidate set C containing the t -NNs ($t > K$), sorted in the ascending order of their Q2D distances (as specified in the HNSW code manual [1]), under the corresponding base metric (Line 3 and Line 5). When t is reasonably large, C empirically contains most of the true top- K NNs of $\vec{q}$ in L_p distance (so a high recall can potentially be achieved); we will explain how to set the parameter t in Section 3.2. This extends the aforementioned observation of LazyLSH (described in Section 1.2) to a wider range of p values. This extension is justified by the well-known norm equivalence in finite dimensions [18].

Candidate verification. The candidate set C contains t ($t > K$) points sorted by L_1 or L_2 distance, not by the target L_p distance. Although C contains most of the true top- K NNs (in L_p distance),

Input: Query tuple $(\vec{q}, p)$; integer K
Output: Top- K NNs of $\vec{q}$ under L_p
1 **Parameters:** t (candidate set size, default 300 for $K = 50$);
 τ (early-termination threshold, default 0.92);
 κ (batch size, default K)
2 **if** $p \leq 1.4$ **then**
3 | $C \leftarrow G_1.\text{KnnSearch}(\vec{q}, t)$
4 **else**
5 | $C \leftarrow G_2.\text{KnnSearch}(\vec{q}, t)$
6 $R \leftarrow$ first K points of C
7 **foreach** batch of κ points R' of C **do**
8 | $R_{\text{new}} \leftarrow$ top- K points by L_p Q2D distance in $R \cup R'$
9 | **if** $|R_{\text{new}} \cap R|/K \geq \tau$ **then return** R_{new} ;
10 | $R \leftarrow R_{\text{new}}$
Algorithm 1: U-HNSW Query for ANNS- $U-L_p$

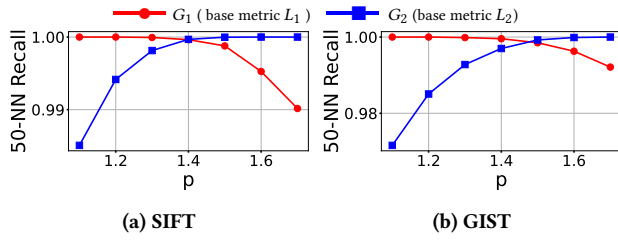


Figure 2: Recall under the target L_p metric when the candidate set is the true top- t NNs on GIST and SIFT datasets in Table 1. We set $t = 300$ as described in Section 3.2. The crossover point appears around 1.4, motivating our default cutoff 1.4.

simply returning the first K points of C (ranked by their base-metric Q2D distances) may not result in a high recall. Therefore, we re-rank (verify) points in C by their L_p Q2D distances, as follows. First, we pop the first K points from the candidate set C , and initialize them as (a set) R . Then, we iteratively pop a batch of κ ($\kappa > 1$) points from C (denoted as R'), and find the K points (denoted as R_{new}) with the smallest L_p Q2D distances in the union $R \cup R'$. The iteration stops once the intersection ratio $|R \cap R_{\text{new}}|/K$ is above a pre-defined threshold τ , indicating that the result will not significantly improve with further iterations. This early-termination strategy ensures that as few expensive L_p distance computations as needed (to achieve the desired recall) are performed.

The early-termination parameter τ can be viewed as an estimator of the recall for the query $\vec{q}$. To make this estimator accurate, the batch size κ must be sufficiently large so that the intersection cardinality $|R_{\text{new}} \cap R|$ reflects how many candidates in R_{new} are likely to be among the true top- K NNs of $\vec{q}$. If κ is too small, U-HNSW may terminate prematurely before examining all potential true top- K NNs. We find that setting κ to K empirically achieves a good trade-off between query accuracy and query speed.

3.2 Parameter Tuning

In this subsection, we describe how to tune the parameters in U-HNSW to achieve a short query time while maintaining a good query accuracy (measured by the recall). As mentioned earlier, in U-HNSW, the total query time T_{query} consists of two parts: the time for candidate generation ($N_b T_b$) and the time for candidate

verification ($N_p T_p$). That is,

$$T_{\text{query}} = N_b T_b + N_p T_p, \quad (1)$$

where N_b and N_p denote the number of Q2D distance computations under the base metric and under the L_p metric, respectively; and T_b and T_p are the average time per Q2D distance computation under the base metric and under the L_p metric, respectively.

In U-HNSW, we primarily tune two parameters: t and τ . As we will elaborate next, t determines the number of base-metric distance computations N_b in Equation 1, and τ determines the number of L_p distance computations N_p in Equation 1. We now explain how to set these parameters t and τ .

As described in Section 3.1, given K , the number of NNs desired by the query, we make t sufficiently large to achieve a target recall value of $r = 0.9$ (used throughout Section 4). For $K = 50$ (used throughout Section 4), we have found, through the following idealized recall experiment, that $t = 300$ is large enough to meet this target recall for all datasets. Figure 3(a) reports the *idealized* recall obtained when using the *true* top- t NNs under the base metric (L_1) as the candidate set. We evaluate the most demanding setting ($p = 0.5$) with $K = 50$ on both the SIFT and GIST datasets in Table 1. These two datasets are chosen as representatives because they span a wide range of dimensionalities (128 and 960, respectively), and the parameter tuning results on other datasets are similar. As t increases, the idealized recall quickly approaches saturation, so much so that $t = 300$ already achieves an idealized recall close to 1. In this case, the *true potential* recall (if the Q2D distances of all t points in C were “painstakingly” computed) exceeds 0.94, which leaves a “wiggle room” of 0.04 for achieving the actual target recall of 0.9.

Using $t = 300$, which may be larger than necessary on some datasets, only incurs a small additional overhead in the total query time, for the following reason. N_b grows sub-linearly with t when t is within a few hundreds [23], and T_b is more than an order of magnitude smaller than T_p (as explained in Section 2.1). As a result, the term $N_b T_b$ remains small in Equation 1 even if t is larger than necessary.

The threshold parameter τ controls the early-stopping criterion in U-HNSW’s candidate verification step. As mentioned in Section 3.1, τ can be viewed as an estimate of the query recall value. Given a target recall r (e.g., $r = 0.9$ in Section 4), we set $\tau = 0.92$ to leave a small safety margin (a part of the “wiggle room” above) in the most demanding case, to allow for the recall loss caused by the early-termination strategy. As shown in Figure 3(b), setting $\tau = 0.92$ consistently meets the target recall 0.9 on both datasets. If a higher recall value is required, we can increase τ accordingly, which allows U-HNSW to meet the new target recall while keeping N_p small, so that the term $N_p T_p$ remains small in Equation 1. Similar to the analysis of t , we provide empirical evidence from two real-world datasets GIST and SIFT in Figure 3(b). Figure 3(b) clearly shows that setting $\tau = 0.92$ (target 0.9 plus a small margin) consistently meets the target recall (0.9) on both datasets while avoiding unnecessary L_p computations.

4 Evaluation

In this section, we conduct an extensive study using six widely used publicly available datasets of diverse dimensions, sizes (number of points) and types. We first evaluate the query performance of U-HNSW and compare it with MLSH’s on the ANNS- $U-L_p$ problem in Section 4.2. Then we evaluate the query performance of U-HNSW on ANNS- L_p (with fixed p) and compare it with that of HNSW. Our results show conclusively that U-HNSW not only

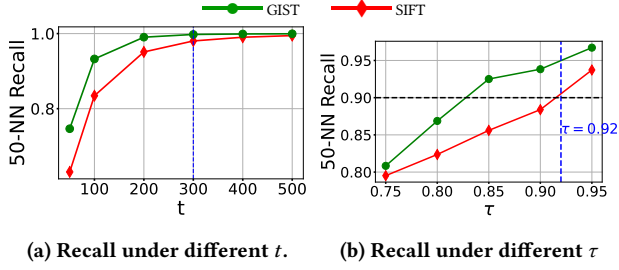


Figure 3: Empirical evidence for tuning t and τ on GIST and SIFT datasets in Table 1. We use $p = 0.5$, which represents the most demanding setting due to its large mismatch to the base metric (L_1).

outperforms the state-of-the-art ANNS- $U-L_p$ solution MLSH by a factor of up to 15.0 in terms of query efficiency, but also is much faster than its baseline HNSW on ANNS- L_p , except for a few special p values.

4.1 Evaluation Settings

4.1.1 Evaluation Datasets. We use six widely used publicly available datasets of diverse dimensions, sizes (number of points), and types. The SIFT dataset contains 2 million points sampled uniformly at random from the 1 billion points contained in SIFT1B [6]. We cannot use SIFT1B instead because constructing the HNSW graph index for it would not fit into the 128 GB main memory of our workstation. For each of the six datasets, Table 1 shows its size n and its dimension d . For each of the six datasets, the query set contains 1,000 points sampled uniformly at random from the set of query points associated with the corresponding dataset.

Table 1: Evaluation datasets summary.

Dataset	Sun [27]	Trevi [26]	GIST [6]	Deep [19]	GloVe [24]	SIFT [6]
n	78K	99K	1M	1M	1.2M	2M
d	512	4,096	960	256	100	128

4.1.2 Performance Metrics. We evaluate the performance of U-HNSW, HNSW and MLSH in three aspects: index scalability, query efficiency, and query accuracy. To measure index scalability, we use *index size* (excluding the size of the original dataset $\mathcal{D}$). To measure query efficiency, we use *query time*. To measure query accuracy, we use *recall*, which is defined as follows. Denote by $\mathcal{S}^*$ the set of true top- K NNs under the L_p metric and by $\mathcal{S}$ the set of K approximate NNs in the result. The top- K query recall is equal to $|\mathcal{S}^* \cap \mathcal{S}|/K$. Throughout our evaluation, unless otherwise stated, we use $K = 50$; using any other value of K ranging from 10 to 100 (commonly used in the ANNS literature [14, 21–23]) results in similar query performance. Each query time or recall value presented in this section is the average over all queries.

4.1.3 Implementation Details. We implement U-HNSW in C++ based on the widely used open-source HNSW implementation called HNSWlib [1]. For MLSH, we use the C++ source code provided by its authors. We optimize the L_1 , L_2 , $L_{0.5}$ and $L_{1.5}$ distance computations using the SIMD instructions described in Section 2.1 and general L_p distance computations using the optimized implementation in NMSLIB [3] (a popular ANNS library used in Amazon Elasticsearch service). We compile all C++

source code using g++ 13.3 with -O3. All experiments are done on a workstation running Ubuntu 22.04 with an Intel Core i7–9800X CPU @ 3.80 GHz and 128 GB RAM.

4.1.4 Benchmark Algorithms. In our evaluation, we compare U-HNSW against the original HNSW and MLSH. We do not compare against LazyLSH since MLSH outperforms LazyLSH as shown in [21]. The settings of benchmark algorithms are described as follows:

U-HNSW. For U-HNSW, we first describe the parameter settings of the G_1 and G_2 graph indexes. For these two graph indexes, we set their parameters so that they can achieve a very high recall when searching for nearest neighbors under the base metric. Specifically, we use $M = 32$ and $efConstruction = 500$ for both G_1 and G_2 . As explained in Section 2.2, $efSearch$ is the key HNSW parameter that trades off recall against query time, and we tune it per dataset (ranging from 400 to 2,000). For U-HNSW, we deliberately use a relatively large $efSearch$ so that the candidate generation step already attains the very high potential recall as described in the third paragraph of Section 3.2. Then we set the parameter $\tau = 0.92$ which is slightly higher than the target recall value 0.9 as described in Section 3.2. Such a setting ensures that most queries achieve a recall value larger than 0.9. As for the parameter t , we set it to 300 according to the parameter tuning strategy described in Section 3.2.

HNSW. We compare U-HNSW with the original HNSW on the ANNS- L_p problem with a fixed p value. For $p = 1$ and $p = 2$, the “original HNSW” baseline directly reuses G_1 and G_2 , respectively. For other p values, we build a separate HNSW graph index under the corresponding L_p metric using $M = 32$ and $efConstruction = 200$. The parameter $efSearch$ is tuned per dataset (ranging from 200 to 1,600) so that the original HNSW achieves an average recall comparable to that of U-HNSW (above 0.9) across all p values, ensuring a fair comparison of query times between HNSW and U-HNSW at similar recall levels.

MLSH. We use the parameter settings and the original implementation of MLSH provided by its authors. For our in-memory evaluation (with the MLSH index stored on a RAM disk), the measured query time of MLSH is extremely high: For example, on the Sun and GloVe datasets, the original MLSH implementation takes approximately 4.67 seconds and 5.35 seconds per query, respectively, making U-HNSW 2670 and 890 times faster on these two datasets, respectively. However, one may argue that directly comparing against the original MLSH implementation could be unfair for the following two reasons. First, the L_p distance computations in MLSH are not optimized using the SIMD instructions described in Section 2.1 and the L_p metric optimization in NMSLIB described in Section 4.1.3, which makes each Q2D L_p computation in MLSH substantially slower than that in U-HNSW. Second, MLSH is built upon QALSH (Query Aware LSH [14]), which is designed for external-memory operations and thus aims to minimize the number of sequential I/Os. As an inherent trade-off, its CPU computation time, other than that for distance computations, can be extremely large, as observed in [12].

To make the comparison implementation-agnostic and extremely favorable to MLSH, we report, for MLSH, the *idealized* query time that accounts only for query-to-data (Q2D) L_p distance computations, calculated as $N_p T_p$, where N_p is the number of Q2D L_p distance computations performed by MLSH query processing (as reported by the MLSH program), and T_p is the same average per Q2D distance computation time as achieved

in U-HNSW (using SIMD-optimized L_p implementation). This idealized estimate accounts only for the Q2D L_p distance computations in MLSH and excludes all other expensive CPU computations. As we will elaborate next, even using such an extremely favorable idealized estimate for MLSH, U-HNSW still achieves significantly shorter query times while maintaining comparable or better recall.

4.2 Comparison against MLSH

In this subsection, we compare U-HNSW against MLSH, the state-of-the-art solution to the ANNS- L_p problem. Since MLSH can solve this problem only for $p \leq 1$, we restrict p to values between 0.5 and 1. Accordingly, we report only the size of G_1 (the graph index built on the L_1 metric) as the index size of U-HNSW, because the graph index G_2 is not used (since $p \leq 1$) in this experiment. Following [21], the p value in the query tuple $(\vec{q}, p)$ is uniformly randomly selected from the following set: $\{0.5, 0.6, 0.7, 0.8, 0.9\}$.

Table 2 clearly shows that U-HNSW significantly outperforms the idealized MLSH, achieving up to 15.0 times shorter query times while providing higher recall and requiring smaller index sizes across all six datasets. Specifically, U-HNSW achieves between 4.4 and 15.0 times shorter query times than the idealized MLSH on all datasets.

Table 2: Evaluation results of query times (in milliseconds), recall and index sizes (in megabytes). Numbers in boldface are the best in each group.

Dataset	Recall		Query Time (ms)		Index Size (MB)	
	U-HNSW	MLSH	U-HNSW	MLSH	U-HNSW	MLSH
Sun	0.969	0.942	1.75	13.75	21.3	119.5
Trevi	0.976	0.951	8.72	131.22	26.0	151.2
GIST	0.939	0.917	9.58	66.83	260.2	1525.9
Deep	0.951	0.943	6.34	29.39	260.2	1525.9
GloVe	0.926	0.921	6.01	26.48	309.4	1818.4
SIFT	0.953	0.942	1.71	9.12	985.0	3051.8

4.3 Comparison with the Original HNSW

In this subsection, we compare U-HNSW's query performance with the original HNSW on ANNS- L_p with a fixed p value. The goal of this subsection is to demonstrate that our U-HNSW can achieve better query performance than the original HNSW even on the ANNS- L_p problem, except for a few special p values. As for the index size, U-HNSW's index size is twice that of the original HNSW because U-HNSW uses two HNSW graph indexes (G_1 and G_2). The evaluated p values are 0.5, 0.6, 0.7, $\dots$, 2.0. We report the average query performance of all query points for each p value.

Figure 4 shows the query times of U-HNSW versus those of the original HNSW for different p values when both schemes achieve similar recall values higher than 0.9. As shown in Figure 4, query times of U-HNSW are 3.3 to 8.5 times shorter than those of the original HNSW on all datasets across a wide range of p values, except for $p = 0.5$ and 1.5. At these two values, the query times of HNSW are 1.1 to 2.1 times shorter than those of U-HNSW on all datasets except GloVe, where their query performances are comparable. This is because for $p = 0.5$ and 1.5, the $L_{0.5}$ and $L_{1.5}$ distances can be efficiently computed using SIMD instructions as described in Section 2.1, making their computation cost close to

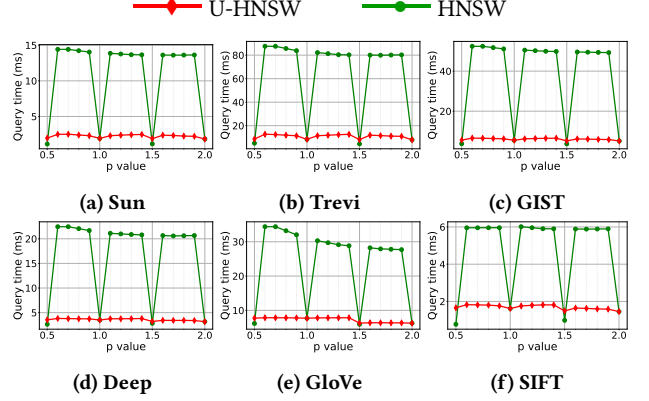


Figure 4: The query times (in milliseconds) of U-HNSW versus HNSW with different p values across all datasets. Both methods achieve similar recall value and thus omitted. U-HNSW is faster than HNSW on ANNS- L_p with a fixed p value, except for $p = 0.5$ or 1.5.

that of the base metric (L_1 or L_2). In such a case, on the one hand, the main advantage of U-HNSW, which is to significantly reduce the number of (presumably expensive) L_p distance computations, becomes blunted. On the other hand, U-HNSW still has to perform additional base-metric distance computations during the candidate generation step as described in Section 3.2. As a result, the overall query time of U-HNSW becomes longer than that of HNSW under these special p values. For general L_p metrics that do not benefit from the SIMD instructions, however, U-HNSW consistently delivers faster query performance than the original HNSW for ANNS- L_p queries.

5 Conclusion

In this paper, we propose U-HNSW, the first graph-based solution for approximate nearest neighbor search under universal L_p metrics. U-HNSW leverages two base graph indexes built on two base metrics (L_1 and L_2) to generate high-recall candidates and then verifies these candidates with an early-termination strategy that substantially reduces the number of expensive L_p distance computations. Experiments on multiple real-world datasets show that U-HNSW not only achieves up to 2670 times shorter query times than the original MLSH implementation running on a RAM disk (up to 15 times shorter than the idealized MLSH), but also outperforms the original HNSW on the ANNS- L_p problem (with a fixed p value), except for a few special p values.

Acknowledgments

This work was supported in part by the National Science Foundation under Grant No. CNS-2007006.

Artifacts

We have uploaded the code of U-HNSW and benchmark datasets in the public GitHub repository [5]. Please refer to the README file for instructions on how to use our code.

References

- [1] [n. d.]. HNSWlib—fast approximate nearest neighbor search. <https://github.com/nmslib/hnswlib>.
- [2] [n. d.]. Milvus: The High-Performance Vector Database Built for Scale. <https://milvus.io/>.

- [3] [n. d.]. Non-Metric Space Library (NMSLIB). <https://github.com/nmslib/nmslib>.
- [4] [n. d.]. Pinecone: The vector database for scale in production. <https://www.pinecone.io/>.
- [5] [n. d.]. U-HNSW: An Efficient Graph-based Solution to ANNS Under Universal L_p Metrics. <https://github.com/ANNS-Research/U-HNSW>.
- [6] Laurent Amsaleg and Hervé Jégou. 2010. Datasets for ANN neighbor search. <http://corpus-texmex.irisa.fr/>.
- [7] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry* (Brooklyn, New York, USA). Association for Computing Machinery, New York, NY, USA, 253–262.
- [8] Wenqi Fan, Yujian Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A Survey on RAG Meeting LLMs: Towards Retrieval-Augmented Large Language Models. In *Proceedings of the 30th ACM SIGKDD Conference* (Barcelona, Spain). Association for Computing Machinery, New York, NY, USA, 6491–6501.
- [9] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proc. VLDB Endow.* 12, 5 (Jan. 2019), 461–474.
- [10] Jianyang Gao and Cheng Long. 2024. RaBitQ: Quantizing High-Dimensional Vectors with a Theoretical Error Bound for Approximate Nearest Neighbor Search. *Proc. ACM Manag. Data* 2, 3, Article 167 (May 2024), 27 pages. doi:10.1145/3654970
- [11] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2014. Optimized Product Quantization. *IEEE Trans. Pattern Anal. Mach. Intell.* 36, 4 (April 2014), 744–755. doi:10.1109/TPAMI.2013.240
- [12] Long Gong, Huayi Wang, Mitsunori Ogihara, and Jun Xu. 2020. iDEC: indexable distance estimating codes for approximate nearest neighbor search. *Proc. VLDB Endow.* 13, 9 (May 2020), 1483–1497.
- [13] Peter Howarth and Stefan Rüger. 2005. Fractional distance measures for content-based image retrieval. In *European Conference on Information Retrieval*. Springer, 447–456.
- [14] Qiang Huang, Jianlin Feng, Qiong Fang, Wilfred Ng, and Wei Wang. 2017. Query-aware locality-sensitive hashing scheme for L_p norm. *The VLDB Journal* 26, 5 (Oct. 2017), 683–708.
- [15] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Trans. Pattern Anal. Mach. Intell.* 33, 1 (Jan. 2011), 117–128. doi:10.1109/TPAMI.2010.57
- [16] Benjamin Klein and Lior Wolf. 2019. End-to-end supervised product quantization for image search and retrieval. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5041–5050.
- [17] Xiang-Zhen Kong, Yu Song, Jin-Xing Liu, Chun-Hou Zheng, Sha-Sha Yuan, Juan Wang, and Ling-Yun Dai. 2021. Joint L_p -norm and L_2 , 1-norm constrained graph laplacian PCA for robust tumor sample clustering and gene network module discovery. *Frontiers in Genetics* 12 (2021), 621317.
- [18] Erwin Kreyszig. 1991. *Introductory functional analysis with applications*. John Wiley & Sons.
- [19] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2019. Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering* 32, 8 (2019), 1475–1488.
- [20] Kevin Lin, Huei-Fang Yang, Jen-Hao Hsiao, and Chu-Song Chen. 2015. Deep learning of binary hash codes for fast image retrieval. In *Conf. on Comput. Vis. and Pattern Recognit. Workshops (CVPRW)*. IEEE, Boston, USA, 27–35.
- [21] Kejing Lu and Mineichi Kudo. 2021. MLSH: Mixed Hash Function Family for Approximate Nearest Neighbor Search in Multiple Fractional Metrics. In *Database Systems for Advanced Applications: 26th International Conference, DASFAA 2021, Taipei, Taiwan, April 11–14, 2021, Proceedings, Part II* (Taipei, Taiwan). Springer-Verlag, Berlin, Heidelberg, 569–584.
- [22] Kejing Lu, Mineichi Kudo, Chuan Xiao, and Yoshiharu Ishikawa. 2021. HVS: hierarchical graph structure based on voronoi diagrams for solving approximate nearest neighbor search. *Proc. VLDB Endow.* 15, 2 (Oct. 2021), 246–258.
- [23] Yu A. Malkov and D. A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42, 4 (2020), 824–836.
- [24] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. <https://nlp.stanford.edu/projects/glove/>.
- [25] Huayi Wang, Jingfan Meng, and Jun Xu. 2026. U-HNSW: An Efficient Graph-based Solution to ANNS Under Universal L_p Metrics. arXiv:2605.02030 [cs.DB] <https://arxiv.org/abs/2605.02030>
- [26] Simon Winder, Matt Brown, Noah Snavely, Steven Seitz, and Richard Szeliski. 2007. Trevis: Local Image Descriptors Data. <http://phototour.cs.washington.edu/patches/default.htm>.
- [27] Jianxiong Xiao, Krista A. Ehinger, James Hays, Antonio Torralba, and Aude Oliva. 2016. SUN Database: Exploring a Large Collection of Scene Categories. *Int. J. Comput. Vision* 119, 1 (Aug. 2016), 3–22.
- [28] Frank F. Xu, Uri Alon, and Graham Neubig. 2023. Why do nearest neighbor language models work?. In *Proceedings of the 40th International Conference on Machine Learning* (Honolulu, Hawaii, USA) (ICML '23). JMLR.org, Article 1596, 17 pages.
- [29] Yuxin Zheng, Qi Guo, Anthony K.H. Tung, and Sai Wu. 2016. LazyLSH: Approximate Nearest Neighbor Search for Multiple Distance Functions with a Single Index. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA). Association for Computing Machinery, New York, NY, USA, 2023–2037.