Evaluating and Generating Query Workloads for High Dimensional Vector Similarity Search

Matteo Ceccarello
matteo.ceccarello@unipd.it
University of Padova, Italy

Alexandra Levchenko
alexandra.levchenko@isep.fr
Isep, LISITE, France

Ioana Ileana
ioana.ileana@parisdescartes.fr
Université Paris Cité, France

Themis Palpanas
themis@mi.parisdescartes.fr
Université Paris Cité, France

Abstract

Similarity search lies at the heart of many modern applications, ranging from databases to deep learning to data series analysis. As such, a vast effort has been invested in developing algorithms, data structures and implementations to speed up this crucial subroutine. To empirically validate these approaches, several benchmarking efforts have been initiated covering a wide array of datasets. In this paper, we observe that usually little control is exercised on the hardness of the workloads with which methods are tested and compared. To address this issue, we first evaluate several query hardness measures with respect to their ability to capture the empirical hardness of a query, i.e. the effort invested by an index data structure to provide an answer. Then, we propose two methods, deemed Hephaestus-Annealing and Hephaestus-Gradient, for synthesizing query workloads so that they meet a user-specified hardness target. Both methods allow to produce workloads with the desired hardness: we find that Hephaestus-Gradient is faster, while Hephaestus-Annealing makes fewer assumptions on the target hardness measure. The resulting workloads can be used to gain insights into the behavior of similarity search algorithms.

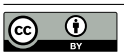
CCS Concepts

• **Information systems** → **Database performance evaluation**;
• **Theory of computation** → **Nearest neighbor algorithms**.

Keywords

Similarity search, Benchmarking, Query hardness, Query generation

ACM Reference Format:
Matteo Ceccarello, Alexandra Levchenko, Ioana Ileana, and Themis Palpanas. 2025. Evaluating and Generating Query Workloads for High Dimensional Vector Similarity Search. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD '25), August 3–7, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3711896.3737383



KDD '25, August 3–7, 2025, Toronto, ON, Canada

ACM ISBN 979-8-4007-1454-2/2025/08
https://doi.org/10.1145/3711896.3737383

1 Introduction

High-dimensionality vector similarity search is a fundamental task in a wide range of applications, from information retrieval and recommendation systems to computational biology and computer vision [11, 18, 37]. The goal of similarity search is to identify items in a dataset that are most similar to a given query, based on some defined similarity measure. Given the prominence of this task, a rich ecosystem of algorithms, index data structures, and implementations has flourished in recent years [19, 20, 36, 55].

Alongside the development of index data structures and algorithms, the necessity of establishing a common testbed to evaluate different implementations has spurred a variety of benchmarking efforts [6, 19, 20, 32, 48]. These benchmarks are now in widespread use and provide the community with a very useful resource to assess the merits of different algorithms and implementations in a variety of scenarios, encompassing different datasets and workloads.

The goal of a benchmark is to find, for a given workload, what is the performance of different implementations under different parameter settings. Performance can be measured in terms of time to run queries, or of number of executed distance computations, or again in terms of the quality of the answer itself, for approximate index data structures. This allows to study the behavior of algorithms under different circumstances, as well as the tradeoffs involved in configuring the implementations, while allowing the community to find the fastest implementation on a given workload.

In this paper we focus on studying the *workloads*, i.e. the set of queries, that are used to evaluate competing algorithms. In fact, the common practice to prepare a workload follows the tradition set by machine learning: a dataset is partitioned randomly in two parts, a larger one to be indexed and smaller one to be used as queries. This approach is sensible in that it gives a set of readily available queries that come from the same distribution of the data. However, how hard are these queries, both intrinsically and for a given index?

As we shall see with our experiments, these workloads do not exercise the full spectrum of possible behaviors of the index data structures, and real-world workloads have been found to be considerably harder [4, 9, 13, 28]. Furthermore, these workloads do not provide focused information about queries with a specific hardness. Finally, the same query point can be hard or easy depending on the value of k. These considerations are relevant in order to stress-test index structures, better understand their behavior, and to design and develop better-performing versions.

The culprit is that the workload selection procedure used so far does not allow to exercise any control on the hardness of the

queries. We thus propose a way to *generate* workloads with a prespecified level of hardness. The first step is measuring the hardness of a workload: *empirical* hardness measures evaluate the work performed by actual index structures, *explicative* measures on the other hand try to capture geometric properties relating queries and data. While *empirical* measures are what a user is ultimately interested in, *explicative* measures allow to further interpret the results: why are some queries doing more work than others? What are the characteristics that make them easy or hard to answer?

Using these measures, we propose methods to generate query workloads both in a index-agnostic and a index-aware way. In the former case, queries should present intrinsic characteristics that make them easy or hard: different index data structures might then be able to deal with them with different degrees of efficiency. In the latter case, queries should be hard (or easy) for a *specific* index data structure: this allows to identify bottlenecks and weak spots during index development and tuning.

Our contributions are articulated as follows: **(a)** We provide an overview of different measures to assess the hardness of queries, both in an index-independent and index-centered way (Section 3); **(b)** We propose methods to generate synthetic queries of given target hardness (Section 4), providing an easy to use Python implementation[1]; **(c)** We experimentally evaluate the relation between hardness measures and actual query difficulty, and test the effectiveness of the methods we propose in generating query workloads (Section 5).

2 Related Work

High-dimensional vector nearest neighbor search is a crucial subroutine in many different contexts, with a wide array of different approaches being proposed. For a recent account of the latest developments see [9, 36, 55] and references therein.

Among the many approaches devised for *approximate* nearest neighbor search, we focus on the ones implemented in the Faiss library [17], including Hierarchical Navigable Small World (HNSW) graphs [34] and an Inverted File [27, 49]. The Faiss library has been adopted as a baseline by some recent benchmarking efforts [48].

Another line of work [38, 39, 41, 54] focused on building data structures providing *exact* answers to queries. Here we focus on MESSI [40], which builds a tree-based index on iSAX words [47], and on DSTree [54].

A popular benchmarking effort is ann-benchmarks [6], which provides a collection of datasets and query workloads. Since its inception, over 40 algorithms have been included in the benchmark. We will use some query workloads of ann-benchmarks as a baseline in our experimental evaluation. A similar benchmarking effort is described in [32], focusing on the Euclidean distance case. Other benchmarking efforts are [19, 20], evaluating both approximate and exact approaches on a variety of datasets, which we also include in this paper. Recently, big-ann-benchmarks [48] scaled the benchmarking effort to billion-scale data, extending the range of tasks to include filtering further the results based on categorical features, out-of-distribution queries, and dynamic data updates. VIBE [28] focuses on datasets derived from embedding models characteristic of modern applications.

[1] https://github.com/cecca/hephaestus/

Dimensionality measures such as the Local Intrinsic Dimensionality [25], the Relative Contrast [24], the query Expansion [3, 8], and the ε-hardness [60] capture features of the distribution of distances from a query that relate to how hard it is to discern nearest neighbors from the rest of the points. The *Steiner*-hardness [56] is specifically designed to investigate the performance of graph-based indices. The relation between these measures and the actual performance of indices was investigated in [7, 8]: in the present paper we build upon and expand their conclusions.

The goal of generating query workloads has also been pursued in Zoumpatianos et al. [60]: given a query, the dataset is modified so that the query achieves the desired hardness. However, modifying the dataset may not be desirable. Moreover, this method incurs significant runtime costs. In the present paper, we take the opposite approach: while keeping the dataset fixed, we carefully place queries in the metric space so to achieve the desired hardness level. Given the fundamentally different scenario that we consider, we will not compare directly with [60].

3 Preliminaries

Let $(\mathcal{X}, d)$ be a metric space, with d being the distance function. In this paper, in particular, we focus on the D-dimensional Euclidean space $\mathbb{R}^D$ under the Euclidean distance, and on S^{D-1}, the unit sphere in D dimensions under the angular distance.

We denote with $S \subset \mathcal{X}$ a dataset, and with $\vec{x} \in \mathcal{X}$ we denote a point in the metric space. Note that in different communities $\vec{x}$ is referred to with different names: it can be a point, a vector, or a data series if the order of coordinates is relevant. Given that in this paper we focus on the Euclidean and angular distances, for which the order of the coordinates is not relevant, we will refer to elements of $\mathcal{X}$ as *points*.

For a dataset S, a query point $\vec{q} \in \mathcal{X}$ and a parameter k, the k-nearest neighbor problem entails finding the k points in S that are closest to $\vec{q}$ according to the distance function d, with ties broken arbitrarily. We denote with $r_1, r_2, \ldots, r_k$ the distance between $\vec{q}$ and its first, second, …, k-th nearest neighbor.

Such queries can be answered exactly in time $O(n)$ by simply evaluating the distance between $\vec{q}$ and all points $\vec{x} \in S$. However, for large datasets it is often desirable to have sublinear query time, often allowing some approximation in the answers.

In the case of approximate approaches, the quality of the answers is usually measured using the *recall*, defined as $\frac{|\{1 \le i \le k : r'_i \le r_k\}|}{k}$. In other words, the recall is the fraction of returned points whose distance from the query is $\le r_k$, the distance of the true k-th nearest neighbor.

3.1 Hardness Measures

We now survey several established *explicative* hardness measures and introduce the *empirical hardness*. We deem *explicative* those measures that are based only on intrinsic properties of a query in relation to a dataset, like the distribution of distances. Such measures, which are independent of any index data structure, can help to *explain* why a given query can be expected to be hard or easy. On the other hand, *empirical* measures reflect the actual performance of a given index data structure on a given query. In

the following sections we will then study the relationship between these measures.

3.1.1 Local intrinsic dimensionality. For a given query point $\vec{q} \in \mathcal{X}$ we consider the distribution of distances to $\vec{q}$ within the metric space, where the distribution arises by sampling n points from the metric space under a certain probability distribution. Let $F : \mathbb{R} \to [0, 1]$ be the cumulative distribution function of distances to $\vec{q}$.

Definition 3.1 ([25]). The local intrinsic dimensionality of F at distance r is

$$ID_F(r) = \lim_{\varepsilon \to 0} \frac{\ln\left(F((1+\varepsilon)r)/F(r)\right)}{\ln\left((1+\varepsilon)r)/r\right)}$$

whenever the limit exists.

Intuitively, the above measure is related to how quickly the probability mass (i.e. the fraction of points that are within the ball of radius r and the ball of radius $(1+\varepsilon)r$ around the query point) increases around the query point.

A large LID value means that it is hard to distinguish points at distance r from the query from the rest of the dataset. Therefore, a hard query is expected to have a large LID value.

The LID can be estimated with a Maximum Likelihood Estimator [5, 31]. Let $r_1 \leq \cdots \leq r_k$ be the the distances of the k-NN of a query $\vec{q}$. Then, the Maximum Likelihood Estimator for $\vec{q}$ at distance r_k is

$$\hat{LID}_k(\vec{q}) = -\left(\frac{1}{k}\sum_{i=1}^{k} \ln \frac{r_i}{r_k}\right)^{-1} \tag{1}$$

3.1.2 Relative Contrast. The Relative Contrast (RC) captures the relationship between the distance of the k-th nearest neighbor of a query and the average distance of points from the query.

Definition 3.2 ([24]). For a query point $\vec{q}$ and a set S, let d_{mean} be the average distance of $\vec{q}$ to points in S. The Relative Contrast of $\vec{q}$ at k with respect to S is

$$RC_k(\vec{q}) = \frac{d_{mean}}{r_k}$$

where r_k is the distance to the k-th nearest neighbor of $\vec{q}$ in S.

A small relative contrast implies that the distance of the k-th nearest neighbor is close to the average distance to the query: as a consequence, a hard query is expected to have a small RC value.

3.1.3 Query expansion. The concept of Expansion around a query was first introduced to analyze the properties of LSH-based indices [3]. Here we adopt the extended definition introduced in [8].

Definition 3.3. Given a query $\vec{q}$ and an integer k, the Expansion of $\vec{q}$ at k with respect to $k' > k$ is

$$\text{Expansion}_{k'|k}(\vec{q}) = \frac{r_{k'}}{r_k}$$

Similarly to the Relative Contrast, a small Expansion around the query implies that for an index it might be hard to discern between the k-th nearest neighbor and the points that are farther away. Hence, a hard query is expected to have a small Expansion value.

3.1.4 ε-hardness. The ε-hardness [60] focuses on measuring the number of points that sit in the ball of radius $(1+\varepsilon)r_1$, where r_1 is the distance of the 1st nearest neighbor of the query. We extend the definition to support the k-th nearest neighbor, and refer to the metric as $\alpha_{\varepsilon,k}$ following [60].

Definition 3.4. For a query point $\vec{q}$ and a set S, let r_k be the distance of the k-th nearest neighbor of $\vec{q}$ in S. For a parameter $\varepsilon > 0$ the ε-hardness is

$$\alpha_{\varepsilon,k}(\vec{q}) = \frac{|\{x \in S : d(\vec{q}, x) \leq (1+\varepsilon)r_k\}|}{|S|}$$

A high ε-hardness value implies that a large fraction of the set S lies at a distance slightly larger than r_k. Therefore, a high value of the ε-hardness is expected for hard queries.

3.1.5 Empirical hardness. The aforementioned measures where introduced in the literature to capture different characteristics of the data that possibly relate to the actual hardness of a query. We now introduce a measure for the work actually invested by an index data structure to answer a given query. Rather than focusing on the running time, which is implementation and platform dependent, we consider the number of full distance computations carried out by the index while answering the query. To make the number comparable across different datasets we normalize it by the total dataset size. Since different index structures are likely to experience different performance on the same query and data, the measures is parameterized by the index structure.

Definition 3.5. Given a data structure $\mathcal{D}$ indexing a point set S, and a recall threshold ρ, the empirical hardness $\mathcal{H}_{\mathcal{D},\rho}(\vec{q})$ of a query $\vec{q}$ is the number of full distance computations carried out by $\mathcal{D}$ in order to achieve recall $\geq \rho$, divided by $|S|$.

For instance, consider the index $\mathcal{D}$ =HNSW, indexing a dataset with one million points. If answering a query $\vec{q}$ with recall at least $\rho = 0.95$ requires computing 100 000 distances, then the empirical hardness of $\vec{q}$ is $\mathcal{H}_{HNSW,0.95}(\vec{q}) = 0.1$.

Note that the definition above considers only *full* distance computations between points in the original space, as this is usually the most expensive subroutine by far, while answering a query. We thus do not count the cost of using of sketches, summaries, or similar estimators that are routinely employed to weed out non-relevant candidate neighbors.

Clearly, hard queries correspond to high empirical hardness.

Remark. *The hardness of a query, for any of the measures discussed in this section, is crucially related to its position in the metric space, relative to the other points in the dataset,* and *to the number of neighbors k we are looking for. Figure 1 reports an example with a dataset of 30 randomly distributed points in $\mathbb{R}^2$. In the figure, the color encodes the Relative Contrast that a query would have in different positions of the plane, for both $k = 1$ and $k = 10$.*

For $k = 1$, notice how the Relative Contrast increases as we get closer to any point: in fact, getting closer to a point makes it easier to distinguish from the others. For $k = 10$, however, the situation is completely different. The locations of the easiest queries are between the points and not close to them, because the hardness is dictated by how similar the 10-th nearest neighbor is to the rest of the points.

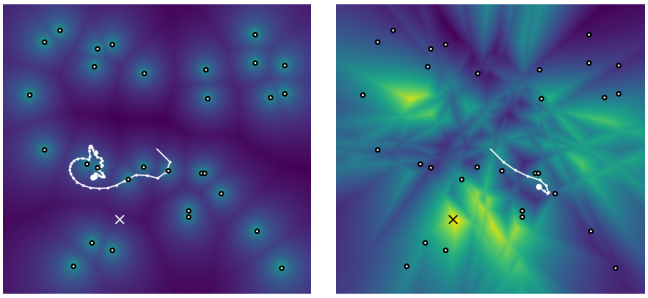


Figure 1: Example dataset of 30 random points in $\mathbb{R}^2$, with colors encoding the relative contrast of each position for $k = 1$ (left) and $k = 10$ (right). High (resp. low) relative contrast is denoted with bright (resp. dark) colors. The white paths mark the trajectories followed by Hephaestus-Gradient to generate a query in each scenario.

Remarkably, if we consider a query located at the position marked with the ×, we have two very different situations depending on the value of k. For $k = 1$ the Relative Contrast of the query is ≈ 1.84, for $k = 10$ its Relative Contrast is ≈ 4.01: in the latter case therefore the same query point is comparatively easier.

4 Generating workloads

We propose two different ways of generating query workloads: Hephaestus-Annealing and Hephaestus-Gradient[2]. Both algorithms generate queries tailored for a specific dataset S with a query hardness measured according to a scoring function ς: for a point $\vec{x}$ and a dataset S, $\varsigma(S, \vec{x})$ is a real number representing how hard or easy the query is. For instance, ς could compute the Relative Contrast of $\vec{x}$. The synthesis of each query then aims at reaching a user-supplied range of desired scores $[y_l, y_u]$. For example, y_l and y_u could be bounds on the desired Relative Contrast of the queries. In general, we will use as scoring functions the hardness measures defined in the previous section.

We also describe a simpler mechanism to build queries based on adding Gaussian noise to queries, to be used as a baseline in our experimental evaluation.

4.1 Hephaestus-Annealing

The first query generation procedure we propose (Algorithm 1) starts from an arbitrary point $\vec{q}$, which can be for instance sampled randomly from the enclosing ball of the dataset. Then, Hephaestus-Annealing iteratively moves $\vec{q}$ so that its score $\varsigma(S, \vec{q})$ falls within the target range $[y_l, y_u]$.

Making no assumption on ς, at every iteration we move $\vec{q}$ in a random direction, by a random amount. If this move leads to a position with a hardness score closer to the target, then we continue to the next iteration. However, the move could of course land into a position whose score is farther from the target range. In this case, we adopt the classic *simulated annealing* strategy: with some probability we accept the move to a worse position, otherwise we backtrack to the previous position of $\vec{q}$. The algorithm terminates

[2]In Greek mythology, Hephaestus is the god of artisans and blacksmiths.

Algorithm 1: Hephaestus-Annealing

Input: Dataset S; starting point $\vec{q}$; hardness scoring function $\varsigma : (\mathcal{X}, S) \to \mathbb{R}$; initial temperature T; maximum number of iterations max_iter; target score range (y_l, y_h).

```
y ← ς(S, q⃗);
for i ← 1 to max_iter do
    q⃗′ ← move q⃗ randomly to a nearby location;
    y′ ← ς(S, q⃗′);
    if y_l ≤ y′ ≤ y_h then return q⃗′ ;
    Δ ← min{|y − y_l|, |y − y_h|};
    Δ′ ← min{|y′ − y_l|, |y′ − y_h|};
    if Δ′ < Δ then (the candidate is closer to the target)
        q⃗ ← q⃗′;
        y ← y′;
    else (we accept a worse move with some probability)
        t ← T/i;
        p ← exp(−|y − y′|/t);
        with probability p do
            q⃗ ← q⃗′;
            y ← y′;
return q⃗;
```

when either $\vec{q}$ has a score within the required bounds or a maximum number of iterations is reached.

In particular, we adopt the *fast annealing* strategy: at iteration i the probability of accepting a bad move is $\exp(-|y - y'|/t)$, where $t = T/i$ is a linearly decreasing *temperature* from an initial value T, and $y = \varsigma(S, \vec{q})$ (resp. $y' = \varsigma(S, \vec{q}')$) is the score of $\vec{x}$ (resp. the moved point $\vec{q}'$).

The intuition is the following: in the first iterations the algorithm boldly explores the space, but as it progresses it becomes more and more conservative, avoiding moves with a worse score.

As for the random distance to move each point in each iteration, we have to consider different distributions depending on the distance metric employed on each particular dataset. For datasets using the Euclidean distance candidate queries are displaced by a distance sampled from an exponential distribution $\text{Exp}(\lambda)$. The rate λ of the exponential distribution is a parameter that we set, by default, to a 100^{th} of the diameter of the dataset. By doing so we favor small moves while still allowing the occasional long-distance jump. For datasets employing the angular distance we apply the same rationale, with the additional constraint that the angular distance cannot exceed 2. Therefore we replace the exponential with the beta distribution with parameters $\alpha = 0.1$ and $\beta = 1$, whose probability density function is skewed towards 0, thus favoring small tweaks to the position of the point $\vec{q}$.

4.2 Hephaestus-Gradient

The approach we discussed above has the advantage of not making any assumption on the scoring function ς. The price we pay for this flexibility is that the algorithm explores the space quite aimlessly.

Algorithm 2: HEPHAESTUS-GRADIENT

Input: Dataset S; starting point $\vec{q}$; hardness scoring function $\varsigma : (\mathcal{X}, S) \to \mathbb{R}$; learning rate η; maximum number of iterations max_iter; target score range $[y_l, y_h]$.

```
for i ← 1 to max_iter do
    y ← ς(S, q⃗);
    if y_l ≤ y ≤ y_h then return q⃗ ;
    else if y < y_l then q⃗ ← q⃗ + η∇ς(S, q⃗) ;
    else q⃗ ← q⃗ − η∇ς(S, q⃗) ;
return q⃗;
```

If the scoring function ς is differentiable (as is the case for the Relative Contrast, for instance) then we can compute its gradient, which gives the direction of steepest change of score. Therefore, for any given candidate query point $\vec{q}$, we can first compute $y = \varsigma(S, \vec{q})$, its hardness score with respect to the set of points S. Then we can check if $y \in [y_l, y_u]$, that is if $\vec{q}$ satisfies the required hardness constraints. If this is not the case, then we compute the gradient $\nabla\varsigma(S, \vec{q})$ and move $\vec{q}$ along the gradient direction, by a step size η. Whether the point is moved upwards or downwards along the gradient depends on whether its score $\varsigma(S, \vec{q})$ is above or below the target score range.

This approach, deemed HEPHAESTUS-GRADIENT, is summarized in Algorithm 2. Note that, as in the Gradient Descent procedure routinely used for training neural networks, the step size η is not required to be constant across all iterations. Furthermore, note that usually in neural network training the Gradient Descent procedure is used to *minimize* a loss function. HEPHAESTUS-GRADIENT instead strives to find a $\vec{q}$ whose score is not necessarily minimal, but rather falls withing a user-given range. As such it is not strictly a gradient *descent* scheme, because the gradient can be followed in either direction.

Figure 1 reports the trajectories followed by the optimization process of HEPHAESTUS-GRADIENT. In the example with $k = 1$ (left pane) the target for the optimization is a query with relative contrast 8 (an easy query), whereas for $k = 10$ (right pane) the algorithm seeks to place a query with relative contrast 1.05 (thus a difficult query).

4.3 Targeting the Empirical hardness

As we shall see in the experimental section, a given fixed value of an explicative hardness measures (e.g. the Relative Contrast) might correspond to a different empirical hardness depending on the dataset and index being used. In some scenarios it might be desirable to generate queries with a given empirical hardness for a specific index. For instance, one might be interested in generating a hard query workload for a specific index to investigate the features of the data that force it to spend a lot of effort to provide the answers.

To tackle this scenario, we propose to modify HEPHAESTUS-GRADIENT to take as a parameter an index $\mathcal{D}$ built on the dataset S, as well as a range of admissible empirical hardness values $[h_l, h_h]$. Then, the stopping condition is modified to check whether the empirical hardness $\mathcal{H}_{\mathcal{D},\rho}(\vec{q})$ is within the requested range. If not, then the algorithm uses the gradient *of a differentiable scoring function* ς to guide the move of $\vec{q}$ to the next position. In fact the empirical hardness $\mathcal{H}_{\mathcal{D},\rho}$ might be, in general, not differentiable.

As a concrete example, this algorithm can be instantiated with MESSI as the index data structure for computing the empirical hardness, and the Relative Contrast to guide the moves of $\vec{q}$ by means of its gradient.

Note that the empirical hardness might be employed directly with HEPHAESTUS-ANNEALING. As we shall see in the experimental section, however, HEPHAESTUS-GRADIENT converges faster.

5 Experimental Evaluation

We aim at answering the following questions:

- How well do *explicative* hardness measures correlate with the observed *empirical* hardness?
- What is the empirical hardness of workloads generated by HEPHAESTUS-ANNEALING and HEPHAESTUS-GRADIENT, compared with the baselines?
- Which algorithm between HEPHAESTUS-ANNEALING and HEPHAESTUS-GRADIENT converges faster?
- How do the algorithms scale with the size of the dataset?
- How effective is HEPHAESTUS-GRADIENT at generating workloads targeting a given empirical hardness range?

Implementation. We implement all the algorithms using Python 3.12, using JAX [12] for automatic differentiation in HEPHAESTUS-GRADIENT and Adam [30] as the optimizer (as implemented in Optax [16]). The learning rate is left as a parameter to be specified in the following. For the sake of reproducibility, our experimental pipeline is publicly available[3]. We also provide an easy to use standalone Python implementation of our methods[4].

We execute our experimental evaluation on a single machine equipped with a 48-core Intel(R) Xeon(R) CPU E5-2650 v4 processor, clocked at 2.20GHz, with 251GB of RAM and a 3.7 TB SCSI SSD disk.

Datasets. We consider the following datasets:

- `astro` (euclidean) [50]: celestial objects represented by 100 million points of dimension 256.
- `deep1b` (euclidean) [44]: 100 million Deep1B vectors of dimension 96, extracted from the final layers of a convolutional neural network.
- `sald` (euclidean) [45]: MRI data, containing 100 million points of dimension 128.
- `seismic` (euclidean) [26]: recordings from seismic instruments at thousands of stations globally, comprising 100 million 256-dimensional points.
- `glove` (angular) [42]: 1 192 512 word embeddings in 100 dimensions derived from 2 billion tweets.
- `nytimes` (angular) [35]: 300 000 New York Times articles embedded in 256 dimensions.
- `text2image` (angular) [1] comprises 10 million, 200-dimensional image embeddings as data, and text embeddings as queries.

We consider a random sample of 5 million points from each dataset (with the exception of smaller datasets), in order to allow

[3] https://github.com/Cecca/workloads-generation/
[4] https://github.com/Cecca/hephaestus/

running a large number of experiments. Nevertheless, we stress that our method is able to scale to much larger datasets: Section 5.4 reports scalability experiments.

Baselines. We consider two baselines for comparing our workload generators.

Each dataset is complemented by a query workload, which we refer to as Baseline, that is routinely used to benchmark index data structures. This query workload, comprising 10 000 queries for `nytimes` and `glove` and 1 000 points for the others, is built by extracting points from the dataset itself.

Furthermore, we consider a simple method: sample points from the dataset, and perturb their coordinates using Gaussian noise. In the following we refer to this approach as GaussianGen.

More in detail, to generate a query we sample a point $\vec{x}$ uniformly at random from S, adding $\mathcal{N}(0, \sigma^2)$ noise on top of each coordinate. We will use different values for σ to control the distance of generated queries from dataset points.

This approach has been used extensively in the literature in the past 30 years to generate synthetic workloads [2, 14, 15, 19–21, 29, 33, 43, 46, 47, 53, 57].

Index data structures. For the sake of generality we consider four different index data structures: two exact indices and two approximate indices. As exact indices we consider MESSI [39] and DSTree [54]. As approximate indices we consider HNSW and IVF in the implementations provided by the `faiss` library [17].

Performance Metrics. First, we assess the relative merits of different hardness measures in terms of their correlation with the empirical hardness. Our main performance measure to evaluate generated workloads is thus the empirical hardness as defined in Section 3.1. Here we detail how we compute this hardness measure for a query $\vec{q}$, dataset S and index data structure $\mathcal{D}$.

Typically, the index $\mathcal{D}$ has several parameters that can be tweaked, resulting in a different levels of performance, possibly at the expense of the accuracy (for approximate indices). For instance, HNSW has parameters controlling the number of neighbors used in the graph, and the depth of the exploration at query time.

For a query $\vec{q}$ and an index $\mathcal{D}$ on a dataset S we perform a grid search of the parameter space of $\mathcal{D}$. For exact indices (i.e. MESSI and DSTree), we pick the configuration that attains the smallest number of distance computations. For approximate indices (i.e. IVF and HNSW), we select among all configurations with recall at least $\rho = 0.95$ the one with the minimum number of distance computations. For brevity, in what follows we denote the empirical hardness as $\mathcal{H}_{\mathcal{D}}$, omitting the subscript $\rho = 0.95$. Note that this is a particularly stringent requirement, on a query level: when $k = 10$ our setup will select the fastest configuration returning the *exact* result, even for approximate indices.

It has been observed [8, 28] that different configurations of the same index data structure can achieve the same average recall with very different runtime performances, when applied to batches of queries. Furthermore, the recall of each query can vary widely [8, Fig 10]. Therefore, in contrast with usual benchmarking setups, we tune each index on a *per query* basis when computing the empirical hardness. While expensive to evaluate, this is to ensure that we get an accurate assessment of the empirical hardness for each query.

Table 1: Absolute value of the Kendall rank correlation coefficient between different explicative measures and the $\mathcal{H}_{IVF}$ empirical hardness (best underlined, second-best in bold).

	astro	deep1b	glove	nytimes	sald	seismic	text2image
$Exp_{20\|10}$	**0.69**	**0.62**	**0.86**	**0.70**	**0.71**	0.35	**0.53**
LID_{10}	0.42	0.37	0.71	0.21	0.35	0.04	0.50
RC_{10}	0.75	0.77	0.91	0.91	0.79	0.81	0.74
$\alpha_{0.05,10}$	0.27	0.08	0.59	0.19	0.13	0.23	0.33
$\alpha_{0.1,10}$	0.19	0.21	0.51	0.32	0.26	0.30	0.40
$\alpha_{0.5,10}$	0.19	0.22	0.23	0.66	0.61	**0.52**	0.45
$\alpha_{1,10}$	0.22	0.28	0.84	0.21	0.38	0.37	0.29

5.1 Evaluating hardness Measures

The aim of our first set of experiments is to capture how well *explicative* hardness measures relate to the empirical hardness. This is instrumental in deciding which measure is better to use to generate query workloads. We consider the query sets that are provided with our benchmark datasets. For each query $\vec{q}$, we consider the empirical hardness $\mathcal{H}_{IVF}(\vec{q})$ for the IVF index implementation provided by `faiss`. Then, we compute the other hardness measures and evaluate the correlation they exhibit with $\mathcal{H}_{IVF}(\vec{q})$ for each dataset. Given that the relation is not linear and that we are only interested in comparing how similar are the *rankings* induced by the hardness measures, we consider the *Kendall* τ rank correlation coefficient. As hardness measures we consider LID_{10}, RC_{10}, $\text{Expansion}_{20|10}$ and ε-hardness with $\varepsilon \in [0.05, 0.1, 0.5, 1]$.

Table 1 reports the absolute value of the Kendall rank correlation coefficient for each of the hardness measures with $\mathcal{H}_{IVF}$. The measure that correlates most consistently with the empirical hardness is the Relative Contrast. In some cases the Expansion and *LID* provide comparable results, but in general they have a lower correlation with $\mathcal{H}_{IVF}$. For the *LID*, this is likely due to its sensitivity to noise in its estimation, especially in regions where nearest neighbors are tightly packed. For the Expansion, its lower correlation can be explained by the fact that it is a very local measure, in that it considers only the distances of the k-th and $2k$-th neighbors. In contrast, the *RC* considers both local (the k-th nearest neighbor) and more global information (the average distance to all points). As for the ε-hardness, we note that its highest correlation with the empirical hardness is achieved for different ε values for each dataset, making it hard to adopt for the purpose of generating workloads. Furthermore, the ε-hardness is rather sensitive to the setting of ε: small changes to ε can lead to large differences in the measure, depending on the data distribution.

Figure 7 in Appendix C gives a more detailed view of the relation between the empirical hardness and the *LID*, *RC* and Expansion. Appendix D reports similar results for the other indices we consider.

Based on these results, we select the Relative Contrast as the hardness measure to be used going forward to guide the workload synthesis.

5.2 Generating Workloads

In this section we report on experiments using the workload generators described in Section 4. Note that we do not consider the method proposed in [60] as the setup is fundamentally different. In [60], in fact, for a given fixed query point $\vec{q}$ the *dataset* is modified

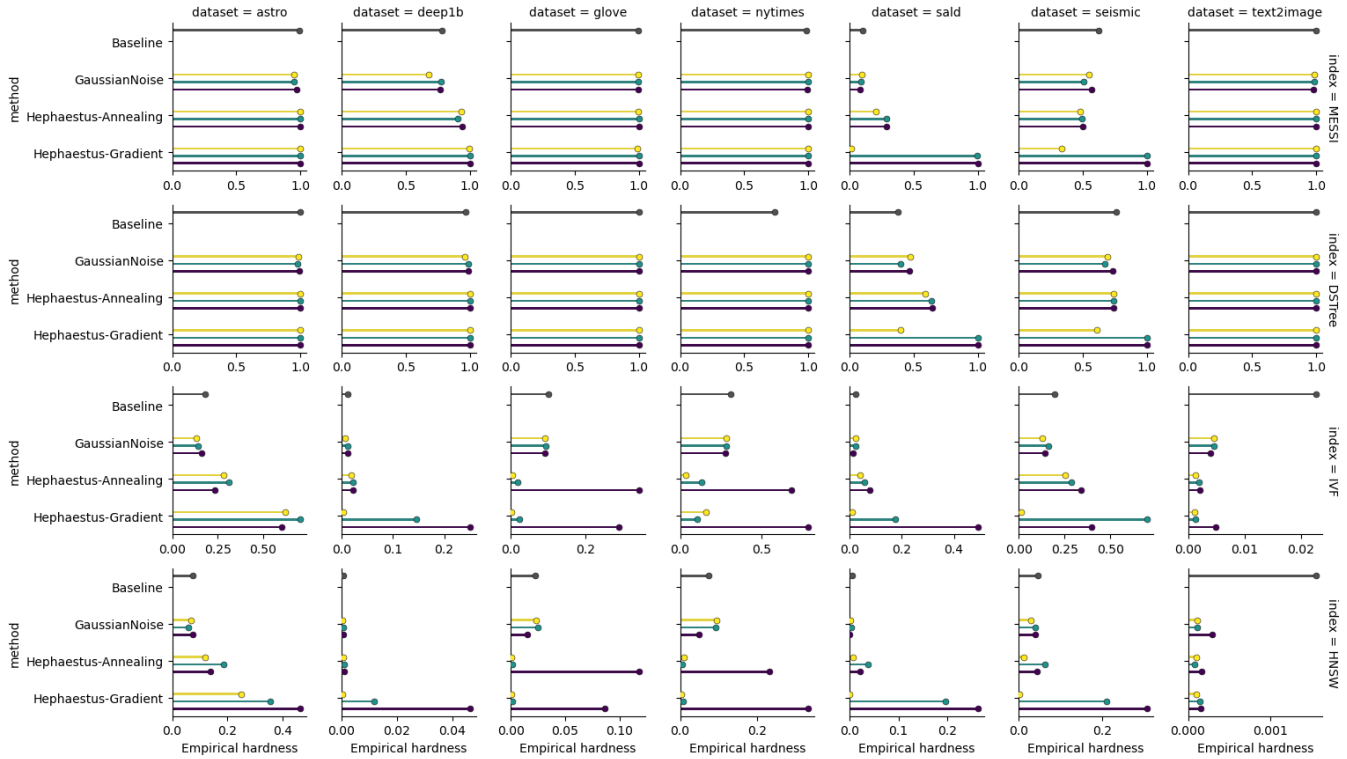


Figure 2: Average empirical hardness for different index data structures, over different datasets and easy, medium, and difficult workloads, for $k = 10$.

so that $\vec{q}$ achieves the desired hardness level. In our setting, instead, we keep the dataset fixed and unmodified while generating query points anew.

Using the methods presented in Section 4 we set out to generate three different workloads for each dataset, deemed *easy*, *medium* and *hard*: the expectation is that index data structures will have to spend more distance computations on the *hard* workload compared to the *easy* one. We remark that we consider different data structures to ensure that our findings are not specific to a single index: we are *not* aiming at comparing different indices.

For a given dataset S and number of nearest neighbors k, let $\overline{RC}_k$ be the average RC of the dataset. The target range of RC for the generators is $1+(\overline{RC}-1)\cdot\gamma \pm 5\%$ where γ is 0.5, 0.1, 0.01 respectively for *easy*, *medium* and *hard* queries for Euclidean distance datasets, and $\gamma = 1.5$ (*easy*), 0.5 (*medium*), 0.25 (*hard*) for Angular distance datasets.

The difference between the target RC values between Euclidean and Angular metric spaces is due to the fact that the Angular distance can only take values between 0 and 2, hence very small RC values are very hard to attain.

For Hephaestus-Annealing we set the initial temperature to 1 and the maximum number of iterations to 2 000. For Hephaestus-Gradient we set the learning rate to 1 and the maximum number of iterations to 1 000 (as we shall see, Hephaestus-Gradient converges faster, hence a smaller maximum number of iterations is reasonable). Both Hephaestus-Annealing and Hephaestus-Gradient, upon reaching the maximum number of iterations, return the last candidate query, whichever its Relative Contrast. For each hardness level, we generate 30 queries with each generator.

As for the GaussianGen generator, which does not explicitly target the Relative Contrast, we consider the diameter Φ of each dataset, setting the standard deviation σ of the added noise tto $\Phi/10^5$, $\Phi/10^4$, and $\Phi/10^3$ for *easy*, *medium*, and *hard* queries, respectively. We generate 100 queries with the GaussianGen generator to account for the larger variability in quality in this queryset.

Figure 2 reports the results in terms of average number of distance computations for $k = 10$. Each panel in the figure reports the results on a particular combination of dataset (arranged in columns) and index data structure (arranged in rows). Each bar represents the empirical hardness of a workload generated with the method reported on the y axis, with colors encoding the level of expected hardness. The `Baseline` entry reports the average number of distance computations on the queryset bundled with each dataset. As such, it is not labelled with any expected hardness, but is included for reference.

First, we observe that the GaussianGen method is not very effective at producing workloads of different empirical hardness, as this measure is comparable between *easy* and *hard* workloads on the same dataset/index pair. In particular, all workloads generated with GaussianGen have an average empirical hardness comparable with the Baseline.

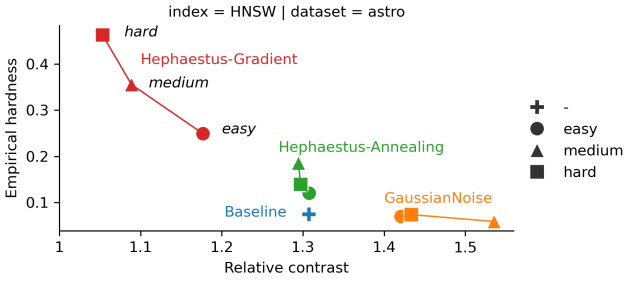


Figure 3: Detailed behavior of the workloads generated by different methods for the `astro` dataset, evaluated by means of HNSW.

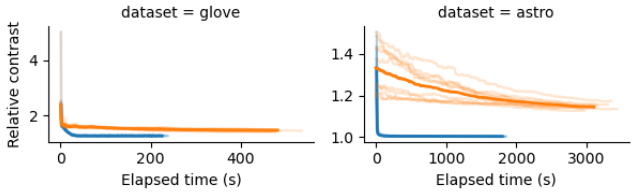


Figure 4: Convergence of the Relative Contrast against the elapsed time for the Hephaestus-Annealing and Hephaestus-Gradient generators.

As for Hephaestus-Annealing and Hephaestus-Gradient, if we focus on approximate indices (IVF and HNSW), we observe that *hard* queries are indeed harder that *easy* ones in terms of empirical hardness, in general. This confirms that generating queries with different target ranges of Relative Contrast translates in queries with different empirical hardness. Furthermore, both methods allow to generate queries that are harder than the Baseline ones, allowing to put more stress on the index data structures.

There are some exceptions: in some cases (e.g. the IVF index on the `astro` dataset), workloads generated with both Hephaestus-Gradient and Hephaestus-Annealing have comparable empirical hardness, irrespective of whether they are supposed to be *easy* or *hard*. Another related phenomenon is that in some cases (e.g. `astro` and index HNSW), the empirical difficulty of workloads generated by Hephaestus-Gradient is higher than Hephaestus-Annealing, although both methods are set to target the same Relative Contrast ranges. To investigate why this happens, consider Figure 3 that reports, for the `astro` dataset and the HNSW index, the relation between the empirical hardness and the relative contrast of workloads generated by the different methods, for $k = 10$. This figure corresponds to the data reported in the bottom left panel of Figure 2. Clearly, the relative contrast is higher for the workloads generated by Hephaestus-Gradient than the ones generated by Hephaestus-Annealing, resulting in a lower empirical difficulty. This is because the generation method did not converge to the target range of RC in the allotted number of iterations. In fact Figure 3 shows that all workloads generated by Hephaestus-Annealing have the same relative contrast. As we shall see this is due to its slow convergence.

As for exact indices (MESSI and DSTree) in most cases the workloads generated by our methods have a high empirical hardness. This stresses the challenge of choosing the right RC range if one wants to generate a query workload with a given empirical hardness for a specific index. We will tackle this issue, and the ones described in the previous paragraph, in Section 5.5.

5.3 Convergence

In this section we set out to investigate how fast Hephaestus-Gradient and Hephaestus-Annealing converge. The setup is as follows: for each dataset we run 10 instances of Hephaestus-Annealing and Hephaestus-Gradient for 1 000 and 200 iterations, respectively, with each instance generating a single candidate query. In each iteration we measure the Relative Contrast of the candidate query and the elapsed time. In this experiment we set the algorithms to minimize the *RC* of generated queries. Given that the smallest possible *RC* is by definition 1, this is achieved by setting the target *RC* range to $[1, 1]$ in Algorithms 1 and 2 and letting the algorithms run until the maximum number of allowed iterations is reached.

Figure 4 reports, for each dataset, the Relative Contrast of each of the 10 candidate queries on the y axis, against the elapsed time in seconds on the x axis using semi-transparent lines. The solid lines report the average behavior across the 10 instances of each of the two algorithms.

Clearly, we can observe that Hephaestus-Gradient converges much faster than Hephaestus-Annealing to values of Relative Contrast close to 1. Therefore, if the hardness measure is differentiable like the Relative Contrast, we recommend using Hephaestus-Gradient.

Observing the elapsed times, we can notice that different datasets require different times to exhaust the allotted number of iterations. Furthermore, Hephaestus-Annealing completes its 1 000 iterations in about twice the time Hephaestus-Gradient completes its 200 iterations, suggesting that each iteration of Hephaestus-Annealing is faster. We will discuss these aspects in the next section.

5.4 Running Time and Scalability

We now consider the running time of both Hephaestus-Annealing and Hephaestus-Gradient in the following setup: we generate 10 queries of *hard* hardness under the Relative Contrast measure and record both the time for each iteration and the overall running time. To test the scalability, we apply the query generation procedures to samples of the datasets with 5, 10 and 15 million points, omitting from the experiment datasets which have less than 5 million points. Table 2 reports the average iteration time and the overall running time, averaged over the 10 generated queries, in seconds.

Clearly, each iteration of Hephaestus-Annealing is faster than those Hephaestus-Gradient, on average. The reason is that the most expensive computation in each iteration of Hephaestus-Annealing is the computation of the distances from the candidate, whereas Hephaestus-Gradient also needs to compute the gradient.

Table 2: Running time of Hephaestus-Annealing (H-Ann) and Hephaestus-Gradient (H-Grad) to produce workloads of hard hardness, both in terms of average time per iteration and overall time.

		Iteration (s)		Total (s)	
	size	H-Ann	H-Grad	H-Ann	H-Grad
astro	5m	2.3	8.1	2303.1	16.1
	10m	5.7	18.5	5704.5	37.0
	15m	9.3	26.4	9333.7	52.7
deep1b	5m	1.9	5.6	1916.5	11.1
	10m	4.6	14.2	4633.3	28.5
	15m	7.4	19.8	7445.0	39.6
sald	5m	2.3	6.2	2342.7	14.3
	10m	4.6	13.2	4550.7	28.9
	15m	7.9	21.5	7878.5	47.4
seismic	5m	3.2	8.2	3156.8	18.9
	10m	6.1	19.9	6140.9	45.7
	15m	9.0	27.6	8988.5	60.7

However, as we saw in Figure 4, Hephaestus-Gradient converges much faster. Hence, the overall running time of Hephaestus-Gradient is much smaller than that of Hephaestus-Annealing.

As for the scalability, each method's iteration scales approximately linearly with the dataset size. This is expected: in both methods the most expensive operation is $O(n)$, with n being the number of points in the dataset.

We omitted from Table 2 the running time for GaussianGen, for the sake of conciseness. From a running time perspective, GaussianGen is much faster than both other approaches, requiring only a few milliseconds to generate each query, irrespective of the dataset size. However, as we have seen in the previous sections, the GaussianGen generator offers no control over the hardness of the queries, and typically generates rather easy workloads.

5.5 Generating Workloads With a Target Empirical hardness

In Section 5.2 we observed that most query workloads are rather hard for exact indices. It might thus be interesting to generate easy queries in order to investigate the characteristics of queries that exact indices find hard. To this end, we apply the variant of Hephaestus-Gradient described in Algorithm 3 to generate a workload with empirical hardness between 0.1 and 0.2 for the MESSI index. We generate 10 queries, using the Relative Contrast as the hardness scoring function ς in Algorithm 3, and the empirical hardness $\mathcal{H}_{\text{MESSI}}$ for the stopping condition. Hence, the Relative Contrast is used to guide the placement of the queries by means of its gradient, and the empirical hardness is used to assess whether the candidate queries satisfy the requirements. We set a maximum of 1 000 steps.

Figure 5 reports the empirical hardness of the queries produced by this process, with each poin representing a single query. Vertical dotted lines mark the target *RC* range. Almost all generated queries exhibit an empirical hardness that is within the requested bound,

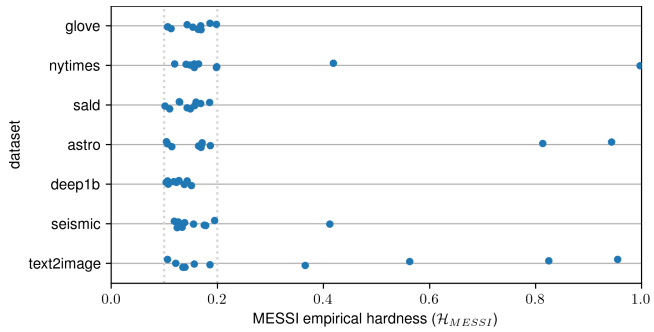


Figure 5: Empirical hardness of a workloads generated with Hephaestus-Gradient, targeting empirical hardness for MESSI in the range $[0.1, 0.2]$. Each dot represents one of the 10 generated queries for each dataset.

on all dataset. There are couple of exceptions: queries that after the allotted 1 000 optimization steps still do not fall in the required $\mathcal{H}_{\text{MESSI}}$ range. We report them for completeness: in practice such queries can be discarded and possibly replaced with other queries generated with another run of Hephaestus-Gradient.

This experiment shows that Hephaestus-Gradient can be used to target a specific hardness for a given index, without requiring prior knowledge of the corresponding range of Relative Contrast values. In Appendix A we will use the queries we just generated to further investigate the behavior of MESSI.

6 Conclusions

In this paper, we considered the problem of evaluating and synthesizing query workloads for benchmarking similarity search approaches, for any kind of similarity search data structure, and for both exact and approximate search. First, we investigated the relation between *explicative* hardness measures with the *empirical* hardness encountered by index data structures. We found that the Relative Contrast is the most consistent metric at characterizing the hardness of a query. Its correlation with the *empirical hardness* is the highest among the tested measures, but is not perfect. Devising a new measure that is both more accurate and easy to compute remains an open problem. We then proposed different methods for generating query workloads. In particular, we found that to generate workloads whose Relative Contrast falls in a given range, our method Hephaestus-Gradient converges rapidly to a solution. The same method can be used to generate queries that are by construction hard or easy for a given index. In future work, we plan to study our method in the context of additional index types and variations [10, 22, 51, 52, 55, 58, 59].

We note that Hephaestus-Gradient is independent of the hardness measure, and will benefit from new hardness measures developed in the future.

Acknowledgments

Supported by EU Horizon projects AI4Europe (101070000), TwinODIS (101160009), ARMADA (101168951), DataGEMS (101188416), RECITALS (101168490), and by *ΥΠΑΙΘΑ* & NextGenerationEU project HARSH (*ΥΠ*3*TA* – 0560901).

References

[1] Baranchuk, Dmitry and Babenko, Artem . [n. d.]. *text2image*. https://research.yandex.com/blog/benchmarks-for-billion-scale-similarity-search

[2] Rakesh Agrawal, Christos Faloutsos, and Arun Swami. 1993. Efficient similarity search in sequence databases. In *FODO*.

[3] Thomas D. Ahle, Martin Aumüller, and Rasmus Pagh. 2017. Parameter-free Locality Sensitive Hashing for Spherical Range Reporting. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, Philip N. Klein (Ed.). SIAM, 239–256. https://doi.org/10.1137/1.9781611974782.16

[4] Amazon. 2024. Private communication. (2024).

[5] Laurent Amsaleg, Oussama Chelly, Teddy Furon, Stéphane Girard, Michael E. Houle, Ken-ichi Kawarabayashi, and Michael Nett. 2018. Extreme-value-theoretic estimation of local intrinsic dimensionality. *Data Min. Knowl. Discov.* 32, 6 (2018), 1768–1805. https://doi.org/10.1007/S10618-018-0578-6

[6] Martin Aumüller, Erik Bernhardsson, and Alexander John Faithfull. 2020. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Inf. Syst.* 87 (2020). https://doi.org/10.1016/J.IS.2019.02.006

[7] Martin Aumüller and Matteo Ceccarello. 2019. The Role of Local Intrinsic Dimensionality in Benchmarking Nearest Neighbor Search. In *SISAP (Lecture Notes in Computer Science, Vol. 11807)*. Springer, 113–127.

[8] Martin Aumüller and Matteo Ceccarello. 2021. The role of local dimensionality measures in benchmarking nearest neighbor search. *Inf. Syst.* 101 (2021), 101807. https://doi.org/10.1016/J.IS.2021.101807

[9] Martin Aumüller and Matteo Ceccarello. 2023. Recent Approaches and Trends in Approximate Nearest Neighbor Search, with Remarks on Benchmarking. *IEEE Data Eng. Bull.* 46, 3 (2023), 89–105.

[10] Ilias Azizi, Karima Echihabi, and Themis Palpanas. 2025. Graph-Based Vector Search: An Experimental Evaluation of the State-of-the-Art. *Proc. ACM Manag. Data* 3, 1 (2025), 43:1–43:31.

[11] Anthony J. Bagnall, Richard L. Cole, Themis Palpanas, and Konstantinos Zoumpatianos. 2019. Data Series Management (Dagstuhl Seminar 19282). *Dagstuhl Reports* 9, 7 (2019), 24–39.

[12] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018. *JAX: composable transformations of Python+NumPy programs*. http://github.com/google/jax

[13] Sebastian Bruch, Franco Maria Nardini, Cosimo Rulli, and Rossano Venturini. 2024. Efficient Inverted Index-based Approximate Retrieval over High-dimensional Learned Sparse Representations. *IEEE Data Eng. Bull.* 48, 3 (2024), 43–62.

[14] Alessandro Camerra, Jin Shieh, Themis Palpanas, Thanawin Rakthanmanon, and Eamonn Keogh. 2013. Beyond one billion time series: indexing and mining very large time series collections with iSAX2+. *KAIS* (2013).

[15] Kin-Pong Chan and A.W.-C. Fu. 1999. Efficient time series matching by wavelets. In *ICDE*.

[16] DeepMind, Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, Antoine Dedieu, Claudio Fantacci, Jonathan Godwin, Chris Jones, Ross Hemsley, Tom Hennigan, Matteo Hessel, Shaobo Hou, Steven Kapturowski, Thomas Keck, Iurii Kemaev, Michael King, Markus Kunesch, Lena Martens, Hamza Merzic, Vladimir Mikulik, Tamara Norman, George Papamakarios, John Quan, Roman Ring, Francisco Ruiz, Alvaro Sanchez, Laurent Sartran, Rosalia Schneider, Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Miloš Stanojević, Wojciech Stokowiec, Luyu Wang, Guangyao Zhou, and Fabio Viola. 2020. *The DeepMind JAX Ecosystem*. http://github.com/google-deepmind

[17] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library. *CoRR* abs/2401.08281 (2024).

[18] Karima Echihabi, Kostas Zoumpatianos, and Themis Palpanas. 2020. Scalable Machine Learning on High-Dimensional Vectors: From Data Series to Deep Network Embeddings. In *WIMS*. ACM, 1–6.

[19] Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. 2018. The Lernaean Hydra of Data Series Similarity Search: An Experimental Evaluation of the State of the Art. *Proc. VLDB Endow.* 12, 2 (2018), 112–127.

[20] Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. 2019. Return of the Lernaean Hydra: Experimental Evaluation of Data Series Approximate Similarity Search. *Proc. VLDB Endow.* 13, 3 (2019), 403–420.

[21] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. 1994. Fast Subsequence Matching in Time-series Databases. In *SIGMOD*.

[22] Jianyang Gao, Yutong Gou, Yuexuan Xu, Jifan Shi, Cheng Long, Raymond Chi-Wing Wong, and Themis Palpanas. 2024. High-Dimensional Vector Quantization: General Framework, Recent Advances, and Future Directions. *IEEE Data Eng. Bull.* 48, 3 (2024), 3–19. http://sites.computer.org/debull/A24sept/p3.pdf

[23] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. Exploring Network Structure, Dynamics, and Function using NetworkX. In *Proceedings of the 7th Python in Science Conference*, Gaël Varoquaux, Travis Vaught, and Jarrod Millman (Eds.). Pasadena, CA USA, 11 – 15.

[24] Junfeng He, Sanjiv Kumar, and Shih-Fu Chang. 2012. On the Difficulty of Nearest Neighbor Search. In *ICML*. icml.cc / Omnipress.

[25] Michael E. Houle. 2013. Dimensionality, Discriminability, Density and Distance Distributions. In *ICDM*. 468–473. https://doi.org/10.1109/ICDMW.2013.139

[26] I.R.I. for Seismology with Artificial Intelligence. 2019. *Seismic Data Access. http://ds.iris.edu/data/access/*. http://ds.iris.edu/data/access/

[27] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Trans. Pattern Anal. Mach. Intell.* 33, 1 (2011).

[28] Elias Jääsaari, Ville Hyvönen, Matteo Ceccarello, Teemu Roos, and Martin Aumüller. 2025. VIBE: Vector Index Benchmark for Embeddings. arXiv:2505.17810 [cs.LG] https://arxiv.org/abs/2505.17810

[29] Shrikant Kashyap and Panagiotis Karras. 2011. Scalable kNN search on vertically stored time series. In *KDD*.

[30] Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG]

[31] Elizaveta Levina and Peter J. Bickel. 2004. Maximum Likelihood Estimation of Intrinsic Dimension. In *NIPS*. 777–784.

[32] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2020. Approximate Nearest Neighbor Search on High Dimensional Data - Experiments, Analyses, and Improvement. *TKDE* 32, 8 (2020), 1475–1488.

[33] Michele Linardi and Themis Palpanas. 2018. Scalable, Variable-Length Similarity Search in Data Series: The ULISSE Approach. *Proc. VLDB Endow.* 11, 13 (2018).

[34] Yury A. Malkov and Dmitry A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 42, 4 (2020), 824–836.

[35] David Newman. 2008. Bag of Words. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5ZG6P.

[36] Themis Palpanas. 2020. Evolution of a Data Series Index: The iSAX Family of Data Series Indexes. *Communications in Computer and Information Science (CCIS)* 1197 (2020).

[37] Themis Palpanas and Volker Beckmann. 2019. Report on the First and Second Interdisciplinary Time Series Analysis Workshop (ITISA). *SIGMOD Rec.* 48, 3 (2019), 36–40.

[38] Botao Peng, Panagiota Fatourou, and Themis Palpanas. 2018. ParIS: The Next Destination for Fast Data Series Indexing and Query Answering. In *IEEE BigData*.

[39] Botao Peng, Panagiota Fatourou, and Themis Palpanas. 2020. MESSI: In-Memory Data Series Indexing. In *ICDE*. IEEE, 337–348.

[40] Botao Peng, Panagiota Fatourou, and Themis Palpanas. 2021. Fast data series indexing for in-memory data. *VLDB J.* 30, 6 (2021), 1041–1067.

[41] Botao Peng, Panagiota Fatourou, and Themis Palpanas. 2021. ParIS+: Data Series Indexing on Multi-Core Architectures. *TKDE* 33, 5 (2021), 2151–2164.

[42] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*. 1532–1543.

[43] Davood Rafiei and Alberto Mendelzon. 1997. Similarity-based Queries for Time Series Data. In *SIGMOD*.

[44] S. C. Vision. 2019. *Deep billion-scale indexing. http://sites.skoltech.ru/compvision/noimi*. http://sites.skoltech.ru/compvision/noimi

[45] S. University. 2019. *Southwest University Adult Lifespan Dataset (SALD). http://fcon_1000.projects.nitrc.org/indi/retro/sald.html?utm_source=newsletter&utm_medium=email&utm_content=SeeData&utm_campaign=indi-1*. http://fcon_1000.projects.nitrc.org/indi/retro/sald.html

[46] Patrick Schäfer and Mikael Högqvist. 2012. SFA: A Symbolic Fourier Approximation and Index for Similarity Search in High Dimensional Datasets. In *EDBT*.

[47] Jin Shieh and Eamonn J. Keogh. 2008. *i*SAX: indexing and mining terabyte sized time series. In *KDD*. ACM, 623–631.

[48] Harsha Vardhan Simhadri, George Williams, Martin Aumüller, Matthijs Douze, Artem Babenko, Dmitry Baranchuk, Qi Chen, Lucas Hosseini, Ravishankar Krishnaswamy, Gopal Srinivasa, Suhas Jayaram Subramanya, and Jingdong Wang. 2021. Results of the NeurIPS'21 Challenge on Billion-Scale Approximate Nearest Neighbor Search. In *NeurIPS (Competition and Demos)*, Vol. 176. 177–189.

[49] Josef Sivic and Andrew Zisserman. 2003. Video Google: A Text Retrieval Approach to Object Matching in Videos. In *ICCV*. IEEE Computer Society, 1470–1477.

[50] Soldi, S., Beckmann, V., Baumgartner, W. H., Ponti, G., Shrader, C. R., Lubi´nski, P., Krimm, H. A., Mattana, F., and Tueller, J. 2014. Long-term variability of AGN at hard X-rays. *A&A* 563 (2014), A57. https://doi.org/10.1051/0004-6361/201322653

[51] Théophane Vallaeys, Matthew J. Muckley, Jakob Verbeek, and Matthijs Douze. 2025. Qinco2: Vector Compression and Search with Improved Implicit Neural Codebooks. In *ICLR*.

[52] Qitong Wang, Ioana Ileana, and Themis Palpanas. 2025. LeaFi: Data Series Indexes on Steroids with Learned Filters. *Proc. ACM Manag. Data* 3, 1 (2025), 51:1–51:27.

[53] Qitong Wang and Themis Palpanas. 2021. Deep Learning Embeddings for Data Series Similarity Search. In *KDD*. ACM, 1708–1716.

[54] Yang Wang, Peng Wang, Jian Pei, Wei Wang, and Sheng Huang. 2013. A Data-adaptive and Dynamic Segmentation Index for Whole Matching on Time Series. *Proc. VLDB Endow.* 6, 10 (2013), 793–804.

[55] Zeyu Wang, Peng Wang, Themis Palpanas, and Wei Wang. 2023. Graph- and Tree-based Indexes for High-dimensional Vector Similarity Search: Analyses,

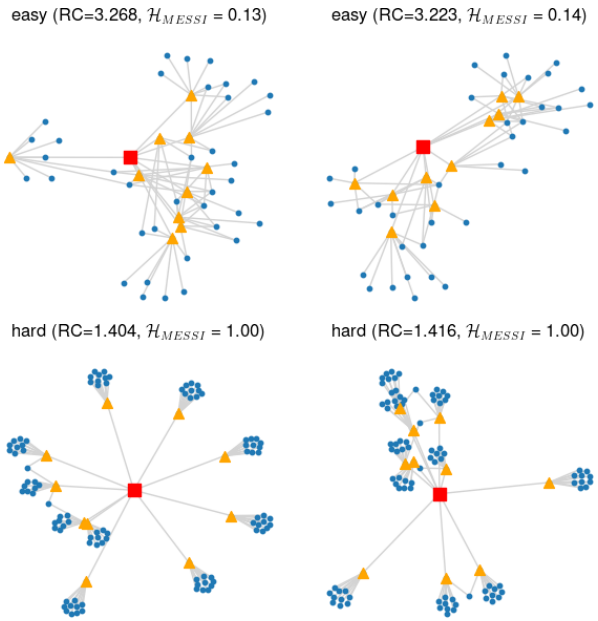


Figure 6: Visualization of the local neighborhood of four different queries on glove, with different empirical hardness related to MESSI. The top two queries are easy, while the bottom two queries are hard. Red squares represent queries, orange triangles are the 10-nearest neighbors, blue circles are the neighbors of the neighbors.

Comparisons, and Future Directions. *IEEE Data Eng. Bull.* 46, 3 (2023), 3–21.
[56] Zeyu Wang, Qitong Wang, Xiaoxing Cheng, Peng Wang, Themis Palpanas, and Wei Wang. 2024. Steiner-Hardness: A Query Hardness Measure for Graph-Based ANN Indexes. *Proc. VLDB Endow.* 17, 13 (2024), 4668–4682.
[57] Zeyu Wang, Qitong Wang, Peng Wang, Themis Palpanas, and Wei Wang. 2023. Dumpy: A Compact and Adaptive Index for Large Data Series Collections. *Proc. ACM Manag. Data* 1, 1 (2023), 111:1–111:27.
[58] Jiuqi Wei, Xiaodong Lee, Zhenyu Liao, Themis Palpanas, and Botao Peng. 2025. Subspace Collision: An Efficient and Accurate Framework for High-dimensional Approximate Nearest Neighbor Search. *Proc. ACM Manag. Data* 3, 1 (2025), 79:1–79:29.
[59] Jiuqi Wei, Botao Peng, Xiaodong Lee, and Themis Palpanas. 2024. DET-LSH: A Locality-Sensitive Hashing Scheme with Dynamic Encoding Tree for Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 17, 9 (2024), 2241–2254.
[60] Kostas Zoumpatianos, Yin Lou, Ioana Ileana, Themis Palpanas, and Johannes Gehrke. 2018. Generating data series query workloads. *VLDB J.* 27, 6 (2018), 823–846.

A Case Study

To study how the measures and workload generators considered in this paper can help in investigating the performance of index data structures, we consider the MESSI index and the glove dataset, for $k = 10$. In particular, we consider four queries generated by Hephaestus-Gradient. The first two queries are easy, from the workload generated in the previous section, with empirical hardness $\mathcal{H}_{MESSI} \in [0.1, 0.2]$ and Relative Contrast > 3. The other two queries are hard, generated from the Baseline workload associated with the dataset, with $\mathcal{H}_{MESSI} \approx 1$ and Relative Contrast < 1.5.

To further characterize the structure of these queries, for each of them we consider the graph consisting of the query, its k-nearest neighbors (which we deem *immediate* neighbors), and also each neighbor's k-nearest neighbors. In other words, we consider the subgraph of the k-nearest neighbor graph comprising the nodes at hop count at most 2 from the query. The intuition is that by comparing the immediate neighborhood of the query with the neighborhood of its k-nearest neighbors we can reason about the behavior of the index.

Figure 6 shows this graph for the four aforementioned queries. In particular, the red square is the query, and the orange triangles are its k-nearest neighbors (i.e. the answer to the query). The blue dots are the neighbors of each one of the k-nearest neighbors of the query. The graphs are laid out using the *spring* layout from the networkx library [23]. The length of the drawn edge is influenced by the edge's weight, which we set to the distance between the points the nodes represent. Therefore, short edges connect nodes corresponding to similar points. It is worth reminding that this is a 2-dimensional visualization of 100-dimensional points: as such nodes that appear to be close together but *are not* connected by an edge are not, in fact, close in the original space.

The top pair of graphs in Figure 6 represents the two easy queries, whereas the bottom pair is for the two hard queries. The graphs are remarkably different when comparing hard and easy queries, and similar when comparing queries of the same hardness.

For easy queries, we notice that the nearest neighbors of the query are also mostly nearest neighbors of each other. On hard queries the opposite is true: the query's neighbors share very little and their own neighborhoods are quite dissimilar from one another.

Furthermore, on easy queries the immediate neighbors are comparably similar to the query and among them, while being reasonably different from the rest of the dataset, as suggested by the Relative Contrast score above 3.2. Conversely, on hard queries the immediate neighbors are closer to their own neighbors than they are to the query, and from the perspective of the query they are similar to the other points of the dataset, as suggested by the Relative Contrast score close to 1.4.

Consider now that MESSI partitions the dataset into a tree data structure by means of SAX words, i.e. symbolic representations of reduced dimensionality of the input vectors. Crucially, these symbolic representations can be used to approximate the distance of the vectors they represent, allowing to restrict the computation of the distance to fewer candidates. Considering the examples in Figure 6, we have that for easy queries the symbolic representation of the query and its neighbors are very similar, while being at the same time dissimilar from the others in the dataset. For hard queries, though, the approximate distance between symbolic representations is not accurate enough to allow a meaningful pruning of the candidates. An interesting avenue of research is then to allow MESSI to better handle this latter case. A possibility might be to switch to a different space partitioning scheme for this scenario.

B Additional pseudocode

We report here the pseudocode (Algorithm 3) for the adaptation of Hephaestus-Gradient targeting the empirical hardness.

C Details on correlations

Figure 7 gives a more detailed view of the relation between the empirical hardness $\mathcal{H}_{IVF}$ and the *LID*, *RC* and Expansion, which

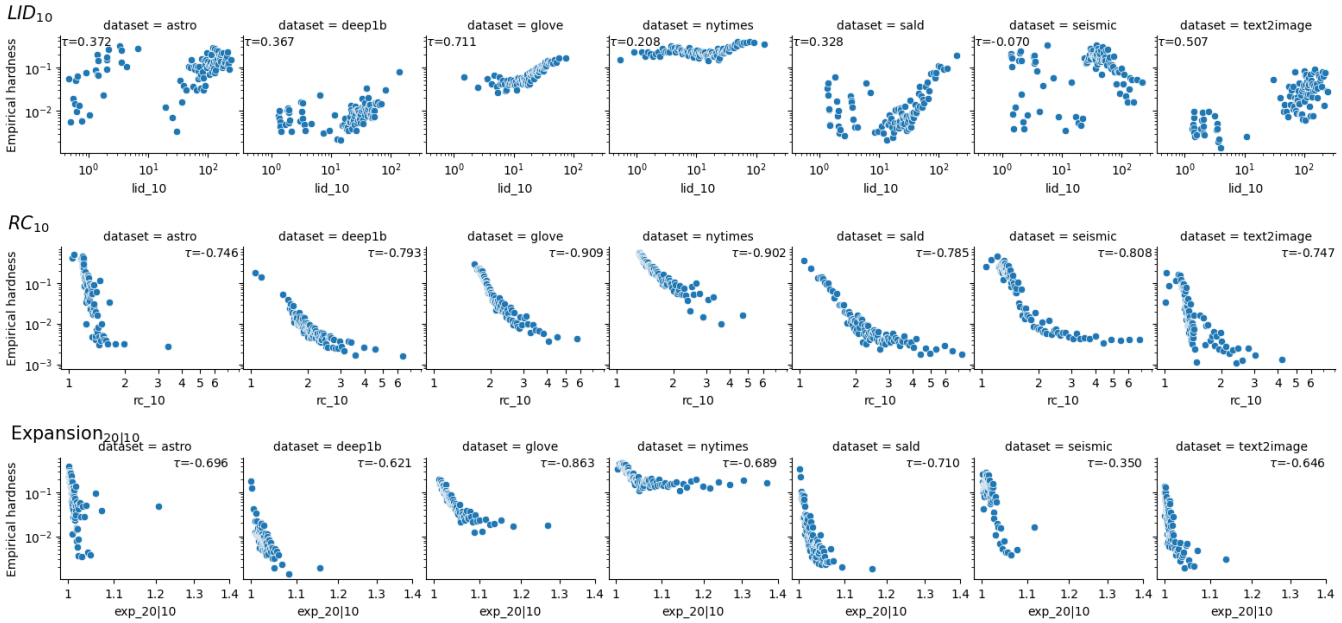


Figure 7: Scatterplots of the relation between *LID* (top row), *RC* (mid row), Expansion (bottom row) and empirical hardness. The τ reported in each plot is the Kendall rank-correlation coefficient.

Algorithm 3: HEPHAESTUS-GRADIENT for empirical hardness

Input: Dataset S; starting point $\vec{q}$; hardness scoring function $\varsigma : (\mathcal{X}, S) \to \mathbb{R}$; index data structure $\mathcal{D}$ for S; learning rate η; maximum number of iterations *max_iter*; target empirical hardness range $[h_l, h_h]$.

```
for i ← 1 to max_iter do
    h ← H_D(q);
    if h_l ≤ h ≤ h_h then return q ;
    else if h < h_l then q ← q + η∇ς(S, q) ;
    else q ← q − η∇ς(S, q) ;
return q;
```

was discussed in Section 5.1, using a logarithmic scale on both axes. The plots confirm that the Relative Contrast is the measure with the strongest association with the empirical hardness in most cases. Interestingly, the Expansion takes values that are very concentrated towards 1, making the association with the empirical hardness very steep. As for the *LID*, an association with the empirical hardness $\mathcal{H}_{IVF}$ can be discerned from the plot, albeit with the presence of many outliers.

D Other empirical hardness measures

For completeness, Table 3 reports the absolute values of the Kendall rank-correlation coefficient between the explicative measures and different empirical hardness measures, respectively $\mathcal{H}_{HNSW}$, $\mathcal{H}_{MESSI}$, and $\mathcal{H}_{DSTree}$. The highest correlation is underlined, the second-highest is in bold. We observe that for both HNSW and MESSI the explicative hardness measure with the highest correlation is the Relative Contrast. On the other hand, for DSTREE the explicative hardness measure with the highest correlation is the Local Intrinsic Dimensionality.

Table 3: Absolute value of the Kendall rank correlation coefficient between explicative and empirical hardness (best underlined, second-best in bold).

		astro	deep1b	glove	nytimes	sald	seismic	text2image
$\mathcal{H}_{\text{HNSW}}$	$Exp_{20\|10}$	**0.65**	**0.68**	**0.84**	**0.77**	0.75	0.28	**0.57**
	LID_{10}	0.32	0.45	0.77	0.35	0.37	0.04	0.55
	RC_{10}	0.82	0.73	0.88	0.88	**0.72**	0.79	0.67
	$\alpha_{0.05,10}$	0.26	0.22	0.42	0.15	0.02	0.36	0.19
	$\alpha_{0.1,10}$	0.08	0.17	0.44	0.16	0.26	**0.62**	0.34
	$\alpha_{0.5,10}$	0.27	0.48	0.16	0.51	0.51	0.30	0.17
	$\alpha_{1,10}$	0.00	0.42	0.79	0.18	0.27	0.30	0.34
$\mathcal{H}_{\text{MESSI}}$	$Exp_{20\|10}$	**0.49**	0.61	**0.69**	0.15	0.80	0.21	0.40
	LID_{10}	0.38	0.32	0.69	**0.20**	0.36	0.10	**0.37**
	RC_{10}	0.69	0.92	0.65	0.07	0.92	0.94	0.36
	$\alpha_{0.05,10}$	0.02	0.09	0.49	0.33	0.15	0.12	0.08
	$\alpha_{0.1,10}$	0.02	0.08	0.16	0.01	0.02	0.43	0.10
	$\alpha_{0.5,10}$	0.25	0.53	0.44	0.08	**0.90**	0.68	0.35
	$\alpha_{1,10}$	0.14	**0.62**	0.14	0.01	0.65	**0.74**	0.25
$\mathcal{H}_{\text{DSTREE}}$	$Exp_{20\|10}$	0.36	0.12	0.11	0.15	0.11	0.20	0.11
	LID_{10}	0.52	0.48	0.01	0.17	0.39	0.56	0.16
	RC_{10}	0.22	0.20	0.03	0.10	0.05	0.07	0.04
	$\alpha_{0.05,10}$	0.32	0.34	0.59	**0.28**	0.26	**0.37**	0.33
	$\alpha_{0.1,10}$	**0.43**	**0.47**	0.52	0.33	**0.36**	0.28	**0.29**
	$\alpha_{0.5,10}$	0.10	0.46	**0.58**	0.08	0.10	0.07	0.17
	$\alpha_{1,10}$	0.05	0.33	0.36	0.02	0.05	0.06	0.14