Information Sciences 619 (2023) 208–234

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Unconventional application of k-means for distributed approximate similarity search ☆

Felipe Ortega [a,*], Maria Jesus Algar [a], Isaac Martín de Diego [a], Javier M. Moguerza [a]

[a] DSLAB, Research Centre for Intelligent Information Technologies (CETINIA), Rey Juan Carlos University, C/Tulipán s/n, Móstoles, 28933 Madrid, Spain

ARTICLE INFO

Article history:
Received 28 July 2022
Received in revised form 6 November 2022
Accepted 8 November 2022
Available online 15 November 2022

Keywords:
Data indexing
Approximate similarity search
Metric distance
Unsupervised learning
Distributed computing
k-means

ABSTRACT

Similarity search based on a distance function in metric spaces is a fundamental problem for many applications. Queries for similar objects lead to the well-known machine learning task of nearest-neighbours identification. Many data indexing strategies, collectively known as Metric Access Methods (MAM), have been proposed to speed up these queries. Moreover, since exact approaches to solving similarity queries can be complex and time-consuming, alternative options have emerged to reduce query execution time, such as returning approximate results or resorting to distributed computing platforms. In this paper, we introduce MASK (Multilevel Approximate Similarity search with *k*-means), an unconventional application of the *k*-means algorithm as the foundation of a multilevel index structure for approximate similarity search suitable for metric spaces. We show that this method leverages inherent properties of *k*-means for this purpose, like representing high-density data areas with fewer prototypes. An implementation of this new indexing procedure is evaluated using a synthetic dataset and two real-world datasets in high-dimensional and high-sparsity spaces. Experimental tests show that MASK performs better than alternative algorithms for approximate similarity search. Results are promising and underpin the applicability of this novel indexing method in multiple domains.

1. Introduction

Similarity search, also known as *proximity search*, [1–3] is a cornerstone for applications in many different fields, such as databases, information retrieval [4], distributed data processing, computer vision [5] and bioinformatics [6], among others. Elements from a dataset are represented by *feature vectors* in a multidimensional space, and the goal is to find which elements are similar (close) to a given query object, subject to a certain measure of similarity or, conversely, dissimilarity.

More formally, let $\mathscr{X}$ be the domain of elements represented by p descriptive features and $s : \mathscr{X} \times \mathscr{X} \mapsto \mathbb{R}$ a similarity function that, for each pair of elements $a, b \in \mathscr{X}$, returns $s(a, b) \in \mathbb{R}$, a similitude score between these two elements. In some cases, it is more convenient to define an equivalent dissimilarity function $\delta : \mathscr{X} \times \mathscr{X} \mapsto \mathbb{R}$ so that for $a, b, c \in \mathscr{X}$, it holds that $s(a, b) > s(a, c) \iff \delta(a, b) < \delta(a, c)$. The pair $(\mathscr{X}, \delta)$ is known as a *dissimilarity space*, a kind of topological space [7]. In addition,

☆ This work was supported by the following research grants of the Spanish Ministry of Science and Innovation: MODAS-IN (Ref. RTI2018-094269-B-I00) and XMIDAS (Ref. PID2021-122640OB-I00).

* Corresponding author.
E-mail addresses: felipe.ortega@urjc.es (F. Ortega), mariajesus.algar@urjc.es (M.J. Algar), isaac.martin@urjc.es (I. Martín de Diego), javier.moguerza@urjc.es (J.M. Moguerza).

https://doi.org/10.1016/j.ins.2022.11.024
0020-0255/

δ is a *metric*, usually termed as a *distance* [8], if it satisfies the properties: $\delta(a,b) \geqslant 0$ (non-negativity); $\delta(a,a) = 0$ (reflexivity); $\delta(a,b) = \delta(b,a)$ (symmetry); and $\delta(a,b) + \delta(b,c) \geqslant \delta(a,c)$ (triangle inequality).

Given a query object $q \in \mathscr{X}$, the goal of similarity search is to resolve some of the following queries [2]:

- *Point query*: Finding elements in $\mathscr{X}$ with exactly the same feature values as q.
- *Range query*: Retrieving a subset of elements $\{o_i\} \in \mathscr{X}$ whose feature values lie within the scope of a given similarity threshold r around q, so that $\forall o_i,\ \delta(o_i,q) \leqslant r$.
- *Nearest-neighbours query*: Recovering elements in $\mathscr{X}$ whose features are the most similar to the feature vector representing q. In this case, we may be interested in finding the single most similar element to q (*nearest neighbour*), denoted by NN_q, or the k closest elements to q (k-*nearest neighbours*), denoted by $k-NN_q$.

Numerous data indexing methods, collectively known as *access methods*, have been proposed to speed up similarity queries. These methods create an index structure to partition the whole domain $\mathscr{X}$ into different regions according to a distance function. After this, users employ different search algorithms to solve similarity queries using this index. There is a wide range of access methods, including exact approaches, approximate solutions and variants tailored to distributed computing systems.

Exact similarity search methods identify the completely accurate result set for a query. In vector spaces, multi-attribute access methods use the absolute position of objects to group elements and search for similar instances. In a p-dimensional Euclidean space, these indexing structures are jointly known as *multidimensional access methods*. These can be classified as Point Access Methods (PAM) for elements without a spatial extension and Spatial Access Methods (SAM) to search extended elements such as lines, polygons or higher-dimensional polyhedra. In the research literature, the general acronym SAM commonly designates both classes of indexing strategies in vector spaces [4].

However, SAM present several limitations. First, in applications like textual and multimedia databases, bioinformatics or pattern recognition, elements in $\mathscr{X}$ cannot always be described in a vector space. Second, to compare the similarity between any two elements, we must use a distance function that avoids introducing any correlation between feature values [9]. A general distance function that meets this requirement is the Minkowski distance [8]. The *Manhattan* or *city block* distance (L_1 norm), the *Euclidean* distance (L_2 norm) and the *Chebyshev* distance (L_∞ norm) are typical instances of this family. Third, SAM are prepared for data with spatial components or represented in a vector space with a relatively low number of dimensions. In high-dimensional spaces, their data partition algorithms become unusable due to the *curse of dimensionality* [10]. Hence, all elements tend to be very far apart, regardless of the distance function chosen for the index.

Feature selection or dimensionality reduction techniques, like multidimensional scaling, can mitigate this problem [11]. However, these approaches are only helpful for elements represented in a relatively low number of dimensions. The so-called *intrinsic dimensionality* of a dataset $\mathscr{X}$ accounts for this notion of "effective dimensions", and is given by $\rho = \frac{\mu^2}{2\sigma^2}$, where μ and σ^2 are the mean and variance of the distribution of distance values between any pair of elements in $\mathscr{X}$, respectively.

Metric Access Methods (MAM) provide a more general framework for similarity search problems in metric spaces [1,3]. If $d : \mathscr{X} \times \mathscr{X} \mapsto \mathbb{R}$ is a distance function (hence, a metric), then the pair $(\mathscr{X}, d)$ defines a metric space. In this setting, MAM exploit the triangle inequality to partition the metric space into subspaces to filter out portions of the dataset that cannot contain valid results. This more general framework subsumes SAM, since every normed vector space induces a metric. Additionally, alternative frameworks can leverage different properties, like Ptolemaic Access Methods [12], that substitute the triangle inequality for Ptolemy's inequality and use distances where it is applicable.

Nevertheless, MAM also present limitations. There is no topological information about the application domain in certain complex problems. Therefore, only non-metric similarity or dissimilarity functions are available [13]. Examples include cosine similarity in information retrieval, dynamic time warping in time series and various edit distances in computational biology. This lack of information about the underlying problem impedes the development of MAM for those cases. Additionally, MAM exact search can be expensive in terms of computation time and updating the index structure due to dynamic data. Moreover, in high-dimensional problems, MAM indexes have serious issues narrowing down the search for candidate elements in a query. As a result, these methods usually default to a sequential scan. In this situation, users will accept a trade-off solution sacrificing accuracy for a significant reduction in search time.

Approximate indexing methods implement compromise solutions for similarity search that may introduce some error in results, although not necessarily. In exchange, they provide faster response time by computing fewer distance values or decreasing the usage of computing resources to complete the search. These methods apply to either vector spaces or metric spaces [14]. This paper introduces MASK (Multilevel Approximate Similarity search with k-means), a novel indexing method based on a multilevel design for approximate similarity search. This method involves an unconventional application of the k-means partitioning algorithm [15,16] to quickly reduce the number of regions to check when searching for results. The input to k-means is always a fixed number of k *prototype points* or *centroids* which, following an iterative process, eventually divide the dataset into k groups (Voronoi cells) by assigning each point to its closest centroid, according to a given distance function.

Let us assume $C_1, C_2, \ldots, C_K$ represent K sets, each containing the indices of observations belonging to each cluster. These sets must satisfy that $C_1 \cup C_2 \cup \ldots C_K = \{1, \ldots, n\}$. That is, each data element must be assigned to at least one of the K clusters. Moreover, for any two clusters k and $k\prime$, the clusters do not overlap between them so that no data element can be assigned to more than one cluster, that is, $C_k \cap C_{k\prime} = \varnothing$. The goal of k-means is to minimize the *within-cluster variation*,

denoted as $W(C_k)$ and defined as the amount by which observations corresponding to a cluster C_k differ from each other, given by:

$$\underset{\{C_k\}_1^K}{\operatorname{argmin}} \left\{ \sum_{k=1}^{K} W(C_k) \right\}.$$

For instance, in the specific case of the Euclidean distance between elements in $\mathbb{R}^p$, the optimization problem set out in k-means is:

$$\underset{\{C_k\}_1^K}{\operatorname{argmin}} \left\{ \sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{a,b \in C_k} \sum_{j=1}^{p} (a_j - b_j)^2 \right\}.$$

The typical choice in clustering problems is finding a *low value* of k that generates different groups in the dataset. In contrast, MASK adopts an unconventional approach to initialize k-means. Instead of setting a low number of k centroids for each level, as it would be customary in clustering problems, we force k-means to generate a *high number* of centroids. Each centroid, as a prototype point, represents all elements assigned to its cluster. Thus, by creating a high number of centroids, we represent underlying data points with finer detail. In the same way, our indexing method applies to any metric space, including the particular case of vector spaces. We show that this unusual application of k-means, different from its standard usage in clustering problems, has attractive properties and renders superior performance, even with high-dimensional datasets residing in feature spaces that are usually sparse.

Scalability is another severe limitation for many indexing methods that struggle to work with large datasets [17]. Indexes following a top-down data partition strategy are inadequate for big data problems in distributed systems, where it is unfeasible to centralize metadata on a single node. Instead, successful indexing methods for large datasets usually follow a bottom-up partitioning strategy. We propose the same approach in MASK, which makes it suitable for distributed computing applications. In this case, an independent, multilevel index structure can be created for each data partition without the need for any information exchange between computing nodes to build the indexes or solve queries. At search time, the top-level index on each node discards partitions that cannot contain a valid answer to speed up the retrieval of candidate elements.

The rest of the paper is organized as follows. Section 2 reviews previous research related to multidimensional data indexing, particularly MAM and, within them, those based on clustering algorithms. Section 3 describes MASK, the novel indexing method based on an unconventional application of k-means. Section 4 presents the results of two empirical experiments to gain intuition about the properties exploited by this new indexing method and evaluate its performance in high-dimensional, high-sparsity problems. In addition, it shows performance results of MASK compared with other alternative algorithms for approximate similarity search. Section 5 discusses design considerations and possible applications of this indexing method, such as for similarity search on distributed computing platforms. Finally, in Section 6, we recap the main conclusions for this work and lay out promising lines for future work.

1.1. Contributions

MASK follows a multilevel, bottom-up design, similar to several methods found in previous research (see Section 2 for further details). However, unlike existing proposals for approximate indexing, it is based on an unconventional application of the k-means algorithm that can take advantage of increased resources in modern computing systems and distributed platforms. This strategy assigns data points to closer k-means centroids distributed in the feature space according to data density in different regions. This way, MASK improves the performance of similarity search queries in high-dimensional problems, which often exhibit high sparsity. In consequence, two distinctive traits characterize this novel algorithm:

- A bottom-up approach drives the creation of the multilevel structure of clusters. As a result, index construction can be distributed among several computing nodes, where each node stores one or several data partitions. Furthermore, we show that this procedure lets the index recover underlying patterns in the dataset represented, in the general case, via pairwise dissimilarities between elements that define a dissimilarity space [7].
- Rather than determining a restrained number of prototype points (k-means centroids) to cluster elements at each level, our indexing method *bombards* each data partition with *as many centroids* as available resources on each computing node can afford. After centroids reach their final location, they become the new set of elements to be clustered at the next layer above, using again as many centroids as possible. MASK repeats this procedure at each index level until the set of centroids at the final top level reaches a manageable size.

2. Background and related work

This section outlines the main concepts and strategies that constitute the basis for different similarity search methods. It also describes featured access methods related to MASK. The main goal is to contextualize our work with previous research in this area and highlight the major novelties introduced by MASK compared with existing approaches.

2.1. Exact similarity search

As described above, access methods for exact similarity search include SAM (Spatial Access Methods) and MAM (Metric Access Methods). Extensive surveys exist comparing dozens of SAM indexing algorithms [18,10]. Archetypal examples include the B-Tree [19], Bloom filters [20], the k-d tree [21], as well as the R-tree [22] and its variants.

Except for a few exceptions, these methods perform poorly when partitioning high-dimensional representation spaces. Experiments by Nene and Kayar [23] suggest that they become unusable for more than 15 dimensions. In that case, or when there is no explicit representation of elements in the set and only a distance function is available to compare any pair of them, it is necessary to resort to MAM.

There is a wide variety of MAM to implement indexing structures and query operations in metric spaces. Some use clustering algorithms, like *k*-means [16], to recursively partition the whole set of elements, creating a hierarchical structure (tree) of nested clusters. The creation of this tree of clusters usually follows a top-down approach. Furthermore, in all cases, using the *k*-means algorithm for index building involves estimating a restricted number of centroids and grouping nested subsets of elements around them at each level, following the standard procedure.

As in the case of SAM, several surveys [24] and monographs [2,3] provide thorough comparisons among alternative MAM and their properties. [25] introduces general properties of these indexing structures, along with an initial taxonomy to classify MAM according to two possible data partition strategies:

- *Ball partitioning*: These indexing methods select a subset of featured elements (sometimes referred to as *vantage points*) $\{v_i\} \in \mathscr{S},\ i = 1, \ldots, n$ and define a ball of radius r around each of them, with $r = \text{median}(d(v_i, a)),\ \forall a \in \mathscr{S}$. As a result, each ball defines a data partition separating all points within the scope of the ball from all other points outside the ball. For a dataset distributed uniformly over the metric space, this method creates many partitions intersected by the query region, providing poor performance in many similarity search problems [25,26].
- *Generalized hyperplane partitioning*: This class relies on a data partitioning scheme that defines a set of generalized hyperplanes (GH). Given two elements $v_1, v_2 \in \mathscr{S} \mid v_1 \neq v_2$, a GH is defined as the subset of elements $\{q_i\} \in \mathscr{S} \mid d(q_i, v_1) = d(q_i, v_2),\ \forall i$. In consequence, each GH partitions the space into two regions, one for all elements closer to v_1 and the rest for elements closer to v_2. This strategy can produce more balanced partitioning schemes with GH calculated using randomly sampled elements.

One of the most popular algorithms for metric space indexing, the M-Tree [27], is a prominent example of a ball partitioning algorithm (although it partially incorporates some aspects of GH methods). In contrast, the generalized hyperplane tree (GHT) [25] and its extension into an *m*-ary tree, the GNAT [26], are two featured examples of GH-based indexing methods.

Complementing the previous taxonomy, [1] presents a coherent framework to analyze data indexing methods in metric spaces, mainly from the point of view of multimedia databases and information retrieval systems, and introduces an alternative classification of MAM with two groups:

- *Pivoting algorithms*: They identify a subset of reference elements $\{v_i\} \in \mathscr{S},\ i = 1, \ldots, n$ (known as *pivots*), then classify all remaining objects according to their distances from the pivot objects. Clearly, ball partitioning methods fall in this category, as do other MAM that precompute distance matrices between pairs of elements in the dataset.
- *Compact partitioning algorithms*: These methods partition the metric space into clusters, according to the proximity of elements to each cluster centroid. They guarantee that each element is associated with its closest cluster centre, whereas in pivoting algorithms that may not be the case for elements associated with certain pivots. Methods based on the GH strategy, like GHT and GNAT, pertain to this category, along with algorithms defining a tree of nested clusters.

Uhlmann introduces in [25] the *metric tree*, whose design is further extended in the VP-Tree [28], introducing the alternative term *vantage point* to name the pivots. In contrast, the hierarchical *k*-means tree [29] is one of the first examples of a compact partitioning algorithm. Most of these indexing methods are designed for static datasets, and they are not well-prepared for frequent insertions and deletions. Recent MAM variants have emerged specifically for dynamic data, such as the DBM-Tree [30], whose index structure adapts to the density of local data regions. As a result, the tree height is higher in denser areas to achieve a trade-off solution.

2.2. Approximate similarity search

The approach proposed in this paper to resolve similarity queries belongs to the class of approximate search methods. Previous research works have introduced multiple examples such kind of algorithms. They usually aim at situations in which exact methods cannot provide a fast answer, such as in high-dimensional problems or using big data, where exact techniques default to a full scan of the entire dataset.

It is possible to find approximate similarity search methods conceived for either vector spaces or metric spaces. [31] introduces a unified framework to study fast similarity search methods that can be either exact or approximate. Besides,

[14] presents a comparative summary of relevant approximate methods, introducing a taxonomy to classify them according to relevant traits: the target space (vector, metric); the strategy to obtain approximate results (changing the representation space, reduced number of comparisons); guarantees provided on results quality (no guarantees, deterministic, probabilistic); and interaction degree with users (static, interactive).

One of the first algorithms for approximate similarity search in metric spaces is FastMap [32]. It is based on a method to obtain a fast projection of elements in the original dataset into a k-dimensional space, where k is a user-defined parameter, and the projection preserves the distance between pairs of elements. Thus, it can reach an approximate answer via dimensionality reduction.

Certain approximate algorithms combine different strategies. An interesting example is the Integrated Progressive Search [33], which aims at vector spaces. This method first applies dimensionality reduction methods and, later, uses conventional k-means clustering on the resulting data points in a lower dimension space. This approach reduces the number of distance comparison calculations. However, in general, it is not straightforward to find a dimensionality reduction technique that works well for any problem. As we will see later, MASK takes a different approach, taking advantage of some properties of the core k-means algorithm [34] to reach an approximate result without the need to reduce the number of dimensions of the original feature space. In contrast, the DAHC-Tree [35] is an approximate indexing method aimed at high-dimensional problems in metric spaces. It is easy to construct and valid for static and dynamic datasets. Therefore, it offers a versatile compromise solution for similarity search.

2.3. Scalable similarity search

Several indexing methods for similarity search have been designed to work with distributed systems [24], handling large and complex datasets. An early example is the Metric Chord (M-Chord) [36]. Later, implementations in software products began to emerge. MD-HBase [37] integrates multidimensional indexing into the open-source key-value store HBase, focusing on location-based services.Another example is D-Cache [38], which caches distance information that distributed systems can leverage to speed up similarity queries.

Nowadays, distributed systems involving big data and high-dimensional, high-sparsity problems have proliferated. Technological frameworks like Apache Spark have been crucial for the widespread adoption of cluster-based systems in many areas, including machine learning. Recent advances show promising implementations of SAM on Apache Spark, specifically for similarity search with spatial data and IoT applications [39]. However there is no straightforward implementation of more general MAM in modern distributed data processing frameworks to date. As described in Section 3, the approximate similarity search method proposed in this paper can be naturally implemented on distributed computing platforms. The multilevel, bottom-up approach to index building can be independently executed in each data partition on different nodes. Furthermore, the index construction does not imply any exchange of information or metadata between nodes storing different data partitions. The only coordination requirement would be to maintain the top-level layer of representative points to help decide which nodes should be involved in resolving a particular query.

2.4. Machine learning and similarity search

As we have seen, there is a close connection between machine learning and similarity search indexing. The foundations for designing data access methods rely on familiar concepts in machine learning classification, such as dissimilarity spaces and distance functions. They are also affected by the same limitations, namely the curse of dimensionality and the challenge of identifying the actual intrinsic dimensionality governing some problems.

Clustering algorithms play a central role in many indexing methods for similarity search. The k-means algorithm [15] stands out as a recurring solution for building indexing structures, starting with hierarchical k-means [29]. In fact, according to [2], the GHT and GNAT indexing methods can be regarded as special cases of a broader class of hierarchical clustering methods. However, the pivots selected by GNAT are not necessarily k-means centroids, which are k points that may not belong to $\mathscr{S}$ and minimise the sum of squared distances of individual objects to their closest centroid.

Interestingly, the applicability of k-means for constructing hierarchical indexing structures is already described, albeit without implementation details, by [15]. This application entails building tree clusters in a top-down fashion so that the within-cluster variance does not exceed an upper threshold R. The set of centroids at each level acts as a fair representation of data objects, effectively summarising the clusters of all lower levels. Likewise, MacQueen also demonstrates that k-means can be readily extended beyond vector spaces to the general case of metric spaces. A recent application of k-means and Voronoi diagrams for multidimensional data indexing and spatial data query processing in sensor networks [40] confirms the validity of multilevel k-means indexing for contemporary distributed data problems.

Nevertheless, this approach for building hierarchical indexes with k-means clusters establishes an upper limit R for within-cluster variance. In fact, this is an alternative formulation of the typical problem in partitioning clustering, namely selecting the appropriate number of clusters that must be identified in the dataset. Different methods have arisen to solve this problem (see, for instance, [41] for a detailed discussion), some of which rely on fixing a within-cluster variability threshold.

To avoid these practical issues, it would be desirable to let the multilevel clustering algorithm adapt itself to the density of objects in different regions of the metric space to be indexed. In a certain way, this resembles the idea behind the DBM-Tree

discussed in Section 2.1, but considering a tree of clusters. Our indexing algorithm introduces an unconventional application of *k*-means because it eliminates the restriction of determining the optimal number of clusters in advance. Instead, we propose to create as many centroids at each level as our computational resources can afford.

As shown in Section 3, we can take advantage of minimising the sum of squared distances from each object to the closest centroid to adjust the prototypes position for each cluster according to the density of elements in different metric space regions. In a certain way, this approach also follows the recent strategy of *learned indexes* in databases [42]. This approach states that multiple levels of machine learning algorithms can substitute traditional indexing structures to approximate the underlying data distribution. The Z-order Model (ZM) index is an example application of the learned index strategy to spatial index structures (for instance, the R-tree). In our method, we let the *k*-means algorithm determine good locations to place the centroids representing each cluster, assuming that a sufficiently large number of centroids map the set $\mathscr{S}$.

3. Multilevel k-means structure for data indexing

This section describes the MASK method for approximate similarity search in metric spaces using a multilevel index structure. It also explains the rationale behind the unconventional application of the *k*-means clustering algorithm. This method can be implemented in distributed systems, independently creating sections of the multilevel index that map data partitions stored on different nodes.

3.1. Multilevel index design

Fig. 1 represents the design principles of MASK. At the lowest level, we have the data points in $\mathscr{X}$ to be indexed. Blue boxes in the diagram represent different data partitions or groups. These groups could be stored on different cluster nodes, although in this case, data partitions reside on the same node for simplicity.

As described in Section 2 above, several MAM rely on building a hierarchy of clusters using unsupervised machine learning algorithms like *k*-means. In MASK, the multilevel structure of *k*-means centroids is created following a bottom-up approach. Algorithm 1 details the construction of this multilevel index structure using *k*-means. The first level of centroids (identified as n_1 in Fig. 1) summarises the data points at the lowest level so that each *k*-means prototype (centroid) represents all points assigned to its cluster. The second level (identified as n_2) corresponds to centroids representing the prototypes calculated at the first level, n_1. The same procedure is repeated, creating additional layers of prototypes summarizing the immediately lower group of centroids. The method stops when the number of centroids at the top layer of this multilevel index structure becomes manageable.

Algorithm 1: Multilevel index construction

Algorithm 1: Multilevel index construction

Input: data, lengthGroup, nCentroids
Output: layerPoints, layerLabels, layers

```
ngroups ← lengthData/lengthGroup;
vector ← split(data, ngroups);
Initialize lists layerPoints and layerLabels, and variable idLayer;
while ngroups ≥ 1 do
    // Initialize groupsPoints, groupsLabels and points
    for idGroup = 1 to ngroups do
        points ← vector[idGroup];
        groupsPoints.add(kmeans(nCentroids, points));
        groupsLabels.add(kmeans.labels);
    layerPoints.add(groupsPoints);
    layerLabels.add(groupsLabels);
    ngroups ← length(groupsPoints)/lengthGroup;
    if ngroups ≥ 1 then
        vector ← split(layerPoints, ngroups);
    idLayer += 1;
layers ← idLayer − 1
```

The assembly process requires two initial parameters: the group length (*lengthGroup*) and the number of centroids calculated for each group (*nCentroids*). The group length refers to the number of elements in each data partition at the lowest level. In general, we assume that all partitions will have the same size, although this condition is not strictly necessary. Besides, the size of partitions should depend on available computing power and the number of nodes in distributed computing systems.

Fig. 1 shows a simple example: 80 points in the dataset at the lowest level, with *lengthGroup* = 10 and *nCentroids* = 5. Thus, the algorithm assumes 8 groups of 10 points at the first level. In each group, the algorithm builds 5 clusters using

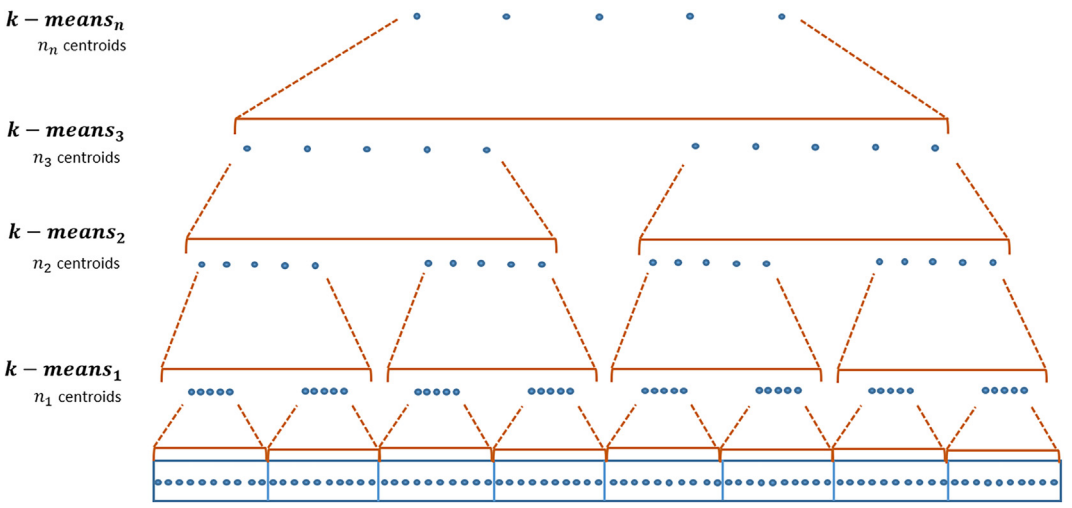


Fig. 1. Conceptual representation of the multilevel structure approach followed in MASK.

k-means. Depending on their distance to the corresponding centroid, a different number of points may fall in each cluster. For this reason, regions with higher data density will also tend to receive more centroids. This clustering at the first level reduces the initial number of data points by a particular proportion, the *data summarization ratio*. The relationship between the values of *lengthGroup* and *nCentroids* determines this ratio. In this example, this ratio is 2:1. Hence, the size of each data group is shrunk by half, from 80 to 40.

At the second level, the algorithm takes 4 groups of 10 points (since $lengthGroup = 10$), and *k*-means is applied to each group to obtain 5 new clusters, each represented by a centroid. This procedure is repeated recursively until the number of points at the top level is small enough. The stop condition implemented in this case is that the total number of centroids at the top level must be less than or equal to *lengthGroup*. Therefore, the problem's computational complexity decreases as the total number of points drops after each iteration.

Nevertheless, the critical aspect of determining the optimal number of centroids at each level remains open. In general, previous indexes based on trees of clusters generate a limited number of centroids at each level. In contrast, recent methods for metric spaces, such as DBM-Tree [30], adapt the height of the indexing tree to changes in the data density of different regions in $\mathscr{X}$. However, the downside of the latter approach is that we may not attain consistent search performance because the algorithm must traverse more or fewer levels depending on the density of the target region for each query. The core novelty in MASK to overcome this issue is an unconventional application of the *k*-means algorithm, tailored to the specific case of information indexing on modern computing platforms, that departs from the traditional strategy suggested in clustering problems.

3.2. A new approach for information indexing using k-means

A typical usage of *k*-means in unsupervised machine learning must find the appropriate number of clusters for a given problem beforehand, where a centroid represents each cluster. Therefore, the number of centroids should not be too high since they would not summarise the underlying data effectively or too low, which would group unrelated data points. In practice, different techniques allow determining the optimal number of centroids, such as the *elbow* method, through a plot of the total within-cluster sum of squares for different *k*, the Silhouette Coefficient Algorithm or the Gap statistic. Nevertheless, the final result can be pretty sensitive to the choice of initial locations of cluster centroids. To circumvent this limitation, algorithms such as *k*-means++ [34] propose spreading the initial random centroids more evenly, generally leading to better results.

However, the goal pursued by MASK is approximate data indexing, not clustering. As more powerful computational infrastructures become available, providing larger memory and storage capacity, prior restrictions about the number of clusters to maintain at each level become less relevant. It is interesting to check what happens in this new scenario when the dataset is *bombarded* with a high number of *k*-means centroids, as large as the computing infrastructure can afford.

A conceptual experiment can be helpful to illustrate this approach on a dataset comprising 4 clouds generated from a t-distribution in $\mathbb{R}^2$, with 12 degrees of freedom and different means for each cloud so that they clearly separate from each other. The experiment consists of two tests:

- The first test generates an increasing number of *k*-means centroids (from 4 to 128) on the complete dataset, following a top-down approach. That is, *k*-means receives the complete set of points to calculate the centroids' positions in each case.

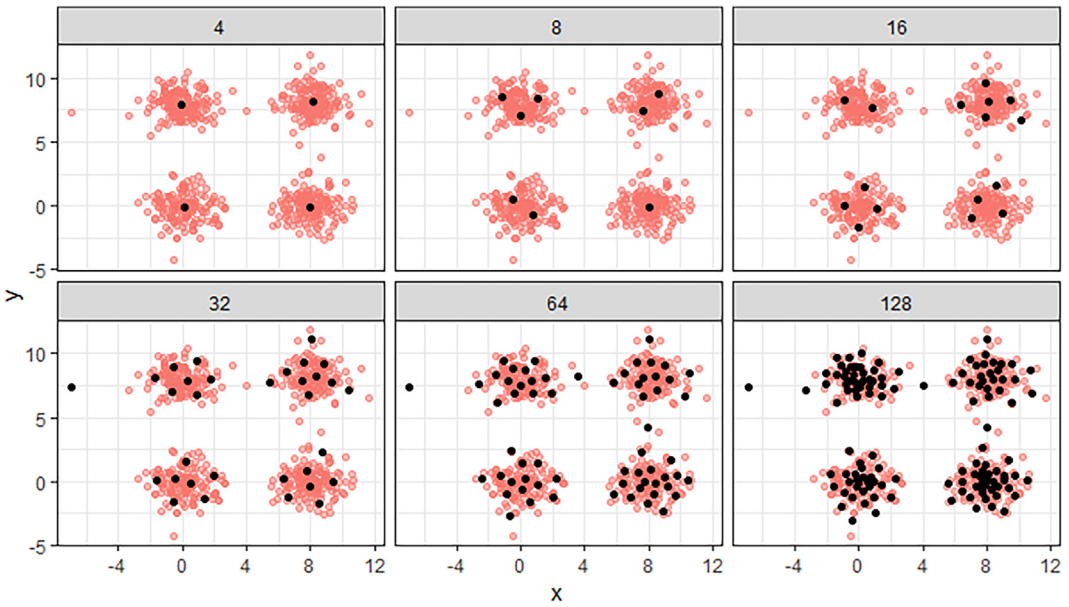


Fig. 2. Results of bombarding 4 data clouds with an increasing number of *k*-means centroids, following a top-down approach. Labels on each pane indicate the total number of centroids generated for each case.

- The second test randomly assigns data points to 4 different data partitions (groups), simulating the situation MASK would find in a distributed system. Then, a growing number of *k*-means centroids independently bombard each group so that the aggregated number of centroids in all groups is the same as in the first test. This procedure represents a bottom-up indexing strategy.

The experiment aims to demonstrate that it is possible to accomplish comparable results regardless of the data indexing approach. Fig. 2 shows the result of the first test. All panels depict the same 4 t-distribution clouds bombarded with increasingly more *k*-means centroids (in black). Interestingly, as the number of centroids increases, more of them tend to concentrate in denser data regions. This behaviour is in line with known results on the consistency of the k-means method [43]. Moreover, when using a vast number of centroids, some even gravitate towards outliers, providing effective coverage for extreme data points.

Fig. 3 represents the bottom-up indexing procedure in a single case of the second test (8 centroids per group). As explained above, each group may have points that belong to any of the 4 initial clouds. Partial outcomes from each group shown in Fig. 3a are combined to create the final result exhibited in Fig. 3b.

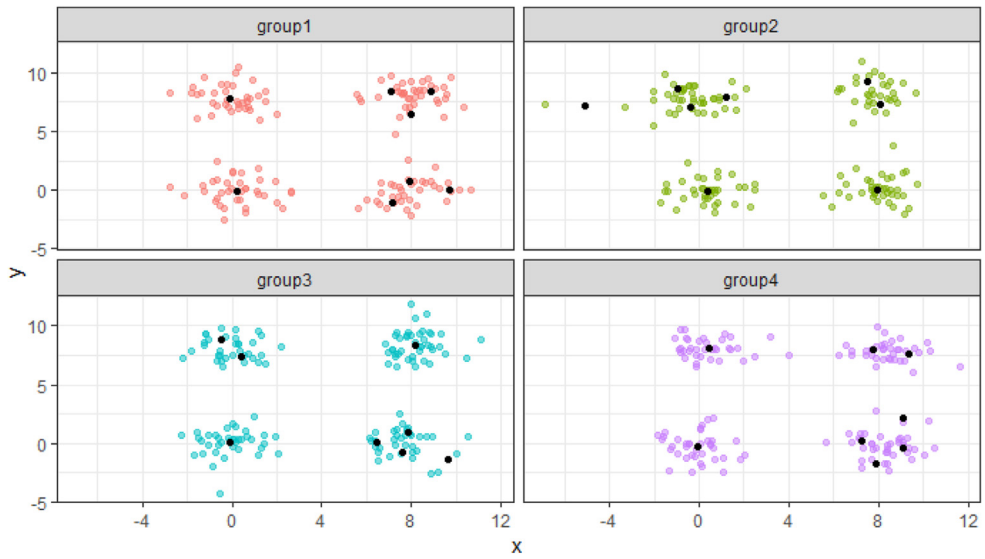
Fig. 4 presents the results for all cases in the second test. In each case, the *k*-means algorithm is run independently on each group, using a growing value of *nCentroids*, from 1 to 32. The panes in Fig. 4 depict the aggregation of partial results from groups. Pane labels indicate the values of *nCentroids*, chosen to ensure that the total number of centroids in each case is the same as in the first experiment. Outcomes from this second test are comparable to those presented in Fig. 2, provided that the indexing structure generates a high enough number of centroids. These results demonstrate that different nodes can store separate data partitions, and then each node can create independent indexing structures. Hence, MASK can scale up to handle problems involving big data distributed in parallel computing clusters.

3.3. Searching and data insertion

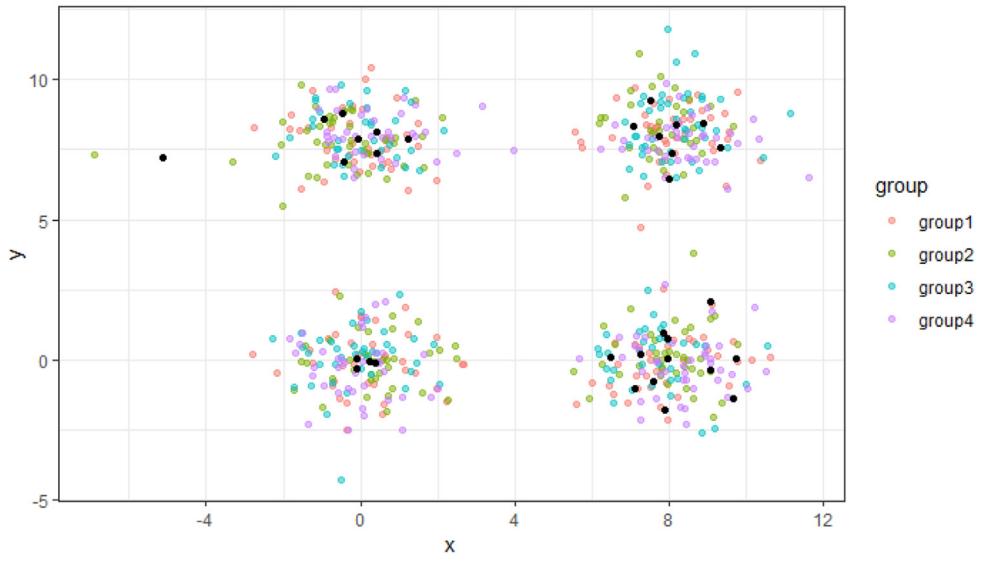
The index based on the multilevel structure with *k*-means centroids in MASK can accelerate similarity search queries. The centroids hierarchy can help discard sections of the dataset where it is unlikely to find candidate results. However, the *k*-means algorithm usually provides a local optimum solution and cannot guarantee finding the global optimum for the centroids location. Consequently, we cannot assure that the search process using our multilevel index returns exact results.

For this reason, our algorithm belongs to the class of approximate similarity search methods. Despite this apparent limitation, we will show that the time and resources required to build the index are affordable. In addition, depending on the dataset characteristics to index, the accuracy of results can be sufficiently high for many practical applications.

To solve any type of query using this multilevel index, we start at the top level of the centroids hierarchy, steering the search according to the distance between the centroids at each level and the target query object $q \in \mathscr{X}$. Then, we adapt this general approach to solving a specific query.



(a) Partial results of generating 8 centroids (in black) in each group (data partition). Points from the 4 original clouds are randomly assigned to each group.



(b) Aggregation of partial results from (a) in a single plot. Data points are coloured according to their data partition. A total of 32 centroids (in black) are created.

Fig. 3. Indexing in a single case of the second test, using 32 *k*-means centroids (8 centroids per group). Partial results in (a) are combined in a single plot in (b) for comparison with the equivalent case in the first test.

Point query. At the top level, calculate the distance from each centroid to q, and select the centroid with the minimum distance value. Then, at the next level, only consider the subset of centroids represented by the one previously selected at the top level. Calculate the distance from each centroid in this subset to q again and choose the nearest centroid. Repeat the same procedure in subsequent iterations over the remaining levels, focusing on children of the selected centroid at the prior level and taking the closest child to q. Eventually, the algorithm reaches the subset of data points represented

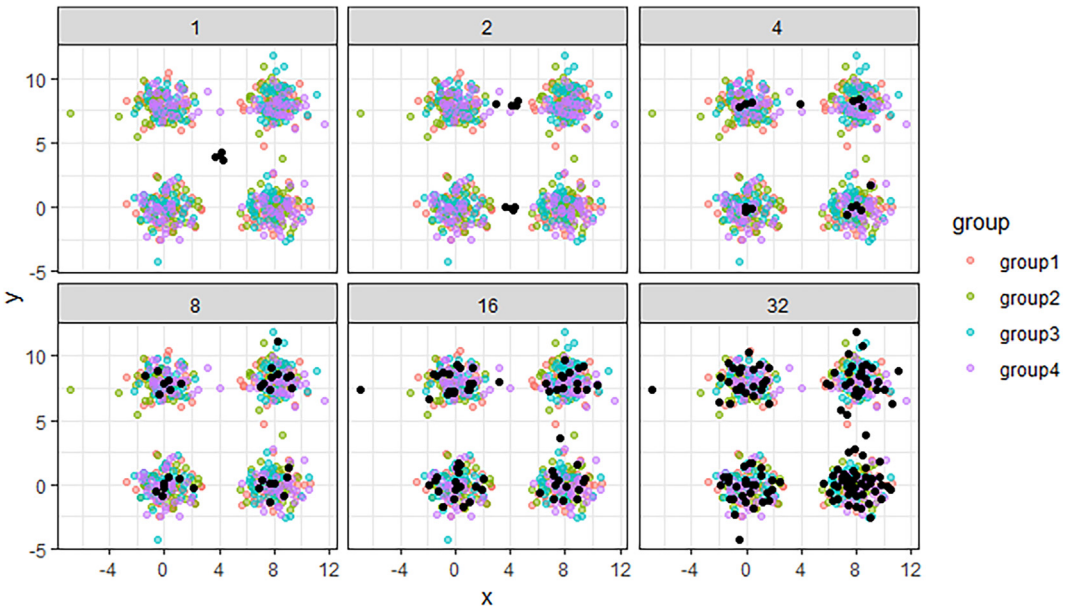


Fig. 4. Results of bombarding the same dataset with an increasing number of *k*-means centroids, following a bottom-up approach with separate data partitions. Points are coloured according to the data partition (group) to which they have been assigned. Labels on each pane indicate the number of centroids generated in each group.

by the selected centroid at the lowest index layer. At this stage, it is easy to perform a full scan over this small subset of data points that returns the one with the lowest distance to q as the point query result. Fig. 5 illustrates this process, marking the centroid selected at each index level with arrows. This procedure is almost equivalent to the nearest-neighbour search described in Algorithm 2 below. The only difference is that the point query algorithm seeks a perfect match within the data partition reached at the end of the search process.

Nearest neighbour and k-nearest neighbours queries. The nearest neighbour search proceeds similarly to resolving the point query. However, the closest element to q in the data partition is returned instead of looking for an exact coincidence. In *k*-nearest neighbours queries, the final subset of data points is ranked according to their distance from q. Then, the query returns the k members with the lowest distances from the rank. Algorithm 2 illustrates the case of the k-nearest neighbour search.

Algorithm 2: *k*-NN search algorithm.

Algorithm 2: *k*-NN search algorithm.

Input: layerPoints, spoint, layers, nCentroids

for $idLayer = layers$ **to** 1 **do**
  if $idLayer \neq layers$ **then**
    $idGroup \leftarrow$ `searchGroup`$(layerLabels, idLayer)$
  else
    $idGroup \leftarrow 0$
  $centroids \leftarrow layerPoints[idLayer][idGroup]$;
  $matrixD \leftarrow$ `eucldeanDistance`$(spoint, centroids)$;
  $posCentroid \leftarrow$ `searchPosMin`$(matrixD)$;
  `// Correction of the centroid identifier`
  if $idLayer \neq layers$ **then**
    $idGroup \leftarrow posCentroid/nCentroids$;
    $idCentroid \leftarrow posCentroid - (idGroup * nCentroids)$;
  else
    $idGroup \leftarrow 0$;
    $idCentroid \leftarrow posCentroid$;
`// Data layer`
$selecPoints \leftarrow layerPoints[idGroup][idCentroid]$;
$matrixD \leftarrow$ `euclideanDistance`$(spoint, selecPoints)$;
$idKPoints \leftarrow$ `searchKNN`$(matrixD, k)$;

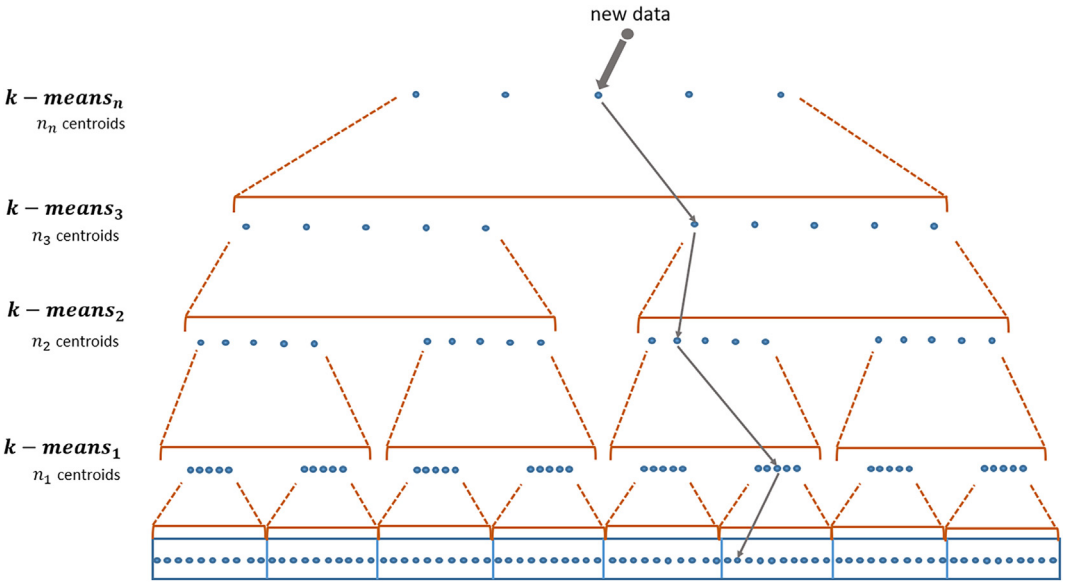


Fig. 5. Procedure to perform a point search using the proposed multilevel index based on *k*-means.

Range query. At each level, consider the subset of children centroids whose distance to q lies within a ball r around q. Then, follow the same search path for every child selected in the lower layer. Finally, we obtain a set of candidate groups of points, and the query returns the point collection that matches the query condition in each candidate set. Algorithm 3 details this approach.

Algorithm 3: Range search algorithm.

Algorithm 3: Range search algorithm.

Input: layerPoints, spoint, layers, nCentroids, radius

```
// Top layer
idGroup ← 0;
centroids ← layerPoints[layers][idGroup];
matrixD ← euclideanDistance(spoint, centroids);
sortVecCentroids ← sortDist(matrixD);
vecCandidates ← selectCandidates(sortVecCentroids, radius);
for idLayer = layers − 1 to 1 do
    // Initialize newCandidates, vecChildren
    for candidate ∈ vecCandidates do
        idGroup ← searchGroup(layerLabels, idLayer);
        centroids ← layerPoints[idLayer][idGroup];
        matrixD ← euclideanDistance(spoint, centroids);
        sortVecCentroids ← sortDist(matrixD);
        vecChildren ←
         selectCandidates(sortVecCentroids, radius);
        // Correction of the centroid identifier
        for pos ∈ vecChildren do
            idGroup ← pos/nCentroids;
            newCandidates ←
             posCentroid − (idGroup * nCentroids);
    vecCandidates ← newCandidates;
// Data layer
matrixD ← euclideanDistance(spoint, centroids);
sortVecCentroids ← sortDist(matrixD);
idRangePoints ← selectCandidates(sortVecCentroids, radius);
```

Besides, MASK can also be applied to dynamic datasets since new points can be stored using the index structure with a simple procedure:

- First, select the k-mean prototype at the top level with the minimum distance to the new point.
- Then, choose the *k*-means centroid again with the minimum distance to the data point from the centroids at the second level, which are children of the selected prototype at the top level.

- Repeat these steps, iterating over the multilevel index layers until we reach one of the data points groups with bounded within-group variance, and assign the new point to that group.

It is pertinent to note that if many new elements are added to $\mathscr{X}$ following this procedure, the size of data partitions at the bottom of the hierarchy may become quite uneven. However, this is not a major problem since new points fall next to other similar elements, guided by the multilayer structure of centroids. As a result, if one partition grows beyond a certain threshold, it can be split into smaller data groups. After this, just the local part of the index covering that specific region needs reconstruction without affecting the rest of the structure.

3.4. Indexing distributed datasets

MASK can be implemented in parallel and distributed computing systems, as shown in Fig. 6. In this example, the dataset consists of 4 element subsets separated from each other in the feature space. Moreover, let us assume that this dataset is split into several partitions, and each partition is stored in a different node. Hence, every partition receives points from any of the 4 original sets. Each node works with its data partitions using MASK in parallel with the rest of the nodes. The algorithm is executed following a bottom-up approach to obtain a local multilevel index structure in each node. Then, at the management level of the distributed system, like the primary cluster node, only the top-level centroids from each node need to be recorded. MASK uses these metadata to decide which nodes will be involved in resolving search queries.

It is possible to search on all nodes in parallel or restrict the inquiry to nodes with centroids whose distance to the target query object is lower than a given threshold ϵ. In the example of Fig. 6, MASK creates 4 centroids at the top level of each node. In addition, it creates 2 groups with 4 centroids per group at the lower indexing level of each node. The algorithm summarises the original dataset effectively using this distributed index structure. As a result, the size of hierarchical metadata maintained by the centralized cluster management service to speed up the similarity search shrinks.

Of course, each particular application can tune several design parameters to adapt the multilevel index construction procedure to the peculiarities of any specific dataset:

- At the lowest index level, it is advisable to bombard data partitions with as many k-means centroids as can be reasonably afforded by available computational resources. As explained in Fig. 4, the higher the number of centroids calculated on the data points, the better the accuracy of MASK to find the actual position of points, effectively mapping high-density regions, sparse regions or outliers.

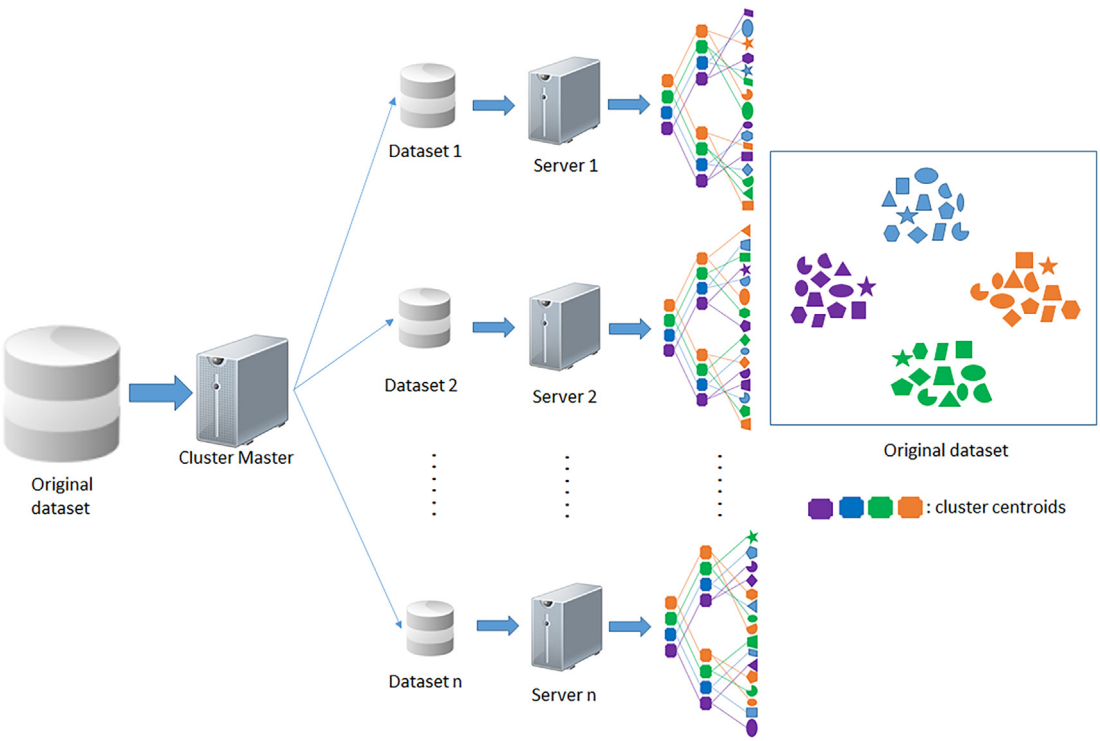


Fig. 6. Schematic application of the MASK indexing method on a distributed dataset.

- At higher index levels, there must be a compromise between indexing accuracy and data reduction. The data summarization ratio of centroids between one layer and its adjacent level above determines this trade-off. For the experiments described in Section 4, we have fixed a value of 2:1 for such a ratio in adjacent layers, effectively halving the number of centroids to calculate in each new iteration. However, further analyses must be conducted to evaluate the impact of this configuration parameter on the MASK performance.

3.5. Complexity analysis

To analyse the complexity of MASK, we consider index construction time to create the multilevel structure and approximate nearest neighbour search time.

Regarding the *index construction* complexity, it is well-known that the optimisation problem involved in the k-means algorithm is NP-hard [44]. In practice, truncated versions of this algorithm are applied so that a rough worst-case bound can be assumed to be $O(I * k * n * p)$ [8], where I is the maximum number of iterations, k is the selected number of centroids, and p is the number of dimensions of the problem space. Thus, since I is fixed, for large enough values of k, the complexity of building each level in the MASK index is $O(knp)$. Let us assume the creation of a balanced multilevel index, easily attained by selecting appropriate values for the *lengthGroup* and *nCentroids* configuration parameters. In that case, the number of levels in the index structure, denoted as L, will be $L = \log n / \log k$ (of course, for $k > 0$). Thus, the total cost of building the MASK index is $O(knp(\log n / \log k))$.

As for approximate nearest neighbours search time complexity, MASK must traverse $\log n / \log k$ levels, plus one additional search in a leaf node (a group of actual data points). MASK examines at most k centroids at each index level to choose the one with the minimum distance to the query point, an operation whose cost is $O(kp)$. If we choose the value of *lengthGroup* to be a multiple of k, then the final search cost within the group of points at the bottom of the structure is also $O(kp)$. As a result, the global cost for the approximate nearest neighbours search in MASK is $O(kp(1 + (\log n / \log k)))$. These results for index construction and nearest neighbour search are comparable to those of previous tree-based algorithms using k-means [33,5].

However, in contrast with previous approaches, one of the main advantages of our proposed algorithm is the bottom-up design for index construction. This strategy lets us directly parallelise index building and similarity searches, distributing sections of the multilevel structure on different nodes. As the algorithm does not require any synchronisation between nodes for these two operations, increasing the number of available node reduces execution time drastically. In turn, this also allows increasing the value of k substantially, improving approximate similarity search precision even with very large datasets.

4. Experimental results

Several experiments have been conducted to evaluate the performance of MASK using three different datasets:

- The first experiment employs a synthetic dataset, including 8 Gaussian clouds, generated using the standard procedure described in [45]. In this case, the main goal is to illustrate the behaviour of MASK against a dataset that exhibits well-known theoretical properties.
- We use the Reuters-21578 dataset [46] for the second experiment, a popular benchmark for high-dimensional and high-sparsity text classification obtained from the UCI repository. Here, we aim to evaluate the capacity of MASK to map and retrieve elements in an adverse scenario for many other alternative algorithms.
- The third experiment involves the MNIST_784 dataset [1], a standard benchmark in similarity search research. We compare MASK performance in approximate nearest neighbours search with respect to existing algorithms for this purpose.

All experiments have been executed on a server equipped with 2 AMD EPYC 7451 microprocessors (24 cores/48 threads, 2.30 GHz, 64 MB L3 cache), 128 GB of DDR4 RAM and an SSD Intel D3-S4510 (capacity 480 GB) for secondary storage. The following section describes the methodology developed to undertake these empirical tests, including metrics to assess MASK performance. After this, we present the experimental results.

4.1. Performance evaluation

The main advantage of MASK is providing a rapid answer to queries at the expense of returning approximate results. Thus, a straightforward strategy to assess the performance of this algorithm is to undertake an exhaustive search of all elements in a given dataset $\mathscr{X}$. Then, the proportion of data points that were correctly retrieved can be determined. This is the common approach to all evaluation experiments developed in this study.

Besides, MASK performance could be improved even more by relocating points not correctly identified after the first build of the multilevel structure of k-means centroids. Intuitively, when the point search reaches a partition at the tree bottom, and the target element is not found in that group, if the target point is reassigned to that partition, it will join other close elements according to the distance function. Then, similar points are placed together after rebuilding the multilevel index,

[1] http://yann.lecun.com/exdb/mnist/

facilitating that the point search retrieves more elements correctly than with the previous index version. Our experiments perform several iterations of data relocation and index rebuilding to check whether this strategy can help or not decrease the error rate in some cases.

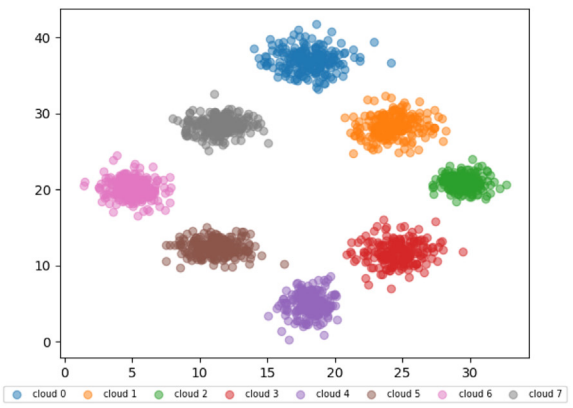
Table 1 summarizes the main characteristics of the datasets used for performance evaluation experiments in this study.

The first experiment evaluates the capacity of MASK to provide good approximations for similarity search, using a synthetic dataset with 8 Gaussian clouds in $\mathbb{R}^2$. This dataset is a common benchmark in many previous studies on data indexing since it is well-characterised from a theoretical point of view [45,8]. It is possible to adjust the mean and standard deviation of each Gaussian distribution to control the percentage of overlap between adjacent clouds. As a result, we created 3 different versions of this dataset:

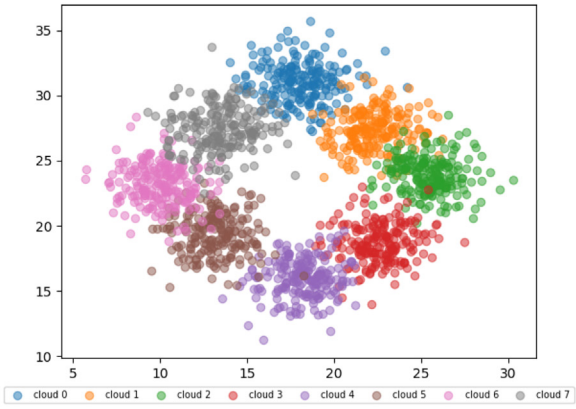
- In the first version, the Gaussian clouds have a clear separation, as depicted in Fig. 7a. This is a baseline test, where most data points should be correctly retrieved. This dataset is identified as $\mathscr{X}_{GNO}$ in experiments.

Table 1
Notation and description of datasets used to evaluate the performance of MASK.

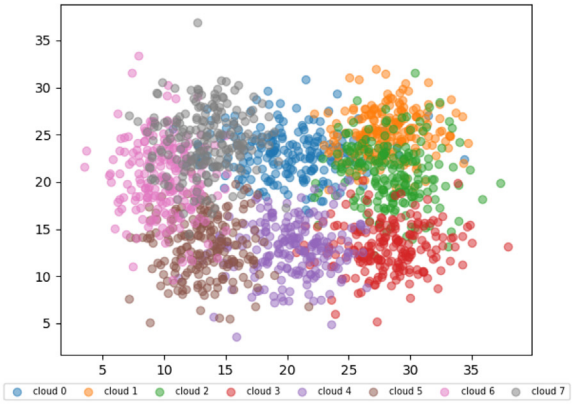
Id	Name	n	d	Sparsity
$\mathscr{X}_{GNO}$	Gaussian clouds (no overlap)	800,000	2	Low
$\mathscr{X}_{GMO}$	Gaussian clouds (moderate overlap)	800,000	2	Low
$\mathscr{X}_{GRO}$	Gaussian clouds (remarkable overlap)	800,000	2	Low
$\mathscr{X}_{REUT}$	Reuters-21578 (UCI repository)	21,578	5,532	High
$\mathscr{X}_{MNIST}$	MNIST_784 (OpenML)	70,000	784	High



(a) Dataset $\mathcal{X}_{GNO}$ (no overlap).



(b) Dataset $\mathcal{X}_{GMO}$ (moderate overlap).



(c) Dataset $\mathcal{X}_{GRO}$ (remarkable overlap).

Fig. 7. Synthetic datasets used in the first experiment, comprising 8 Gaussian clouds with 200 points each and different degrees of overlap between adjacent groups.

- The second version represents a compromise benchmark, with a moderate overlap between contiguous Gaussian clouds, as shown in Fig. 7b. Our experiments evaluating the effect of problem size or configuration parameters on MASK performance use this version. This dataset is identified as $\mathscr{X}_{GMO}$ in our evaluation tests.
- The third version, depicted in Fig. 7c, consists of Gaussian clouds with substantial overlap. In this case, the error rate in point search should rise noticeably since it is more difficult for MASK to accurately retrieve data points inside overlapping regions. In experiments, this dataset is referred to as $\mathscr{X}_{GRO}$.

It is expected that MASK should retrieve, without problems, most data points in $\mathscr{X}_{GNO}$ with clearly separated clouds. However, the error rate should also grow as the degree of overlap between contiguous clouds increases in datasets $\mathscr{X}_{GMO}$ and $\mathscr{X}_{GRO}$. In theory, for an exact indexing algorithm, the error rate should tend to the Bayes (irreducible) error of the whole dataset [8]. In this case, the 8 overlapping regions between contiguous clouds determine the classification error [47]. In practice, MASK will incur in certain additional error for an exhaustive point search on top of this threshold.

MASK performance in a real-world scenario is assessed in the second experiment using a high-dimensional and high-sparsity problem. Many competing algorithms for similarity search in metric spaces struggle to deal with datasets of this kind. The Reuters-21578 dataset [46] from the UCI repository represents a compelling case study. It is a widely used benchmark in supervised text classification, with a collection of more than 10,000 news documents published in 1987 and categorised into 90 different topics. Some of these topics are very similar and present substantial overlap among them. Besides the results for topic categorisation, this experiment measures the performance of MASK for indexing purposes. Henceforth, this dataset is identified as $\mathscr{X}_{REUT}$.

Finally, we use the MNIST_784 dataset available online [2] to compare MASK performance with respect to other existing algorithms for approximate nearest neighbours search. This dataset is identified as $\mathscr{X}_{MNIST}$. It contains 70,000 samples of handwritten digits, which have been size-normalised and centred in a fixed-size image, described by 784 features. Besides its high dimensionality, the dataset also exhibits high sparsity. It is considered a classical benchmark in the field of similarity search.

Fig. 8a visualises a subset of three categories ('money-supply', 'coffee' and 'oilseed') from the $\mathscr{X}_{REUT}$ dataset. Fig. 8b presents another visualisation corresponding to the complete $\mathscr{X}_{MNIST}$ dataset. These graphs have been obtained by calculating the first 50 components from a PCA dimensionality reduction [11] for both datasets, then applying the t-SNE tool [48] to visualise high-dimensional data. In both cases, we can see the complexity of these problems that exhibit substantial overlap among data elements from different categories.

4.2. Synthetic dataset results

The first experiment compares MASK performance against the two opposite versions of the synthetic dataset: $\mathscr{X}_{GNO}$ (no overlap, see Fig. 7a) and $\mathscr{X}_{GRO}$ (remarkable overlap, see Fig. 7c). In this execution, 8 Gaussian clouds in $\mathbb{R}^2$ are generated for each dataset, with 200 points per cloud. The distance function to measure dissimilarity between data elements is the Euclidean metric. The initial configuration parameters for MASK are $lengthGroup = 16$ and $nCentroids = 8$, which render a data summarization ratio between adjacent index levels of 2:1. In both cases, MASK divides the initial dataset of 1,600 points into 100 groups, with 16 points per group. All data points are randomly assigned to one of these groups so that each partition (group) may contain points from any of the 8 clouds. This simulates the situation in a distributed dataset, where data partitions are processed separately.

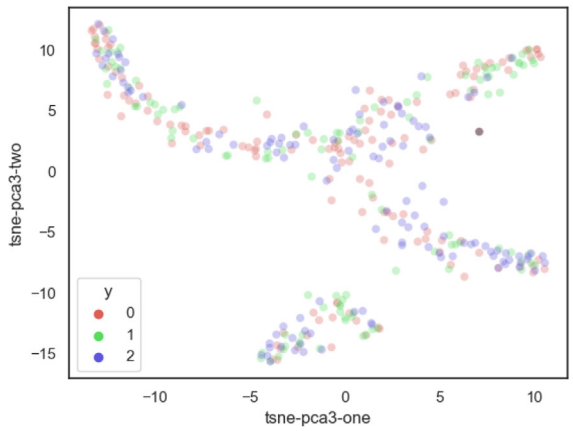
After creating the data groups, we build the multilevel index. At the bottom level, the algorithm obtains 8 centroids per group, calculating 800 centroids in this first layer. The algorithm repeats this procedure at the next level, summarizing the 800 centroids obtained before. Hence, the centroids from the previous level are assigned to a new centroid at the next level above, using k-means and the Euclidean distance.

The algorithm eventually generates a multilevel index of k-means centroids, with a final depth of 8 layers, since the stop condition is that the number of centroids at the top layer must be lower than $lengthGroup$. Now, the index is ready to accept point queries that systematically search for all points in the original dataset, following Algorithm 2 to seek a perfect match. In the case of dataset $\mathscr{X}_{GNO}$, the indexing method correctly finds all points without errors, as the original clouds are well-separated from each other. Due to this, there is no need to perform any iterations to relocate data points since the error rate cannot be further reduced.

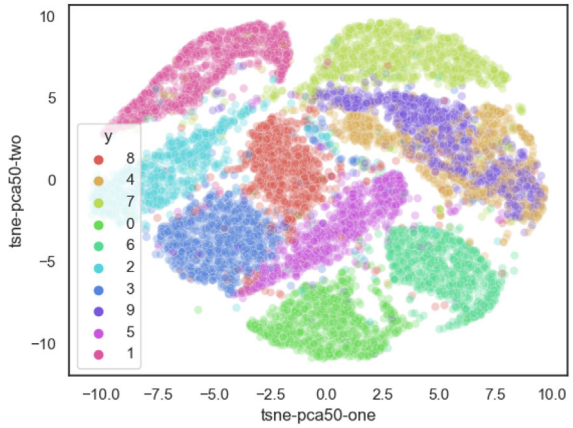
Nonetheless, the error rate increases noticeably working with dataset $\mathscr{X}_{GRO}$. In this case, the multilevel index cannot accurately retrieve elements in proximity or inside overlapping regions, producing an error rate of 40.5% for a point search of all elements in $\mathscr{X}_{GRO}$.

As mentioned above, an option to reduce this error rate is relocating data points not correctly found in the corresponding partition, reached at the end of the point search algorithm. After rebuilding the multilevel index, we repeat the exhaustive point search through all dataset elements to obtain the new error rate. Following this approach, the error rate decreases from 40.5% at iteration 0 to 30.31% at iteration 3. Fig. 9 represents the error rate values corresponding to each iteration to relocate data points. The error fluctuates around the best asymptotic solution due to the iterative optimization method to implement

[2] http://yann.lecun.com/exdb/mnist/



(a) Visualisation of a subset of vectors representing documents for categories 'money-supply', 'coffee' and 'oilseed' in $\mathcal{X}_{REUT}$.



(b) Visualisation for dataset $\mathcal{X}_{MNIST}$.

Fig. 8. Visualisation of datasets $\mathcal{X}_{REUT}$ and $\mathcal{X}_{MNIST}$ using t-SNE.

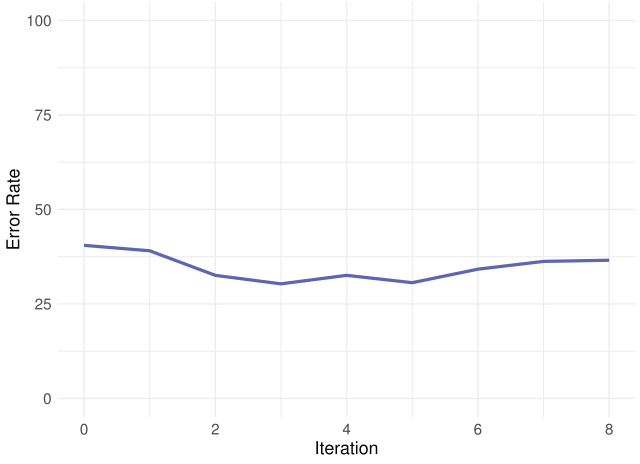


Fig. 9. Error rates for each iteration to relocate data points with dataset $\mathcal{X}_{GRO}$.

k-means. From these results, we can conclude that the algorithm converges to its best result very fast, even if it can still not determine the correct partition for some points on overlapping areas.

4.2.1. Influence of the dataset size

Since MASK is suitable for distributed computing platforms, evaluating how the problem size affects its performance is crucial. The dataset $\mathscr{X}_{GRO}$ leads to a significant error rate caused by the considerable overlap between Gaussian clouds. This high error may conceal the effects of other sources of variation on index performance, such as the size of the dataset or changes in configuration parameters. For this reason, from this point, we compare the performance between datasets $\mathscr{X}_{GNO}$ (see Fig. 8 and $\mathscr{X}_{GMO}$ (see Fig. 7b). Different points per cloud are considered for both datasets (200, 1,000, 10,000 and 100,000) in order to analyze how this variation affects the error rate and computation time of MASK. All simulations have been performed with *lengthGroup* = 16 and *nCentroids* = 8.

In the case of dataset $\mathscr{X}_{GNO}$, Fig. 10a displays the total time consumed during the index construction stage (*tree time*) and the time required to search the whole set of points (*search time*) for each problem size. Table 2 presents the numerical results along with the corresponding error rates. The conclusion is that the error rate remains very low for any dataset size. However, as the problem size grows, the computation time for index construction and exhaustive point search also increases.

We undertake the same analysis with dataset $\mathscr{X}_{GMO}$ (moderate overlap). It is clear, from computation times represented in Fig. 10b and shown in Table 3, that when the problem size grows, the time spent by MASK building the tree and searching for all points also increases. The error rate only exhibits a slight variation, mainly due to the overlap between clouds. As the number of points per cloud grows, the overlapping region expands, thereby increasing the error rate.

Fig. 11 compares the error rate obtained from datasets $\mathscr{X}_{GNO}$ and $\mathscr{X}_{GMO}$. As expected, the error rate is higher for the moderate overlap case but does not rise significantly for larger problem sizes with any of these two datasets. Therefore, the degree of overlap between adjacent clouds has a more relevant impact on the error rate than the size of the problem.

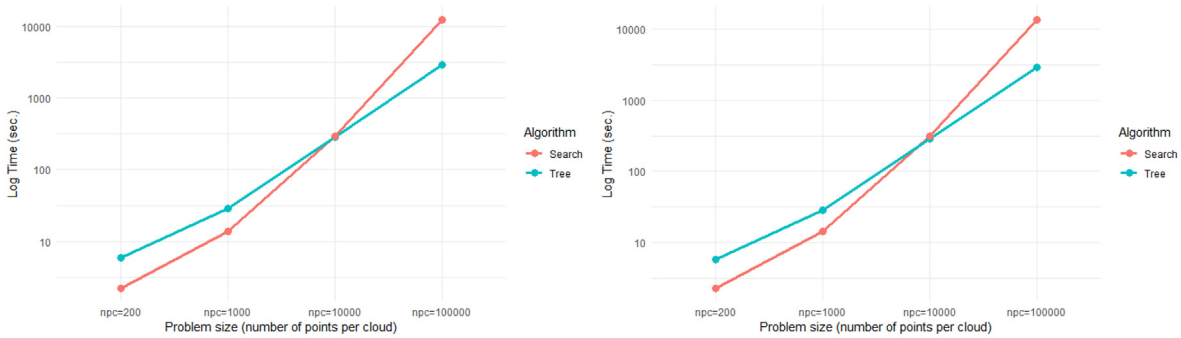


Fig. 10. Comparison of tree time and search time (log scale) for different values of problem size.

Table 2
Index construction time (tree time), exhaustive point search duration (search time) for all elements in $\mathscr{X}_{GNO}$ (no overlap) and error rate, with different dataset sizes. The number of points per cloud in each problem (*npc*) is also indicated.

	Problem size $npc = 200$	Problem size $npc = 1,000$	Problem size $npc = 10,000$	Problem size $npc = 100,000$
Tree time (sec.)	5.905	28.601	287.674	2863.450
Search time (sec.)	2.250	13.892	293.547	12415.074
Error rate(%)	0.0	0.11	0.09	0.05

Table 3
Index construction time (tree time), exhaustive point search duration (search time) for all elements in $\mathscr{X}_{GMO}$ (moderate overlap) and error rate, with different dataset sizes. The number of points per cloud in each problem (*npc*) is also indicated.

	Problem size $npc = 200$	Problem size $npc = 1,000$	Problem size $npc = 10,000$	Problem size $npc = 100,000$
Tree time (sec.)	5.842	28.861	288.439	2874.815
Search time (sec.)	2.284	14.382	313.715	13730.897
Error rate (%)	11.937	14.212	14.087	16.334

4.2.2. Impact of configuration parameters

The evaluation continues analysing the influence of configuration parameters, *lengthGroup* and *nCentroids*, on MASK performance. Again, datasets $\mathscr{X}_{GNO}$ and $\mathscr{X}_{GMO}$ are used in these tests, fixing the number of points per cloud to 10,000 in both cases.

Fig. 12 compares the computation time for the tree-building phase and the exhaustive point search for dataset $\mathscr{X}_{GNO}$, using different values of *lengthGroup* and *nCentroids*. In all cases, the data summarization ratio between these two configuration parameters is kept at 2:1. As *lengthGroup* and *nCentroids* increase their values, the time to build the multilevel index decreases moderately, suggesting an inverse relationship between these magnitudes. At the same time, since the number of indexing levels decreases when the group length and the number of centroids grow, a slight reduction in search time can be achieved in some cases. Table 4 explores these differences in more detail.

A similar situation occurs when MASK is applied to dataset $\mathscr{X}_{GMO}$, using the same series of increasing values for *lengthGroup* and *nCentroids* and keeping the data summarization ratio between them at 2:1 for all cases. Fig. 13 and Table 5 present the results. As *lengthGroup* and *nCentroids* values increase, the computation time required by MASK for tree construction tends to decrease, as does the time needed for an exhaustive point search of the entire dataset. The values in column *tree depth* show that the number of layers in the multilevel index drops from 13 to 10 levels. This is the leading cause of time reduction in tree construction and point search. The total time spent on exhaustive point search drops since every individual point query must traverse fewer layers to return a result.

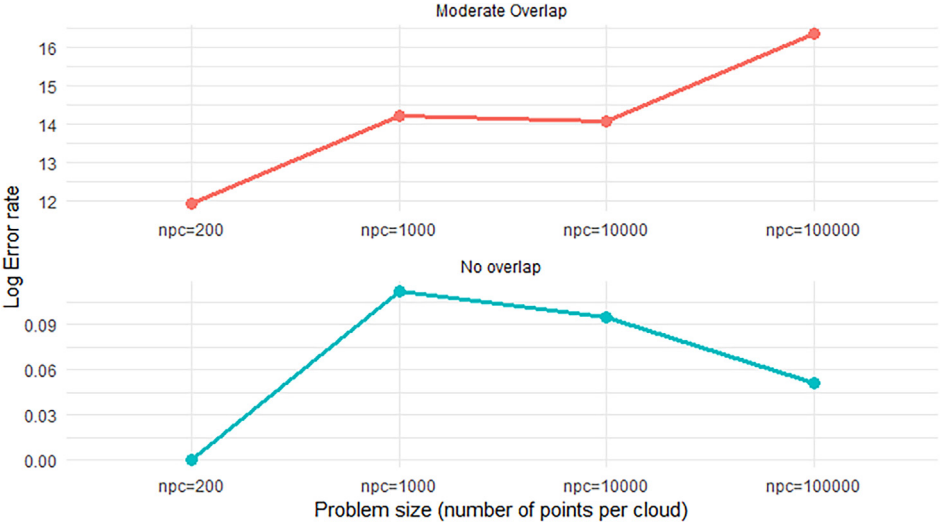


Fig. 11. Error rate comparison when there is no overlap between dataset clouds (bottom panel) and when there is some overlap between them (top panel).

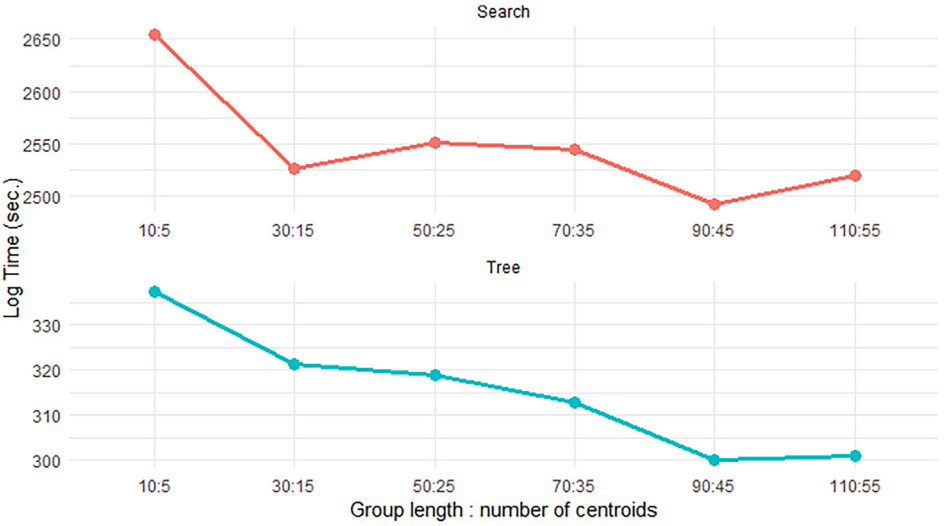


Fig. 12. Computation time spent in multilevel index construction (tree) and exhaustive point query (search), for different values of *lengthGroup* and *nCentroids*, with dataset $\mathscr{X}_{GNO}$ (10,000 points per cloud).

Table 4
Computation time required for multilevel index construction (*tree time*) and exhaustive point query (*search time*), for different values of *lengthGroup* and *nCentroids*, with dataset $\mathscr{X}_{GNO}$ (10,000 points per cloud). The number of levels created by the index structure in each case (*tree depth*) is also indicated.

	Tree time (sec.)	Tree depth	Search time (sec.)	Error rate(%)
lengthGroup = 10 *nCentroids* = 5	337.450	13	2653.897	0.158
lengthGroup = 30 *nCentroids* = 15	321.383	12	2525.883	0.125
lengthGroup = 50 *nCentroids* = 25	318.820	11	2550.783	0.06
lengthGroup = 70 *nCentroids* = 35	312.977	11	2543.880	0.053
lengthGroup = 90 *nCentroids* = 45	300.070	10	2491.706	0.061
lengthGroup = 110 *nCentroids* = 55	301.053	10	2518.820	0.055

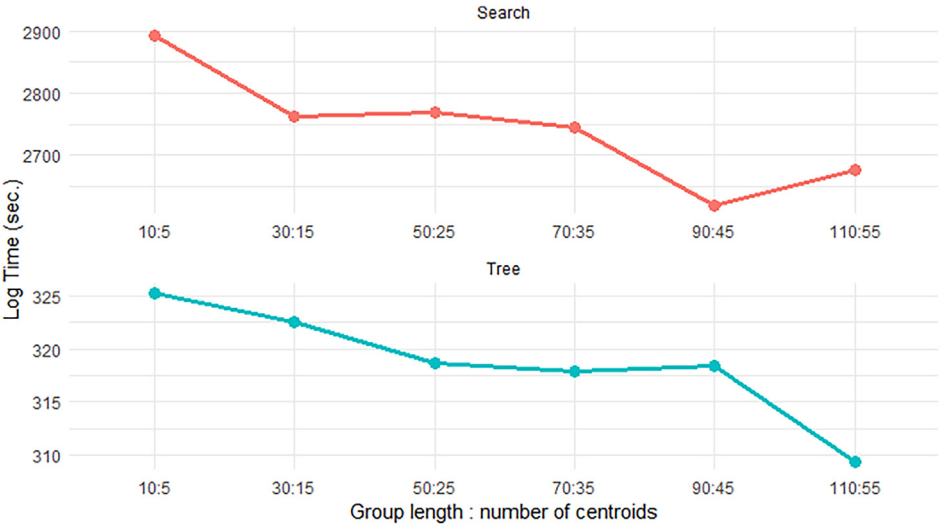


Fig. 13. Computation time spent in multilevel index construction (tree) and exhaustive point query (search), for different values of *lengthGroup* and *nCentroids*, with dataset $\mathscr{X}_{GMO}$ (10,000 points per cloud).

Table 5
Results for the computation time required for multilevel index construction (*tree time*) and exhaustive point query (*search time*), for different values of *lengthGroup* and *nCentroids*, with dataset $\mathscr{X}_{GMO}$ (10,000 points per cloud). The number of levels created by the index structure in each case (*tree depth*) is also indicated.

	Tree time (sec.)	Tree depth	Search time (sec.)	Error rate(%)
lengthGroup = 10 *nCentroids* = 5	325.294	13	2891.359	13.791
lengthGroup = 30 *nCentroids* = 15	322.588	12	2762.296	13.631
lengthGroup = 50 *nCentroids* = 25	318.593	11	2769.591	12.775
lengthGroup = 70 *nCentroids* = 35	317.898	11	2745.472	11.853
lengthGroup = 90 *nCentroids* = 45	318.403	10	2618.803	11.712
lengthGroup = 110 *nCentroids* = 55	309.307	10	2676.827	11.448

The main effect when the values of *lengthGroup* and *nCentroids* grow is the progressive improvement of the error rate in exhaustive point search. Fig. 14 compares the error rate variation for datasets $\mathscr{X}_{GNO}$ (no overlap) and $\mathscr{X}_{GMO}$ (moderate overlap). Even though the error rate values for dataset $\mathscr{X}_{GNO}$ are much lower than for $\mathscr{X}_{GMO}$, as *lengthGroup* and *nCentroids* values rise, we observe similar trends in both graphs. It stems from these results that the influence of both *lengthGroup* and *nCentroids* on MASK performance is consistent, disregarding the degree of overlap between the Gaussian clouds. Index structures with fewer layers need less time to be constructed, leading to a more precise point search with a lower error rate. Therefore, MASK has a performance advantage when configuring larger groups (partitions). Despite this, the error rate levels off after a certain point, and it is not possible to achieve further improvement.

4.2.3. Role of the data summarization ratio

So far, the data summarization ratio between *lengthGroup* and *nCentroids* has been 2:1 in all experiments. Here, we explore how changes in this data summarization ratio affect MASK performance using the $\mathscr{X}_{GNO}$ and $\mathscr{X}_{GMO}$ synthetic datasets again. In both cases, we generate 10,000 points per cloud.

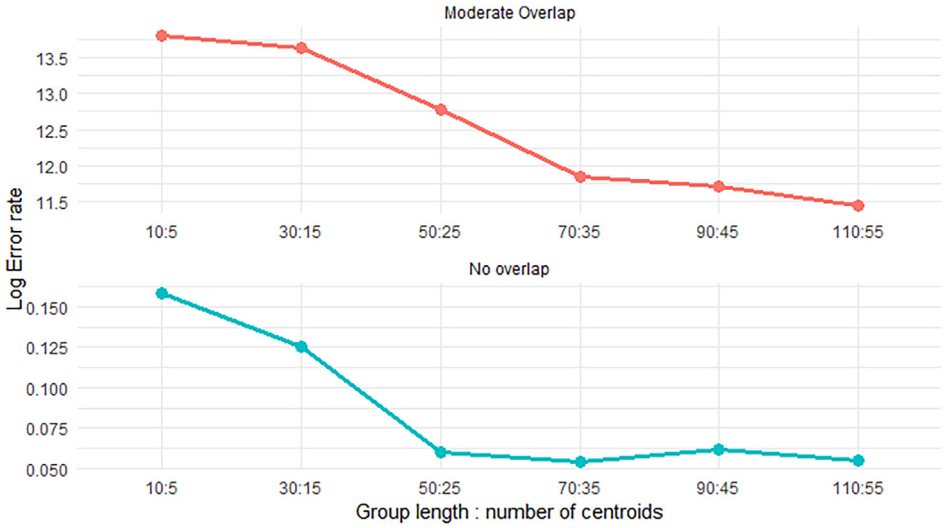


Fig. 14. Comparison of changes in the error rate values with dataset $\mathscr{X}_{GNO}$ (no overlap) and $\mathscr{X}_{GMO}$ (moderate overlap), as *lengthGroup* and *nCentroids* are increased.

Three situations are studied, corresponding to *lengthGroup* $= 10, 50$ and 90, respectively. Fig. 15 presents the results for $\mathscr{X}_{GNO}$ (no overlap case). The left panel displays changes in computation time for tree assembly (blue line) and during the search process (red line) for *lengthGroup* $= 10$. On the horizontal axis, *nCentroids* gets values of $2, 5$ and 8, respectively. The graph shows that computation time for both stages directly depends on the number of centroids per group, as expected. If the number of centroids increases, the computation time rises for index construction and exhaustive point search. The same effect occurs in the other two cases with *lengthGroup* $= 50$ and 90. When *lengthGroup* $= 50$, *nCentroids* is set to $10, 25$ and 40, and when *lengthGroup* $= 90$, the number of centroids is $18, 45$ and 72. In this way, the sequence of data summarization ratios is the same as in the case of *lengthGroup* $= 10$. The difference between computation time during the index construction stage and for the search process remains approximately constant over the three cases of *lengthGroup*.

Fig. 16 displays the computation time for the moderate overlap case (dataset $\mathscr{X}_{GMO}$). The same three scenarios with *lengthGroup* $= 10, 50$ and 90 are considered, with identical variations in the number of centroids for each case. Results are very similar to those in Fig. 15. Therefore, the overlap between adjacent clouds does not affect computation time for tree building or search queries with different data summarization ratios.

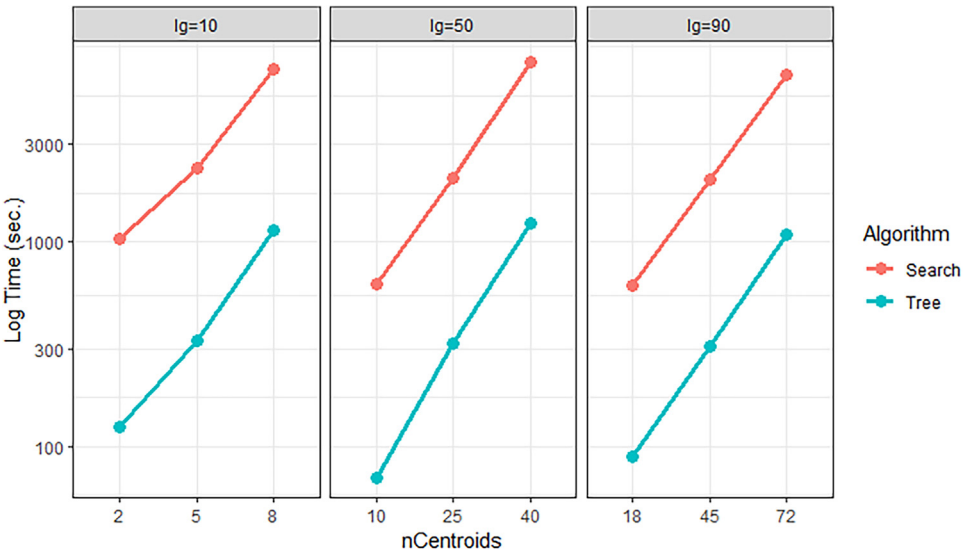


Fig. 15. Computation time during multilevel index building (*tree*) and exhaustive point search (*red*), for different values of the data summarization ratio between *lengthGroup* and *nCentroids*, with dataset $\mathscr{X}_{GNO}$ (10,000 points per cloud). Each panel represents the computation time for a fixed *lengthGroup* (*lg*) and different values of *nCentroids*.

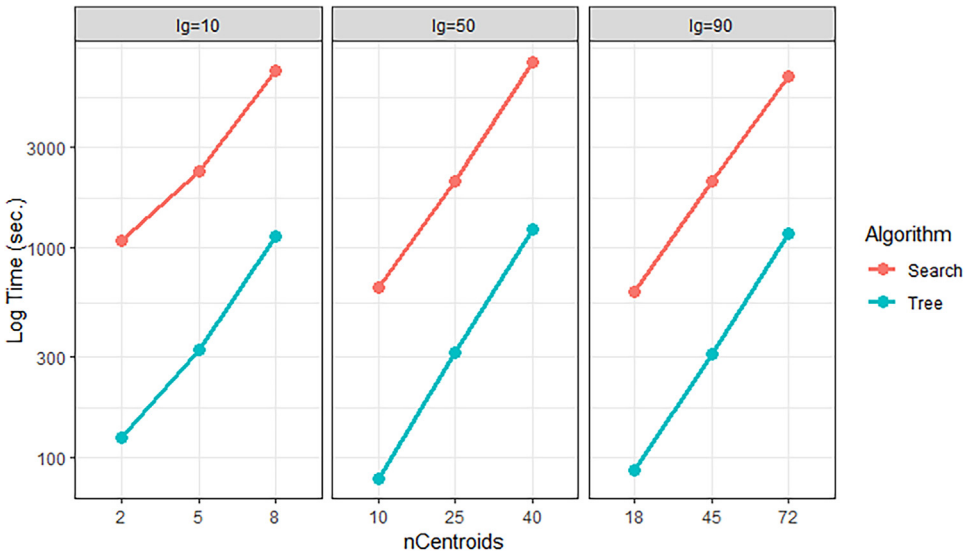


Fig. 16. Computation time for multilevel index building (*tree*) and exhaustive point search (*red*), with different values of the data summarization ratio between *lengthGroup* and *nCentroids*, using dataset $\mathscr{X}_{GMO}$ (10,000 points per cloud). Each panel represents computation time for a fixed *lengthGroup* (*lg*) and different values of *nCentroids*.

Fig. 17 displays changes in the error rate for datasets $\mathscr{X}_{GNO}$ (no overlap) and $\mathscr{X}_{GMO}$ (moderate overlap) in the three previous situations (with *lengthGroup* $= 10, 50$ and 90), respectively. As seen above, the error rate decreases when *lengthGroup* and *nCentroids* increase their values. This difference becomes remarkable in the case of moderate overlap. In spite of this, if we keep the value of *lengthGroup* constant, the error rate drops or remains stable when the number of centroids per group increases. Therefore, a data summarization ratio of 2:1 is a satisfactory compromise solution between computation time (for index construction and search) and the error rate.

4.3. High-dimensional and sparse dataset results

One of the most problematic aspects of MAM is that, in many cases, they render very poor performance in high-dimensional and sparse dissimilarity spaces. As we explained in Section 1, this issue is linked to the intrinsic dimensionality of our dataset, δ, which determines the number of effective dimensions representing all elements. The higher the number of

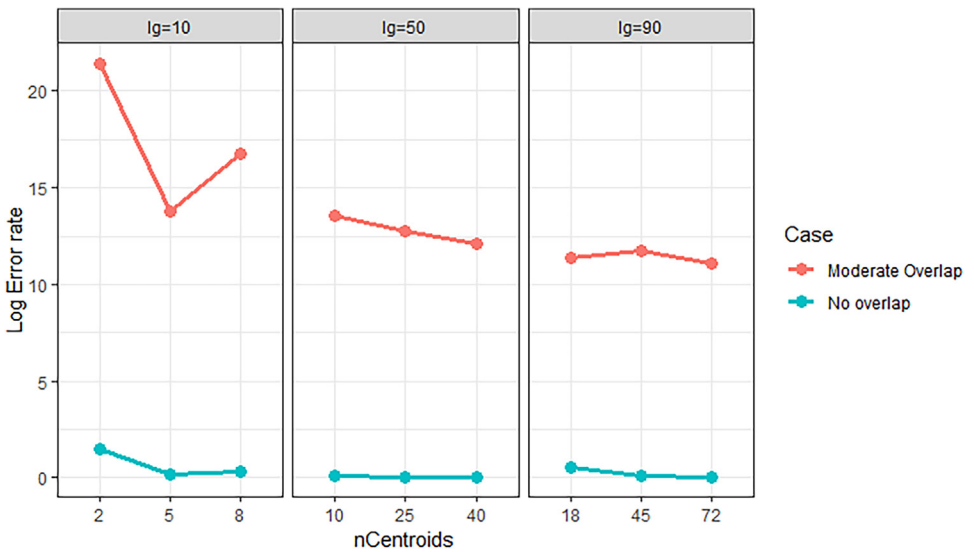


Fig. 17. Comparison of error rates for datasets $\mathscr{X}_{GNO}$, without no overlap between adjacent clouds (blue), and $\mathscr{X}_{GMO}$, with moderate overlap between Gaussian clouds (red).

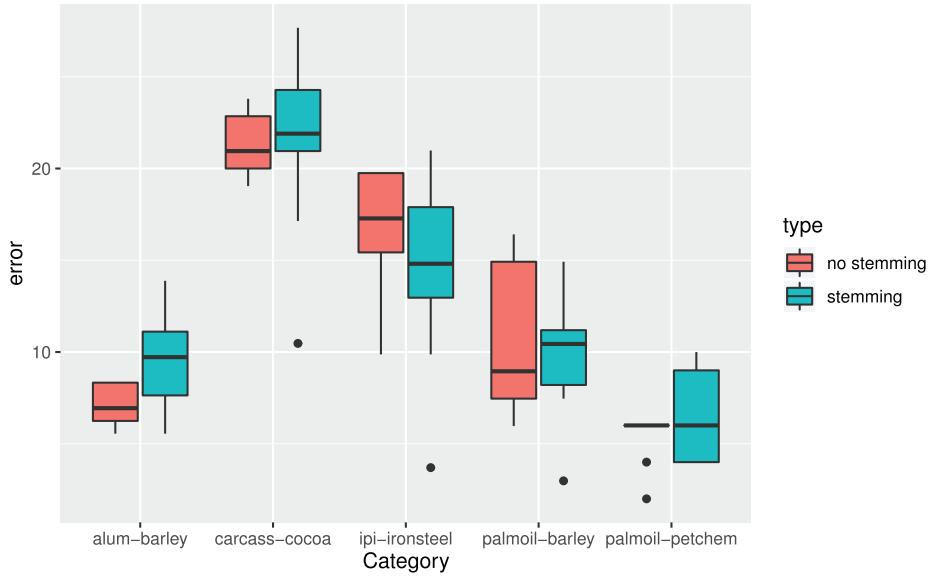


Fig. 18. Boxplots comparing the classification error of documents in each pair of categories, obtained with and without stemming.

dimensions and the sparsity of our dataset, the more difficult it will be for our indexing method to locate query objects effectively.

However, a key advantage of MASK is that if we bombard data partitions with enough k-means centroids, the indexing structure can still retrieve data elements with good accuracy at the expense of returning approximate results. As shown in Section 3.2, when the first layer has a high number of k-means centroids, MASK places them according to the density of the underlying data elements. In this way, we can index not only high-density data regions but also outliers and clusters of elements separated from the main concentrations of points.

Several experiments were conducted to evaluate the performance of MASK in these problems, using dataset $\mathscr{X}_{REUT}$. Individual documents are tagged according to their topic, and there is substantial overlap between different categories. These tests aim to classify documents as belonging to a particular category.

Following conventional text mining procedures, we removed stop words (common words that usually do not add valuable information to the analysis), numbers, punctuation marks and white spaces from documents. After this, the text in each document is converted to lowercase, and the corresponding term-document matrix is created. Thus, an $m \times n$ matrix represents each set of documents, where m is the number of unique terms in the dictionary and n is the number of documents in the data set. Each element w_{ij} of the term-document matrix represents the importance or weight of the term i in document j. We use the TF-IDF measure [4] to calculate w_{ij} through Eq. (1)

$$w_{ij} = tf_{ij} \times \log\left(\frac{n}{df_i}\right), \tag{1}$$

where tf_{ij} denotes the number of occurrences of the term i in document j; n is the total number of documents in the dataset, and df_i represents the number of documents in which term i appears. Given these initial conditions, elements are represented in a high-dimensional dissimilarity space, where each document's components are the TF-IDF measures. This representation space also has high sparsity since many terms will be absent from numerous documents.

In the first round of experiments, documents are encoded using TF-IDF with stemming [4]. Then, results are compared with the second round of experiments, in which the document encoding does not include stemming, to assess the impact of this step on MASK search accuracy. Several pairs of document categories have been considered for the sake of clarity to illustrate the results: 'alum' vs 'barley'; 'ipi vs 'iron-steel'; 'carcass' vs 'cocoa'; 'palm-oil' vs 'pet-chem' and 'palm-oil' vs 'barley'. In all cases, the input parameters for our algorithm will be $lengthGroup = 16$ and $nCentroids = 8$ (thus, the data summarization ratio is 2:1). The algorithm builds the search tree with a maximum depth of 3 levels.

Fig. 18 compares the classification error rate with and without stemming for 10 iterations of the algorithm (relocating data elements in each iteration to improve MASK performance).

Since the maximum classification error rate is less than 25%, this confirms that the results obtained in both cases are very accurate. However, there is a slight improvement in several tests without stemming. Tables 6 and 7 present the classification and indexing error for each pair of categories when stemming is applied and without this step, respectively. Results indicate that, in many cases, the best iteration is either the first or the second attempt to relocate data points, according to the previous position of centroids in the multilevel index. Hence, the actual performance gain attained through this iterative process

Table 6
Classification and indexing error in the best iteration of data relocation in MASK, for each pair of document categories, when stemming is applied. All simulations have been performed with *lengthGroup* = 16 and *nCentroids* = 8.

Categories	Classification		Indexing	
	Best Iter.	Error	Best Iter.	Error
alum/ barley	2	5.55	1	8.33
ipi/ iron-steel	4	3.7	0	12.34
carcass/ cocoa	0	10.47	0	16.19
palm-oil/ pet-chem	0	4.0	0	2.0
palm-oil/ barley	0	2.98	1	17.91

Table 7
Classification and indexing error in the best iteration of data relocation in MASK, when stemming is not applied. All simulations have been performed with *lengthGroup* = 16 and *nCentroids* = 8.

Categories	Classification		Indexing	
	Best Iter.	Error	Best Iter.	Error
alum/ barley	3	5.55	0	6.94
ipi/ iron-steel	5	9.87	3	12.34
carcass/ cocoa	0	19.04	0	15.23
palm-oil/ pet-chem	0	2.0	4	0.0
palm-oil/ barley	1	5.97	0	11.94

is low in many practical situations. Due to this, the approximation provided by the initial construction of the multilevel indexing structure can be quite acceptable for high-dimensional problems.

Finally, we study the algorithm's convergence as the size of groups and the number of centroids to map each group vary. This experiment considers categories 'jobs', 'iron-steel' and 'cotton'. In turn, *lengthGroup* and *nCentroids* are set to values ranging from 8–4 to 60–30 to keep the data summarization ratio in all cases at 2:1. Fig. 19 shows the classification error rate for each case without stemming. The classification error rate decreases when both input parameters increase their values. Results are similar when stemming is not applied. Thus, they suggest that this encoding step does not affect the accuracy of MASK as the size of data partitions and the number of centroids increase. Again, this is consistent with using a high number of centroids to map the underlying dataset, as explained in Section 3.2. As many more centroids map the dataset, their location will become more advantageous for indexing purposes.

4.4. Comparison with existing algorithms

In this section, we evaluate MASK performance regarding approximate nearest neighbour search compared to alternative MAM. Besides MASK, the algorithms included in this study are:

- KD-Tree [21] is an exact similarity search method. Nevertheless, it provides a practical baseline to evaluate the performance of the other approximate algorithms, retrieving the true set of nearest neighbours in each experiment.
- FLANN [5] is an approximate similarity search method widely adopted in many fields, principally image treatment and artificial vision. In this case, we use an implementation named *pyflann* [3], providing Python bindings for the original FLANN library.
- PyNNDescent [4] provides a Python implementation of Nearest Neighbor Descent for k-neighbour-graph construction and approximate nearest neighbour search [49]. It is a recent algorithm generally regarded as offering very high performance in various problems.

For this comparison, we use the $\mathscr{X}_{MNIST}$ dataset, described in Section 4.1 above. It is a standard dataset for nearest neighbour search evaluation, widely adopted in previous research works. Hence, it offers a handy reference to compare the performance of MASK and the other competing algorithms in a high-dimensional problem space with high sparsity. The assessment metric for this comparative study is the *recall* attained in approximate search of an increasing number of nearest neighbours, given a query point.

We proceed as follows to implement our experiments:

1. The complete $\mathscr{X}_{MNIST}$ dataset is divided into a training set of 60,000 elements and a testing set with the remaining 10,000 elements, following a standard procedure.

[3] https://github.com/primetang/pyflann
[4] https://pynndescent.readthedocs.io/en/latest/

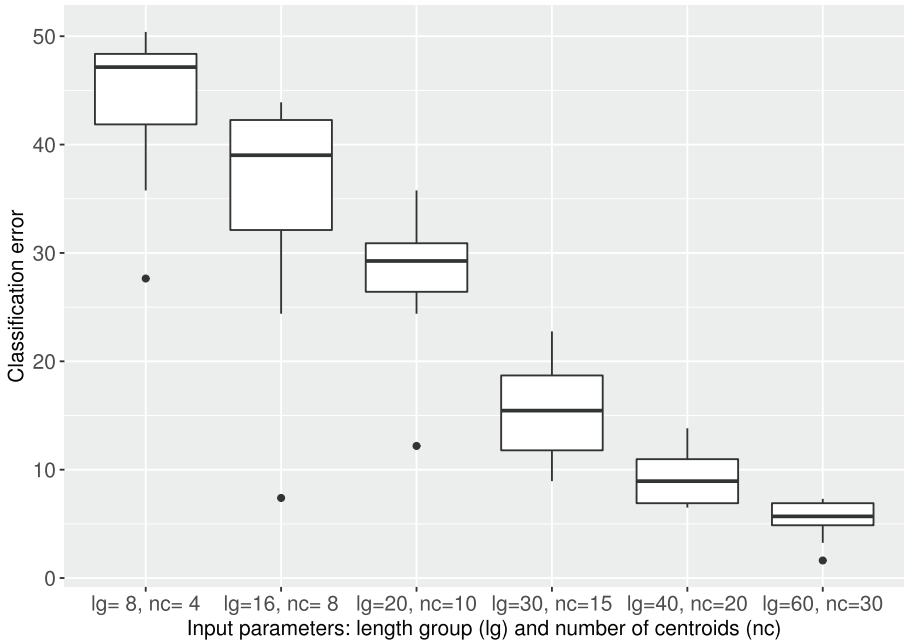


Fig. 19. Classification error obtained without stemming, for categories 'jobs', 'iron-steel' and 'cotton' in dataset $\mathscr{X}_{REUT}$.

Table 8
Recall (%) obtained by each algorithm in approximate nearest neighbour search, for different number of neighbours, with dataset $\mathscr{X}_{MNIST}$.

Algorithm	5-NN	10-NN	15-NN
KD-Tree (baseline)	100.0	100.0	100.0
FLANN	69.2	59.3	47.8
PyNNDescent	94.2	97.0	98.07
MASK	**100.0**	**100.0**	**99.98**

2. The training set acts as a group of elements to be indexed by each algorithm. Hence, we build the respective index structure on this training set for each algorithm.
3. The same random sample of 1,000 elements is selected from the testing set (which has not been previously seen by the indexes) to evaluate each algorithm's performance. Then, we execute a k-nearest neighbours query, using the index created by each algorithm and its associated search procedure, to recover the 5, 10 and 15 nearest neighbours of every point in this random sample from the testing set.

Table 8 summarizes the average recall for these 1,000 queries, testing all algorithms in each case.

As we can see in this comparison, MASK provides better recall than the other two algorithms, disregarding the number of nearest neighbours requested in the search query. Therefore, these results confirm that MASK has high performance compared with other alternative similarity search methods, for practical applications in high-dimensional and high-sparsity problems.

5. Discussion

MASK is an approximate method for similarity search based on k-means that builds multilevel index structures suitable for different scenarios, from low-dimensional problems to high-dimensional and high-sparsity datasets in metric spaces. It enables the deployment of algorithms to solve the principal types of similarity search queries. Likewise, this method performs well without needing to relocate data points after the initial index construction. The multilevel structure created by MASK in the first iteration already provides a valid approximation to solve similarity queries with reasonable efficiency.

One of the main contributions of MASK is an unconventional application of the k-means algorithm for information indexing, demonstrated in Section 3. The classical approach to clustering problems dictates that we must choose the number of centroids to group data points beforehand. Unlike this standard procedure, our approach suggests that, for information indexing, one should use as many centroids as the infrastructure can afford. This strategy is critical at the bottom of the mul-

tilevel index, next to the actual data points. When the number of centroids per data partition is higher, there is a rapid improvement in the capacity of *k*-means to distribute centroids in optimal locations, including denser data regions, scattered sections or outliers. Landmark studies describing *k*-means [15] already suggest theoretical applications of this algorithm for information indexing. In addition, several algorithms for both exact and approximate similarity search employ *k*-means to build their index [29,33,5]. However, to our knowledge, no one uses this unconventional application of *k*-means to map data flexibly according to available system resources.

On top of this, MASK is also adequate for distributed datasets with partitions stored in different nodes, thanks to its bottom-up design. As a result, different nodes can create a multilevel index in parallel, contrasting with current approaches for similarity search in distributed systems [50]. In general, alternative methods involve substantial data interchange between nodes to build the index. The management node can use MASK to aggregate metadata from the top level of all worker nodes to select those involved in resolving a query. Thanks to its multilayer design, MASK input configuration parameters can be adjusted to achieve the desired reduction in the total number of centroids stored at higher levels. Indeed, as the number of layers increases, the set of calculated prototypes for each level is reduced by a factor equal to the data summarization ratio (*groupLength*/*nCentroids*) for that level.

Another advantage of MASK is that it can handle spatially distributed datasets. For instance, consider the case of a large organization with subsidiaries in geographically scattered locations. With other indexing algorithms, it would be challenging to coalesce index metadata from each location into a centralized management system. However, MASK can also tackle this situation by gathering the top-level centroids from each venue. Even more complex hierarchies of data centres involving intermediate headquarters can be supported. In this case, each intermediate centre aggregates the centroids from all venues under its direct oversight. Then, it forwards this aggregated set of centroids upstream as required.

One of the most relevant limitations of this new indexing method is that it can only provide approximate results. The only warranty that MASK can offer is that accuracy will be higher as the number of centroids in each layer increases. Again, the data summarization ratio steers this trade-off. The trivial scenario where one centroid represents each data point (1:1 ratio) does not reduce the index storage space, although it renders perfect accuracy (exact search). Conversely, the storage gain improves by gradually reducing the number of centroids per layer at the expense of an increased loss in search accuracy. Specific applications should test different values for configuration parameters and the number of layers in the indexing structure to find the best fit for particular problems.

Another limitation of MASK is the need to use a powerful computing infrastructure to store both the dataset and index metadata. Nevertheless, continuous improvements in computing hardware and cloud architectures compensate for this shortcoming to a certain extent. Nowadays, nodes shipping large RAM units, fast secondary storage and distributed file systems are becoming prevalent in many organizations. These advances pave the way for designing new indexing algorithms that take advantage of ongoing improvements in computing infrastructure.

Despite these shortcomings, results from experiments with the $\mathscr{X}_{REUT}$ dataset and the comparison with recent alternative algorithms for approximate similarity search using the $\mathscr{X}_{MNIST}$ dataset show that MASK achieves good accuracy. Flexible and straightforward configuration parameters allow MASK to cover various applications in different domains. Hence, this new algorithm suits complex scenarios involving high-dimensional and high-sparsity datasets represented in metric spaces.

6. Conclusion

This paper presents MASK, a novel method for approximate similarity search based on an unconventional application of the *k*-means algorithm, to create a multilevel index in metric spaces. Departing from the typical utilization of *k*-means in clustering problems, where one must specify the number of centroids in advance, we demonstrate that, for data indexing, one should use as many centroids as possible to map the target dataset effectively. The capacity of the underlying computing infrastructure will impose a limit on the total number of centroids. Following this novel approach, the *k*-means algorithm will place the centroids at each level in convenient positions to map dense data regions, scattered element subsets or even outliers.

Moreover, the bottom-up design of this new method makes it suitable for distributed computing systems since each node can create a local multilayered index in parallel without needing any data transfer to other nodes. Even geographically scattered data centres under coordinated management may benefit from using this method. Index configuration can be tuned to meet design goals like index storage space, query resolution speed or accuracy. This balance between computational requirements and search performance makes it a compelling candidate for a wide range of applications demanding approximate similarity search.

Regarding future research on this topic, applying MASK to solve distributed indexing problems in wireless sensor networks and other similar types of federated and ubiquitous computing systems is a line that deserves further exploration. A recent study [40] emphasizes the relevance of effective indexing for technologies that play an essential role in smart cities, Industry 4.0 and many other settings. MASK provides clear advantages, such as eliminating the need for data exchange among device clusters, which reduces energy consumption. Likewise, MASK enables the independent construction of local indexes covering specific network regions that can work in isolation or cooperate to broaden the total coverage of the multilevel index. This capability is another important asset, as the algorithm is flexible enough to prepare an index structure that can operate in either of these two modes, seamlessly changing between them.

Additional experiments are required to evaluate MASK performance on different datasets, including specific applications such as multimedia, spatial datasets and high-dimensional problems. In the last case, it will be critical to study how the feature space dimensionality may influence MASK indexing efficiency. In the same way, a detailed analysis of MASK behaviour and performance using alternatives to the Euclidean distance in metric spaces is necessary. Although *k*-means is widely known to be tightly connected with the Euclidean distance in classical clustering applications, in data indexing, with a large number of centroids, it is possible to adopt alternative distance functions for categorical data or strings, among others. Thus, k-means can build the multilevel structure based on the distance function calculated for the pertinent cases. Results from these experiments will lead to discerning which combination of distance function and initial configuration parameters is better suited to resolve a particular similarity search problem. Eventually, this research line can also consider testing other non-metric distance functions that match specific problems.

CRediT authorship contribution statement

Felipe Ortega: Conceptualization, Methodology, Software, Investigation, Writing - original draft. **Maria Jesus Algar:** Conceptualization, Methodology, Software, Investigation, Visualization, Writing - original draft. **Isaac Martín de Diego:** Conceptualization, Methodology, Writing - review & editing, Funding acquisition. **Javier M. Moguerza:** Conceptualization, Methodology, Writing - review & editing, Supervision, Funding acquisition.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] E. Chávez, G. Navarro, R. Baeza-Yates, J.L. Marroquín, Searching in Metric Spaces, ACM Comput. Surv. 33 (3) (2001) 273–321, https://doi.org/10.1145/502807.502808.
[2] H. Samet, Foundations of Multidimensional and Metric Data Structures, The Morgan Kaufmann Series in Data Management Systems, Academic Press, 2006.
[3] P. Zezula, G. Amato, V. Dohnal, M. Batko, Similarity Search - The Metric Space Approach, Vol. 32 of Advances in Database Systems, Springer, US, 2006. doi:10.1007/0-387-29151-2.
[4] R. Baeza-Yates, B. Ribeiro-Neto, Modern Information Retrieval: The Concepts and Technology behind Search, 2nd Edition., Addison-Wesley Publishing Company, USA, 2011.
[5] M. Muja, D.G. Lowe, Scalable Nearest Neighbor Algorithms for High Dimensional Data, IEEE Trans. Pattern Anal. Mach. Intell. 36 (11) (2014) 2227–2240, https://doi.org/10.1109/TPAMI.2014.2321376.
[6] N. Tuncbag, A. Gursoy, R. Nussinov, O. Keskin, Predicting protein-protein interactions on a proteome scale by matching evolutionary and structural similarities at interfaces using PRISM, Nat. Protoc. 6 (9) (2011) 1341–1354, https://doi.org/10.1038/nprot.2011.367.
[7] R.P. Duin, E. Pekalska, The dissimilarity space: Bridging structural and statistical pattern recognition, Pattern Recogn. Lett. 33 (7) (2012) 826–832, special Issue on Awards from ICPR 2010. doi: 10.1016/j.patrec.2011.04.019.
[8] R.O. Duda, P.E. Hart, D.G. Stork, Pattern Classification, 2nd Edition., John Wiley & Sons, 2001.
[9] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, W. Equitz, Efficient and Effective Querying by Image Content, J. Intell. Inform. Syst. 3 (3/4) (1994) 231–262, https://doi.org/10.1007/BF00962238.
[10] C. Böhm, S. Berchtold, D.A. Keim, Searching in High-Dimensional Spaces: Index Structures for Improving the Performance of Multimedia Databases, ACM Comput. Surv. 33 (3) (2001) 322–373, https://doi.org/10.1145/502807.502809.
[11] T. Hastie, R. Tibshirani, J.H. Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction, second ed., Springer Series in Statistics, Springer, 2009. doi:10.1007/978-0-387-84858-7.
[12] M.L. Hetland, T. Skopal, J. Lokoc, C. Beecks, Ptolemaic access methods: Challenging the reign of the metric space model, Inform. Syst. 38 (7) (2013) 989–1006, https://doi.org/10.1016/j.is.2012.05.011.
[13] T. Skopal, B. Bustos, On Nonmetric Similarity Search Problems in Complex Domains, ACM Comput. Surv. 43 (4) (2011), https://doi.org/10.1145/1978802.1978813.
[14] M. Patella, P. Ciaccia, Approximate similarity search: A multi-faceted problem, J. Discrete Algorith. 7 (1) (2009) 36–48, https://doi.org/10.1016/j.jda.2008.09.014.
[15] J. MacQueen, Some methods for classification and analysis of multivariate observations, in: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, vol. 1: Statistics, University of California Press, Berkeley, CA, USA, 1967, pp. 281–297. URL: https://projecteuclid.org/euclid.bsmsp/1200512992.
[16] S.P. Lloyd, Least squares quantization in PCM, IEEE Trans. Inf. Theory 28 (2) (1982) 129–136, https://doi.org/10.1109/TIT.1982.1056489.
[17] V. Dohnal, C. Gennaro, P. Zezula, Efficiency and Scalability Issues in Metric Access Methods, in: A. Kelemen, A. Abraham, Y. Liang (Eds.), Computational Intelligence in Medical Informatics, vol. 85 of Studies in Computational Intelligence, Springer, 2008, pp. 235–263. doi:10.1007/978-3-540-75767-2_12.
[18] V. Gaede, O. Günther, Multidimensional access methods, ACM Comput. Surv. 30 (2) (1998) 170–231, https://doi.org/10.1145/280277.280279, URL: https://doi.org/10.1145/280277.280279.
[19] D. Comer, Ubiquitous B-Tree, ACM Comput. Survey 11 (2) (1979) 121–137, https://doi.org/10.1145/356770.356776.
[20] B.H. Bloom, Space/Time Trade-Offs in Hash Coding with Allowable Errors, Commun. ACM 13 (7) (1970) 422–426, https://doi.org/10.1145/362686.362692.
[21] J.L. Bentley, Multidimensional binary search trees used for associative searching, Commun. ACM 18 (9) (1975) 509–517, https://doi.org/10.1145/361002.361007.
[22] A. Guttman, R-Trees: A Dynamic Index Structure for Spatial Searching, in: Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data, SIGMOD '84, Association for Computing Machinery, New York, NY, USA, 1984, p. 47–57. doi:10.1145/602259.602266. URL: https://doi.org/10.1145/602259.602266.
[23] S.A. Nene, S.K. Nayar, A simple algorithm for nearest neighbor search in high dimensions, IEEE Trans. Pattern Anal. Mach. Intell. 19 (9) (1997) 989–1003, https://doi.org/10.1109/34.615448.
[24] M.L. Hetland, The Basic Principles of Metric Indexing, in: C.A.C. Coello, S. Dehuri, S. Ghosh (Eds.), Swarm Intelligence for Multi-objective Problems in Data Mining, Springer, Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 199–232, https://doi.org/10.1007/978-3-642-03625-5_9.

[25] J.K. Uhlmann, Satisfying general proximity/similarity queries with metric trees, Inform. Process. Lett. 40 (4) (1991) 175–179, https://doi.org/10.1016/0020-0190(91)90074-R.
[26] S. Brin, Near Neighbor Search in Large Metric Spaces, in: Proceedings of the 21th International Conference on Very Large Data Bases, VLDB '95, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995, p. 574–584.
[27] P. Ciaccia, M. Patella, P. Zezula, M-tree: An efficient access method for similarity search in metric spaces, in: Proceedings of the 23rd International Conference on Very Large Data Bases, VLDB '97, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997, p. 426–435.
[28] P.N. Yianilos, Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces, in: V. Ramachandran (Ed.), Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, Austin, Texas, USA, ACM/SIAM, 1993, pp. 311–321, URL:http://dl.acm.org/citation.cfm?id=313559.313789.
[29] K. Fukunaga, P. Narendra, A Branch and Bound Algorithm for Computing k-Nearest Neighbors, IEEE Trans. Comput. C-24 7 (1975) 750–753, https://doi.org/10.1109/T-C.1975.224297.
[30] M.R. Vieira, C.T. Jr., F.J.T. Chino, A.J.M. Traina, DBM-Tree: A Dynamic Metric Access Method Sensitive to Local Density Data, Journal of Information and Data Management 1 (1) (2010) 111–128. URL: http://seer.lcc.ufmg.br/index.php/jidm/article/view/22.
[31] T. Skopal, Unified framework for fast exact and approximate search in dissimilarity spaces, ACM Trans. Database Syst. 32 (4) (2007) 29, https://doi.org/10.1145/1292609.1292619.
[32] C. Faloutsos, K. Lin, FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets, in: M.J. Carey, D.A. Schneider (Eds.), Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, USA, May 22–25, 1995, ACM Press, 1995, pp. 163–174. doi:10.1145/223784.223812.
[33] H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, A.E. Abbadi, Approximate Nearest Neighbor Searching in Multimedia Databases, in: D. Georgakopoulos, A. Buchmann (Eds.), Proceedings of the 17th International Conference on Data Engineering, IEEE Computer Society, Heidelberg, Germany, 2001, pp. 503–511, https://doi.org/10.1109/ICDE.2001.914864.
[34] D. Arthur, S. Vassilvitskii, k-means++: the advantages of careful seeding, in: N. Bansal, K. Pruhs, C. Stein (Eds.), Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7–9, 2007, SIAM, 2007, pp. 1027–1035. URL: https://dl.acm.org/doi/10.5555/1283383.1283494.
[35] J. Almeida, E. Valle, R. da Silva Torres, N.J. Leite, DAHC-tree: An Effective Index for Approximate Search in High-Dimensional Metric Spaces, J. Inform. Data Manage. 1(3) (2010) 375–390. URL: http://seer.lcc.ufmg.br/index.php/jidm/article/view/82.
[36] D. Novak, P. Zezula, M-Chord: a scalable distributed similarity search structure, in: X. Jia (Ed.), Proceedings of the 1st International Conference on Scalable Information Systems, Infoscale 2006, Hong Kong, May 30-June 1, 2006, Vol. 152 of ACM International Conference Proceeding Series, ACM, 2006, p. 19. doi:10.1145/1146847.1146866.
[37] S. Nishimura, S. Das, D. Agrawal, A.E. Abbadi, MD-HBase: design and implementation of an elastic data infrastructure for cloud-scale location services, Distrib. Parallel Databases 31 (2) (2013) 289–319, https://doi.org/10.1007/s10619-012-7109-z.
[38] T. Skopal, J. Lokoc, B. Bustos, D-Cache: Universal Distance Cache for Metric Access Methods, IEEE Trans. Knowl. Data Eng. 24 (5) (2012) 868–881, https://doi.org/10.1109/TKDE.2011.19.
[39] S.V. Limkar, R.K. Jha, A novel method for parallel indexing of real time geospatial big data generated by IoT devices, Future Gener. Comput. Syst. 97 (2019) 433–452, https://doi.org/10.1016/j.future.2018.09.061.
[40] S. Wan, Y. Zhao, T. Wang, Z. Gu, Q.H. Abbasi, K.-K.R. Choo, Multi-dimensional data indexing and range query processing via Voronoi diagram for internet of things, Future Gener. Comput. Syst. 91 (2019) 382–391, https://doi.org/10.1016/j.future.2018.08.007.
[41] M. Charrad, N. Ghazzali, V. Boiteau, A. Niknafs, NbClust: An R Package for Determining the Relevant Number of Clusters in a Data Set, J. Stat. Softw. 61 (6) (2014) 1–36, https://doi.org/10.18637/jss.v061.i06.
[42] T. Kraska, A. Beutel, E.H. Chi, J. Dean, N. Polyzotis, The Case for Learned Index Structures, in: Proceedings of the 2018 International Conference on Management of Data, SIGMOD '18, Houston, TX, USA, June 10–15, 2018, Association for Computing Machinery, New York, NY, USA, 2018, p. 489–504. doi:10.1145/3183713.3196909.
[43] D. Pollard, Strong consistency of k-means clustering, Ann. Stat. 9 (1) (1981) 135–140.
[44] D. Aloise, A. Deshpande, P. Hansen, P. Popat, NP-hardness of Euclidean sum-of-squares clustering, Mach. Learn. 75 (2) (2009) 245–248, https://doi.org/10.1007/s10994-009-5103-0.
[45] A.K. Jain, R.C. Dubes, Algorithms for Clustering Data, Prentice-Hall, 1988.
[46] D.D. Lewis, Reuters-21578 Text Categorization Test Collection, Distribution 1 (1997).
[47] G.J. McLachlan, K.E. Basford, Mixture models: inference and applications to clustering, Statistics, textbooks and monographs, Marcel Dekker, New York, United States, 1988.
[48] L. Van der Maaten, G. Hinton, Visualizing data using t-SNE, J. Mach. Learn. Res. 9 (11) (2008).
[49] W. Dong, M. Charikar, K. Li, Efficient k-nearest neighbor graph construction for generic similarity measures, in: S. Srinivasan, K. Ramamritham, A. Kumar, M.P. Ravindra, E. Bertino, R. Kumar (Eds.), Proceedings of the 20th International Conference on World Wide Web, WWW 2011, Hyderabad, India, March 28 – April 1, 2011, ACM, 2011, pp. 577–586. doi:10.1145/1963405.1963487.
[50] P. Čech, J. Lokoč, Y.N. Silva, Pivot-based approximate k-NN similarity joins for big high-dimensional data, Inform. Syst. 87 (2020), https://doi.org/10.1016/j.is.2019.06.006 101410.