Efficient Semantic Similarity Search over Spatio-textual Data

George S. Theodoropoulos
University of Piraeus
Piraeus, Greece
gstheo@unipi.gr

Kjetil Nørvåg
NTNU
Trondheim, Norway
noervaag@ntnu.no

Christos Doulkeridis
University of Piraeus
Piraeus, Greece
cdoulk@unipi.gr

ABSTRACT

In this paper, we address the problem of semantic similarity search over spatio-textual data. In contrast with most existing works on spatial-keyword search that rely on exact matching of query keywords to textual descriptions, we focus on semantic textual similarity using word embeddings, which have been shown to capture semantic similarity exceptionally well in practice. To support efficient k-nearest neighbor (k-NN) search over a weighted combination of spatial and semantic dimensions, we propose a novel indexing approach (called CSSI) that ensures correctness of results, alongside its approximate variant (called CSSIA) that introduces a small amount of error in exchange for improved performance. Both variants are based on a hybrid clustering scheme that jointly indexes the spatial and textual/semantic information, achieving high pruning percentages and improved performance and scalability.

1 INTRODUCTION

Spatio-textual data [11] is generated today by GPS-enabled mobile devices and social media apps at unprecedented rates. Despite the wide potential of spatio-textual data analysis, existing approaches for querying spatio-textual data largely rely on exact matching of query keywords to the textual descriptions [6, 7, 9, 28]. In consequence, retrieval results are of inferior quality, as the semantic relationships between words cannot be accurately captured. Motivated by this limitation, we employ word embeddings, a state-of-the-art technique in natural language processing that has been shown to work exceptionally well in different domains [32].

Using word embeddings, a word can be represented as a dense, real-valued vector of high dimensionality, so that vectors that are close, correspond to words with similar meaning. The word embeddings are created based on a large corpus of text, typically either by training a neural network, with prominent example being Word2Vec [30], or using co-occurence matrices, like Glove [32]. In order to find semantically similar documents, we can employ document embeddings. Document embeddings are also dense, real-valued vectors of high dimensionality, with one vector for each document (a document here meaning a text of one or more words). They can be created by combining the word embeddings of words in a document, for example by averaging the word embedding vectors [36]. Then, by finding similar vectors based on a similarity function, we can find documents that are likely to be semantically similar [30].

Accordingly, we formulate the problem of semantic similarity search over spatio-textual data. Given a query specified by a location and a textual description (typically in the form of keywords), the aim is to retrieve the k objects that minimize a distance function defined as a linear, weighted combination of spatial with semantic distance. The technical challenge is twofold. On the one hand, the semantic vectors are high-dimensional, thus building an effective index that allows eager pruning of the search space is extremely difficult. On the other hand, designing a query processing algorithm that supports different ways of weighting the contribution of the spatial and semantic distance is far from trivial.

To address this problem, we propose a novel algorithm called CSSI (Cluster-based Semantic Spatio-textual Indexing) for k-nearest neighbor (k-NN) search over spatio-textual data. The gist of our approach is to group spatio-textual objects into hybrid clusters, by combining two separate sets of clusters: spatial and semantic. Capitalizing on this data organization, we are able to define appropriate search bounds that allow pruning of clusters that cannot contain query results. Moreover, for those clusters that cannot be pruned a priori, we show that we can still prune a significant portion of the enclosed objects. Compared to the state-of-the-art S^2R-tree [8] index that enriches an R-tree with semantic information, we show that our algorithm is consistently faster. Intuitively, the reason is that we jointly index the spatial and the semantic space, whereas S^2R-tree (similarly to previous approaches) prioritizes the spatial domain by following a spatial-first indexing, which often results in having to scan the entire data set. Furthermore, we propose an approximate algorithm CSSIA, which trades accuracy for performance, thus greatly improving the execution time of CSSI while returning the correct result set in most cases. The main technical novelty of our work and differentiating factor with previous work is the creation of a clustering-based index that is applicable to complex, high-dimensional spatio-textual data, and the use of hybrid clusters that *jointly* index the spatial and semantic domains.

In summary, we make the following contributions:

- We propose a novel indexing approach based on hybrid clusters for semantic spatio-textual similarity search.
- We introduce an exact query processing algorithm (CSSI) for this problem, along with the appropriate bounds, which is provably correct (Sect. 4).
- To boost the performance of query processing, we propose an extremely efficient algorithm (CSSIA) that retrieves approximate results very fast, while keeping the error lower than 1% in most cases (Sect. 5).
- We present the results of an extensive experimental evaluation that demonstrates the efficiency and effectiveness of our algorithms, for large real-world data sets and in comparison with the state-of-the-art [8] (Sect. 7).

In addition, we review related work in Sect. 2, we formulate the problem in Sect. 3, we discuss the complexity of our algorithm and handling dynamic data in Sect. 6, and we conclude in Sect. 8.

2 RELATED WORK

Spatial Keyword Search. Searching with a combination of spatial and textual criteria has attracted increased attention recently [6, 7, 9, 28]. Regarding textual matching, early attempts to tackle spatial-keyword search focus primarily on exact matching using Boolean constraints [6]. Relaxed variants of matching based on partial keyword matching are also widely used, such as ranked retrieval using a single distance function that combines spatial with textual similarity [6, 12, 44], as well as using separate similarity functions for spatial and textual matching [40]. Regarding spatio-textual indexing, existing approaches can be classified as spatial-first [15, 26], text-first [37, 42], or interleaved [27, 43]. Other related topics include proportionality in spatial-keyword search [25], spatial object selection with diversity [18], spatial group keyword queries [3, 4, 17], and error-tolerant spatio-textual retrieval [46, 47]. Nevertheless, a common limitation of all these works is that they fail to go beyond syntactic matching or (in best case) fuzzy matching of keywords.

Semantic Representations of Text. The state-of-the-art semantic text representations for data analytics is based on the concept of word embeddings. In addition to on Word2Vec [30] and Glove [32] as already mentioned, recently also context-aware word embeddings methods have appeared, with some representative approaches being ELMo [33] and BERT [13], which take into account the neighboring words in a phrase (the context) in order to deal with polysemous words. More efficient support for sentence-to-sentence similarity calculations has later been introduced in SentenceBERT [36].

Semantic Spatio-textual Search. The state-of-the-art for semantic spatio-textual search is the S^2R-tree [8], which addresses the same problem as in this paper. Practically, it is an extension of the R-tree built on top of spatial coordinates and very low-dimensional semantic vectors that represent the text. The semantic vectors are obtained by projecting the high-dimensional word embeddings to an m-dimensional space using a pivot-based technique. Thus, each semantic vector is finally represented as an m-dimensional vector (with m as low as 2) that keeps the distance of the specific vector from the m pivots in the high-dimensional space. The S^2R-tree is built on spatial coordinates (spatial layer), with every leaf being further built into an R-tree that indexes the m-dimensional representations (semantic layer). Then, index nodes are augmented with m-dimensional MBBs of semantic vectors in a bottom-up way. The query processing algorithm performs best-first traversal of the index using a priority queue, which prioritizes access to nodes based on lower bound of distance. The search terminates as soon as the element at the top of the priority queue has larger distance than the top-k element found so far. However, an important shortcoming is that it follows the spatial-first approach, as the index is built based on spatial coordinates. Also, as will be shown in our experiments, its performance is similar to a pure R-tree based solution that indexes the spatial domain, and due to low pruning in the semantic space it is often worse than a linear scan solution.

The NIQ-tree [34] is a multi-level indexing structure that also belongs to the spatial-first approach. At the top level, a Quadtree is built on the spatial coordinates and then, at the second level, the objects of each leaf are indexed based on topic relevance. For this, Latent Dirichlet Allocation (LDA) is used in order to create a probabilistic topic model that captures semantics. Finally, at the last level, n-gram inverted lists are constructed. Later, the approach is extended by introducing another index, called LHQ-tree [35]. The main difference to our work is the fact that they use topic modelling for the semantic representation, instead of word embeddings, and (in addition) they need to index text. Moreover, they follow a spatial-first approach too. It should also be noted that in [8], the S^2R-tree has been compared with an adaptation of NIQ-tree (an approach based on spatial-first, followed by search in semantic dimensions), and the S^2R-tree shows superior performance.

Symbol	Description
O	Data set of spatio-textual objects
q	Query object with location $(q.x, q.y)$ and text $q.text$
$d_s(q, o)$	Spatial distance of objects' geographical coordinates
$d_t(q, o)$	Semantic distance of objects' textual representations
$d'_t(q, o)$	Semantic distance in the projected space
λ	Parameter balancing spatial and semantic distance
$d(q, o)$	Distance between spatio-textual object o and query q
n	Dimensionality of semantic vectors
m	Dimensionality of projected semantic vectors
(C^s, R^s)	Spatial cluster with centroid C^s and radius R^s
(C^t, R^t)	Semantic cluster with centroid C^t and radius R^t
$K_s \cdot K_t$	Max. number of hybrid clusters (spatial · semantic)
C	Hybrid cluster $C = \langle C^s, R^s, C^t, R^t \rangle$
$\mathcal{H}$	Set of hybrid clusters
A_i	Sorted array of objects in the i-th hybrid cluster
$L(q, C)$	Lower bound of distance of q to hybrid cluster C
o_{nn}	The current k-th nearest neighbor

Table 1: Main symbols used in the paper.

A different approach to semantic search is followed in [5, 39, 45] using a probabilistic topic model (LDA). Sun et al. [39] enhance spatial keyword queries with semantics, but they use LDA to extract semantic representations. To address the problems (a) that queries are short and LDA cannot infer the semantics, and (b) of semantic ambiguity, they rely on user interactions (feedback) on query results. This is different from our approach, which does not require user interaction. In [5, 45], the authors also consider semantic retrieval along with spatial proximity, however they also use LDA. More importantly, their query is different than ours as they try to find a set of (at most) k objects that collectively optimize a distance function, where k is set by the number of query keywords. As such, these approaches are not directly comparable to out work.

High-dimensional k-NN Search. Recently, several approaches for k-NN search in high-dimensional data have been proposed, the state-of-the-art being HNSW [29], provided by, e.g., the Faiss library [24]. Although very efficient for approximate k-NN search, they are not applicable in the context of multi-aspect distance functions, as in the case of combining spatial and semantic distances. The reason is that a separate index would need to be built for each possible combination of spatial and semantic distances. Also, recent approaches based on learned indexes [38, 41] either cannot be trivially become applicable to our setting [38] or their performance for really high-dimensional data still remains unclear [41].

Multi-metric indexing. Our work is also relevant to multi-metric indexing, where combined search can be performed over multiple metric spaces. Notable methods include the M^2-tree [10], the methodology of [2] for adapting metric indexes, the RR*-tree [16] and most recently DESIRE [48]. We provide a comparison with the latter two works in Sect. 7.

3 PROBLEM FORMULATION

In this paper, we address the problem of semantic similarity search over spatio-textual data. That is, given a collection of spatio-textual objects $\{o_i\} \in O$ consisting of a location $(o_i.x, o_i.y)$

and a textual description $o_i.text$ and a query q with a location $(q.x, q.y)$ and a textual description $q.text$, retrieve the k objects in O that minimize the distance function $d(q, o)$ that takes into account both the semantic and the spatial distances of any object to the query.

As such, the distance function $d(q, o)$ includes two components. The first component is the spatial distance $d_s(q, o)$ of the query location $(q.x, q.y)$ to an object's location $(o.x, o.y)$. We use a normalized variant of the Euclidean distance for $d_s(q, o)$, obtained by dividing with the maximum possible Euclidean distance D_s^{max} of any two objects in the data set:

$$d_s(q, o) = \frac{\sqrt{(q.x - o.x)^2 + (q.y - o.y)^2}}{D_s^{max}}$$

The second component is the semantic distance $d_t(q, o)$ between $q.text$ and $o_i.text$. To define this distance, we use word embeddings to represent the textual description of a spatio-textual object o as a dense n-dimensional vector: $\{o[1], o[2], \ldots, o[n]\}$. This vector is called the semantic representation (or semantic vector) of the spatio-textual object. In principle, any method for word embeddings can be used, e.g., Word2Vec [30] or Glove [32], as our algorithms do not make any assumption on the specific method. Then, the semantic distance $d_t(q, o)$ is defined as the normalized Euclidean distance applied on the semantic vectors of q and o, divided by D_t^{max} (the maximum possible Euclidean distance)[1]:

$$d_t(q, o) = \frac{\sqrt{(q[1] - o[1])^2 + \cdots + (q[n] - o[n])^2}}{D_t^{max}}$$

Having both constituent parts normalized in $[0, 1]$, we consider the following equation for the distance function:

$$d(q, o) = \lambda \cdot d_s(q, o) + (1 - \lambda) \cdot d_t(q, o) \qquad (1)$$

where $\lambda \in [0, 1]$ is an user-dependent parameter that balances the contribution of the two parts to the final distance function.

Problem 1. (Semantic Spatio-textual Similarity Search) Given a collection of spatio-textual objects $\{o_i\} \in O$ and a query object q, retrieve the k objects $O_k = \{o_1, \ldots, o_k\}$, such that: $d(q, o_i) \leq d(q, o_j), \forall o_i \in O_k$ and $\forall o_j \in O - O_k$.

4 THE CSSI ALGORITHM

In this section, we present our approach, called CSSI (Cluster-based Semantic Spatio-textual Indexing), for efficient semantic indexing and search of spatio-textual data.

4.1 Index Construction

In order to support efficient search, we organize data based on both spatial and textual dimensions. We argue in favor of joint indexing of the two domains, because the spatial-first approach (e.g., [8, 35]) that prioritizes the easily indexed spatial domain would result in identifying objects that are spatially close to the query, but would have practically no information about their semantic similarity to the query (and vice-versa). Thus, we propose a highly descriptive "hybrid" representation that combines both domains, grouping together objects that are both spatially close and semantically similar.

[1]Since finding the exact values of D_t^{max} and D_s^{max} is not scalable, we use a conservative estimate based on distance from a virtual point having the minimum of the values of each dimension in the data set, to a point having maximum values.

Algorithm 1 Index construction of CSSI

```
1:  function CreateIndex($O, K_s, K_t, m$)
2:      $C_s \leftarrow$ K-Means($\pi_{x,y}(O), K_s$)                ▷ Spatial clustering
3:      $\{C_i^s\} \leftarrow C_s$.cluster_centers
4:      $\{R_i^s\} \leftarrow \max\{d_s(o, C_i^s)\}, \forall o \in C_i^s$
5:      $O_{text} \leftarrow$ WordEmbeddings($\pi_{text}(O)$)
6:      $O'_{text} \leftarrow PCA(O_{text}, m)$            ▷ Dimensionality reduction
7:      $C_t \leftarrow$ K-Means($O'_{text}, K_t$)               ▷ Textual clustering
8:      $\{C_i^t\} \leftarrow C_t$.cluster_centers
9:      $\{R_i^t\} \leftarrow \max\{d_t(o, C_i^t)\}, \forall o \in C_i^t$
10:     $\mathcal{H} \leftarrow \{(C_i^s, R_i^s), (C_j^t, R_j^t)\}$         ▷ Initialize hybrid clusters
11:     for $o \in O$ do                              ▷ Object assignment
12:         assign $o$ to its hybrid cluster
13:     for $i \in [1, K_s \cdot K_t]$ do                  ▷ For each hybrid cluster
14:         create array $A_i$        ▷ Organize objects in $i$-th hybrid cluster
15:     return $\mathcal{H}$
```

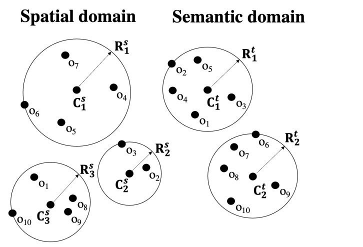
Spatial clustering. We perform two separate clustering algorithms to the spatio-textual data set. First, regarding the spatial information, we group together objects based on their location by applying spatial clustering. We obtain a set of K_s *spatial clusters*, each represented by a centroid C_i^s and a radius R_i^s that is equal to the distance between C_i^s and the furthest point assigned to C_i^s:

$$R_i^s = \max\{d_s(o, C_i^s)\}$$

Semantic clustering. Regarding the textual information, we make use of pre-trained models that produce high-quality word embeddings, obtaining high-dimensional dense semantic vectors that represent the aforementioned object. These semantic vectors are n-dimensional, with n in the order of hundreds. Unfortunately, grouping such high-dimensional data by computing pairwise distances is known to be ineffective due to the *curse of dimensionality* [1, 21]. To tackle this problem, we apply Principal Component Analysis (PCA) in order to project the n-dimensional word embeddings to meaningful m-dimensional ($m << n$) vectors. The choice of PCA is due to its salient property of maximizing the total variance of the projection. Then, we apply K-Means on the projected vectors, which produces K_t *semantic (textual) clusters*. Note that any clustering algorithm producing spherical clusters can be used and we choose K-Means for its efficiency.

Nevertheless, we emphasize that the *representations of the semantic clusters* are based on the *original n-dimensional space*. In more detail, each semantic cluster is represented by a centroid C_i^t which is the mean vector for all n-dimensional vectors in the cluster, and by a radius R_i^t, which is equal to distance (in the original n-dimensional space) between C_i^t and the furthest point that belongs to the cluster: $R_i^t = \max\{d_t(o, C_i^t)\}$.

Building the Index. Algorithm 1 describes the complete process of index construction for CSSI. Essentially, the index consists of a set of hybrid (spatio-textual) clusters $\mathcal{H}$ that organize the data objects of data set O. In the first step, we obtain a spatial clustering (denoted C_s) by applying K-Means to the spatial coordinates (by projecting O, denoted as $\pi_{x,y}(O)$) of each $o \in O$ (line 2). Then, we obtain semantic textual vectors (O_{text}) by applying word embeddings on the projected textual descriptions $\pi_{text}(O)$. By applying PCA we obtain highly descriptive low-dimensional representations, denoted by O'_{text}, which result in a clustering of high quality. In the next step, the semantic clustering (denoted C_t) is obtained by applying K-Means to the low-dimensional representations (line 7). Then, we initialize the set of hybrid clusters $\mathcal{H}$ by forming combinations of spatial (C_i^s, R_i^s) with semantic clusters (C^t_j, R^t_j) (line 10). Finally, each object o is assigned to a single hybrid cluster, which is the one formed by o's spatial and semantic clusters respectively. For the centroid of a hybrid cluster, we use $C^s \circledcirc C^t$ to represent the concatenation of the spatial centroid C^s and the textual centroid C^t respectively[2]. Then, the distance of an object o to the centroid of its hybrid cluster is: $d(o, C^s \circledcirc C^t) = \lambda \cdot d_s(o, C^s) + (1 - \lambda) \cdot d_t(o, C^t)$.



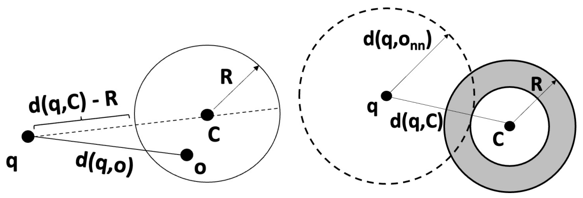
Hybrid cluster	Objects
$C^s_1 \circledcirc C^t_1$	o_4, o_5
$C^s_2 \circledcirc C^t_1$	o_2, o_3
$C^s_3 \circledcirc C^t_1$	o_1
$C^s_1 \circledcirc C^t_2$	o_6, o_7
$C^s_2 \circledcirc C^t_2$	-
$C^s_3 \circledcirc C^t_2$	o_8, o_9, o_{10}

Figure 1: Example of 6 hybrid clusters.

Example 4.1. Fig. 1 presents an illustrative example of how the hybrid clusters are formed. In the example, we have $K_s = 3$ spatial clusters $\{(C^s_1, R^s_1), (C^s_2, R^s_2), (C^s_3, R^s_3)\}$ and $K_t = 2$ semantic clusters $\{(C^t_1, R^t_1), (C^t_2, R^t_2)\}$. As a result, $K_s \cdot K_t = 6$ hybrid clusters can be formed. We make two observations: (a) the hybrid cluster $(C^s_2, R^s_2), (C^t_2, R^t_2)$ is not actually formed, since there exists no data object that belongs to both clusters, and (b) each spatio-textual data object is eventually assigned to a *single* hybrid cluster.

In order to make the index more efficient, we organize objects within a hybrid cluster in such a way that will allow pruning of unnecessary objects at query time. However, notice that λ is user-dependent (i.e., that can differ for each query), and therefore cannot be exploited during index construction. Thus, we can organize the objects in two sorted lists L_t and L_s, one sorted on distance $d_t(o, C^t)$ to the semantic cluster centroid, and the second on distance $d_s(o, C^s)$ to the spatial cluster centroid (each object in the cluster will be a member once in each list). Then, we can apply Fagin's Threshold Algorithm (TA) [14] using λ at query time, which allows access to the next object based on distance $d(o, C^s \circledcirc C^t)$ and supports early termination without exhaustively accessing all objects.

In more detail, when the Threshold Algorithm is applied at query time, the lists L_t and L_s will be processed in parallel, one object from each list, and distance to centroid will be calculated. However, an object already seen in one of the lists might later be seen in the other list. A second distance calculation will then be redundant, as its distance has already been calculated. In order to avoid this, as well as reduce memory usage, we create a *single* array A_i for cluster i (line 14, Alg. 1), which is used during query processing, that has only one element per object. Each element is a tuple consisting of an object, and spatial and semantic distances. The array is created by traversing the two sorted lists L_t and L_s in parallel, processing a pair of objects at a time. For each of the objects $o \in L_s, o' \in L_t$ in the pair, a new element is added to the array only if this is the object's first occurrence. The new element consists of three parts: the seen object $A_i[j].o$, the spatial distance $A_i[j].ds = d_s(o, C^s)$ for the one object o in the pair from the list sorted by distance to spatial cluster centroid, and the semantic distance $A_i[j].dt = d_t(o', C^t)$ for the other object o' in the pair



(a) Lower bound on distance. **(b) Intra-cluster pruning.**

Figure 2: Illustration of pruning.

from the list sorted by distance to semantic cluster centroid (i.e., one of the distances will be from the other object in the pair). As such, for each pair of objects, two, one, or zero elements might be added to the array, depending on whether the objects have been seen before or not. The resulting array has one element for each object in the cluster, and is thus only half the size as the sum of the original two lists, and can be processed very efficiently at query time.

4.2 Lower Bound on Distance to Cluster

The question that needs to be answered next is how these hybrid clusters can be exploited during query processing in order to efficiently retrieve the k nearest neighbors of a given query object q. To this end, we use the concept of a lower bound that represents the minimum distance between a query point q and any object o that belongs to the hybrid cluster. We note that our theoretical results hold for arbitrary metric spaces.

For clarity, we first explain the concept of lower bound for a query q and a cluster (C, R) with centroid C and radius R, assuming that q *is not within distance* R from C (q is located outside a sphere centered at C with radius R).

Lemma 4.2. *Given a query q that is not enclosed in a cluster (C,R), the distance of q to any object o that belongs to this cluster is at least equal to $L(q, C, R) = d(q, C) - R$:*

$$d(q, o) \geq L(q, C, R)$$

Our first observation is that Lemma 4.2 is only valid when q is not enclosed in the cluster, i.e., $d(q, C) \geq R$. In case q is enclosed in the cluster (C,R), we cannot derive a meaningful (i.e., non-zero) lower bound for $d(q, o)$.

We stress that Lemma 4.2 holds for any metric distance function $d()$, as is relies on the triangular inequality, also illustrated in Fig. 2a. It is straightforward to show that this bounding scheme is applicable both for spatial as well as for semantic clusters. In other words, we can define the spatial lower bound $L(q, C^s, R^s)$ and the semantic lower bound $L(q, C^t, R^t)$ accordingly:

$$d_s(q, o) \geq L(q, C^s, R^s) = d_s(q, C^s) - R^s \tag{2}$$

$$d_t(q, o) \geq L(q, C^t, R^t) = d_t(q, C^t) - R^t \tag{3}$$

Now, we are ready to formulate a lower bound $L(q, C)$ for a query object and a hybrid cluster $C = \langle C^s, R^s, C^t, R^t \rangle$ that consists of a spatial (C^s, R^s) and a semantic cluster (C^t, R^t). We consider the cases: ❶ $d_s(q, C^s) \geq R^s \wedge d_t(q, C^t) \geq R^t$, ❷ $d_s(q, C^s) \geq R^s \wedge d_t(q, C^t) < R^t$, ❸ $d_s(q, C^s) < R^s \wedge d_t(q, C^t) \geq R^t$, and ❹ $d_s(q, C^s) < R^s \wedge d_t(q, C^t) < R^t$.

$$L(q, C) = \begin{cases} \lambda \cdot L(q, C^s, R^s) + (1 - \lambda) \cdot L(q, C^t, R^t) & ❶ \\ \lambda \cdot L(q, C^s, R^s) & ❷ \\ (1 - \lambda) \cdot L(q, C^t, R^t) & ❸ \\ 0 & ❹ \end{cases} \tag{4}$$

[2]For convenience, we omit the subscript i, j from a cluster centroid and radius when we refer to a single cluster.

Lemma 4.3. (Lower bound on distance) *Given a query q and a hybrid cluster $C = \langle C^s, R^s, C^t, R^t \rangle$, the distance of any object $o \in C$ to the query q is at least equal to $L(q, C)$:*

$$d(q, o) \geq L(q, C)$$

Proof. By contradiction. Assume that there exists an object $o \in C$ such that: $d(q, o) < L(q, C)$. By definition $d(q, o) = \lambda \cdot d_s(q, o) + (1 - \lambda) \cdot d_t(q, o)$. We separate the following cases:
Case 1: q is not enclosed in (C^s, R^s) nor in (C^t, R^t): Then, by substitution based on Eq. 2 and 3: $d(q, o) \geq \lambda \cdot (d_s(q, C^s) - R^s)) + (1 - \lambda) \cdot (d_t(q, C^t) - R^t))$ and $d(q, o) \geq \lambda \cdot L(q, C^s, R^s) + (1 - \lambda) \cdot L(q, C^t, R^t)$, which leads to $d(q, o) \geq L(q, C)$ (contradiction).
Case 2: q is enclosed in (C^s, R^s) but not in (C^t, R^t): Then, by substitution based on Eq. 2: $d(q, o) \geq \lambda \cdot (d_s(q, C^s) - R^s)) + (1 - \lambda) \cdot d_t(q, o)$ and $d(q, o) \geq \lambda \cdot L(q, C^s, R^s) + (1 - \lambda) \cdot d_t(q, o) \geq \lambda \cdot L(q, C^s, R^s)$ (since $(1 - \lambda) \cdot d_t(q, o) \geq 0$), which leads to $d(q, o) \geq L(q, C)$ (contradiction).
Case 3: q is enclosed in (C^t, R^t) but not in (C^s, R^s): This is symmetric to Case 2.

□

4.3 Inter- and Intra-cluster Pruning

Let o_{nn} denote the (currently found) k-th nearest neighbor at any stage of query processing and $d(q, o_{nn})$ denote its distance to the query point q. Obviously, in the beginning of query processing, no objects have been accessed and o_{nn} is undefined, which implies that $d(q, o_{nn}) = \infty$. Also, it follows that in order to initialize o_{nn}, it is necessary to have accessed at least k data objects.

In brief, our approach examines each hybrid cluster according to an ordering, and attempts to stop processing and report the k-nearest neighbors without exhaustively searching all of the hybrid clusters and their contents. In the following, we present the pruning properties used by our approach. First, we explain how to prune complete hybrid clusters from processing (*inter-cluster pruning*). Then, we present how we can avoid processing all objects within a hybrid cluster (*intra-cluster pruning*).

Inter-cluster Pruning. In order to prune a hybrid cluster C from processing, the condition that must hold is that no object within the cluster can have a smaller distance than the one between q and the current k-th nearest neighbor o_{nn}.

Lemma 4.4. (Pruning property 1) *A hybrid cluster C can be safely pruned, if its lower bound $L(q, C)$ with respect to query object q is greater than the distance $d(q, o_{nn})$ between q and the current k-th nearest neighbor:*

$$L(q, C) \geq d(q, o_{nn})$$

Intra-cluster Pruning. In addition to pruning an entire hybrid cluster based on its derived lower bound, we also provide a method that can prune some objects of a cluster by indexing the distance to the centroid [23]. To achieve this, we would need to access the data objects belonging to a hybrid cluster based on descending distance $d(o, C)$ to the cluster centroid. At any point during this process, we can stop accessing further data objects if the following condition holds.

Lemma 4.5. (Pruning property 2) *During accessing the objects o of a hybrid cluster C in descending order of distance to the centroid $d(o, C)$, we can terminate processing of C safely if:*

$$d(q, C) - d(o, C) > d(q, o_{nn})$$

Proof. By contradiction. Assume that there exists an object o' with smaller distance to the centroid than the current object o: $d(o', C) < d(o, C)$, and that this object belongs to the k-nearest neighbors: $d(q, o') < d(q, o_{nn})$. Starting from: $d(q, C) - d(o, C) > d(q, o_{nn}) > d(q, o')$, which can be rewritten as: $d(q, C) - d(q, o') > d(o, C) > d(o', C)$. Thus, we have: $d(q, C) > d(q, o') + d(o', C)$, a contradiction, since it violates the triangular inequality. □

Example 4.6. In Fig. 2b, the circle centered at q with radius $d(q, o_{nn})$ indicates the area in which we could potentially find a better object than the k-nearest neighbors so far. The grey area inside the cluster indicates exactly those objects of the cluster (C, R) that need to be examined, assuming access order based on descending distance to the centroid, as any of those could be a better result. The white area inside the cluster is defined by the value $d(q, o_{nn})$ and it indicates the area that can be safely pruned based on Lemma 4.5.

However, $d(o, C)$ relies on the value of λ which may vary for different queries. Therefore, we cannot have an ordering at indexing time based on $d(o, C)$. To solve this problem, we use the sorted array A, which uses an ordering based on $d_s(o, C^s)$ and $d_t(o, C^t)$, as described in Sect. 4.1. It is easy to show that the distance computed for an element e of the array A is a conservative approximation of $d(o, C)$, i.e., $d(o, C) \leq \lambda \cdot e.ds + (1 - \lambda) \cdot e.dt$. By using this conservative approximation during query processing, we can save many distance computations and still obtain a correct result.

Algorithm 2 Query processing CSSI

```
1:  function CSSI($\mathcal{H}$, $q$, $k$)
2:      $\mathcal{R} \leftarrow \emptyset$                                  ▷ Result set of size $k$
3:      $U \leftarrow \infty$                          ▷ Distance to the current $k$-NN
4:      sort $\mathcal{H}$ based on $L(q, C)$ ascending
5:      for each hybrid cluster $C \in \mathcal{H}$ do
6:          if $L(q, C) \geq U$ then
7:              break                                      ▷ Pruning Property 1
8:          for each element $e \in A$ of $C$ (let $o = e.o$) do
9:              if $q$ is not enclosed in cluster $C$ then
10:                 $bound \leftarrow \lambda \cdot e.ds + (1 - \lambda) \cdot e.dt$
11:                 if $d(q, C) - bound > U$ then
12:                     break                              ▷ Pruning Property 2
13:             if $d(q, o) < U$ then
14:                 $\mathcal{R} \leftarrow \mathcal{R} \cup \{o\}$
15:                 if $\mathcal{R}.size > k$ then
16:                     $\mathcal{R} \leftarrow \mathcal{R} \setminus \{o'\}, o' = \arg\max_{o_i \in \mathcal{R}} d(q, o_i)$
17:                 if $\mathcal{R}.size = k$ then
18:                     $U \leftarrow \max_{o_i \in \mathcal{R}} d(q, o_i)$
19:     return $\mathcal{R}$
```

4.4 Query Processing

Algorithm 2 presents how query processing is performed exploiting the set of hybrid clusters $\mathcal{H}$. We use a heap $\mathcal{R}$ for maintaining the k-nearest neighbors so far, organized based on distance to q. Also, we use U to denote the distance $d(q, o_{nn})$ between q and the current k-th nearest neighbor o_{nn}, and initialize its value to be equal to infinity. The value U will be updated as soon as we have accessed k data objects. Practically, U serves as an upper bound for distance (i.e., objects with higher distance $d(q, o) > U$ can be safely discarded).

The first step is to sort the set $\mathcal{H}$ based on the lower bound $L(q, C)$ of distance to the query in ascending order (line 4). This prioritizes access to the hybrid clusters that are closer to the

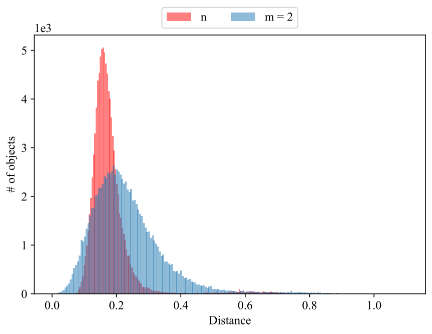
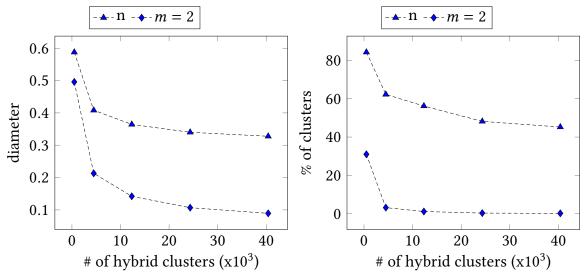


Figure 3: Distribution of distances to a random query in the original (n) and in the projected ($m = 2$) space.



(a) Average diameter of clusters. **(b) % of clusters enclosing q.**

Figure 4: Overlap of hybrid clusters for increasing number of clusters (5M tweets, default parameters, cf. Sect. 7).

query object, thus yielding higher probability to contain the k-nearest neighbors. Then, each hybrid cluster is accessed based on the sorted order. As soon as we encounter a hybrid cluster with lower bound larger or equal to the distance U, we can terminate processing based on pruning property 1 (Lemma 4.4, inter-cluster pruning).

For each hybrid cluster C that cannot be pruned, we start examining each object that belongs to C. Recall that the order of accessing data objects is based on the sorted array A that keeps the objects of the hybrid cluster C. If q is not inside C's bounds (line 9), we can try to prune some of the data objects that are part of C (intra-cluster pruning). More concretely, as soon as we access a data object that satisfies the condition at line 11, then it also satisfies pruning property 2, and we can safely terminate processing of this cluster based on Lemma 4.5. In case we cannot prune the current data object o, we compute its distance $d(q, o)$ and if it is greater than the distance of the current k-th nearest neighbor $d(q, o_{nn})$ we proceed to the next data object (line 13). Otherwise, we add o to $\mathcal{R}$ and if it contains $k + 1$ neighbors, we remove the object that is the furthest away from q (line 16). Finally, we update the value of $d(q, o_{nn})$ to reflect the distance to the current k-th nearest neighbor (line 18).

Lemma 4.7. *(*Correctness*)* CSSI *returns the correct result.*

Proof. By contradiction. Consider that the k-th nearest neighbor o_{nn} is false positive, i.e., it is returned by the algorithm in set $\mathcal{R}$, yet there exists another object $o' \notin \mathcal{R}$ such that $d(q, o') < d(q, o_{nn})$. Then, o' must have been pruned either by pruning property 1 or 2 or by the check in line 13. Pruning property 1 cannot prune o', since o' would be in a cluster C with $L(q, C) \leq d(q, o') < d(q, o_{nn})$ (contradiction). Similarly, pruning property 2 cannot prune o' because $d(q, C) - d(o', C) < d(q, o_{nn})$ (based on the triangular inequality). Finally, the check in line 13 would be evaluated to true for o', which means that $d(q, o') < d(q, o_{nn})$ (contradiction). □

5 AN APPROXIMATE ALGORITHM: CSSIA

In this section we present CSSIA, an algorithm that returns highly accurate results while being significantly faster than CSSI. Intuitively, CSSIA relies on the same hybrid clustering scheme as CSSI, but CSSIA's cluster representations of the semantic vectors are based on the projected m-dimensional space. This means that we can no longer offer theoretical guarantees of correctness, yet – in practice – CSSIA's results are highly accurate, with the subsequent error rate being less than 1% in most cases.

5.1 Intuition

Conceptually, having hybrid clusters represented in the original high-dimensional space presents two shortcomings: (a) low variance of distance distribution, an inherent characteristic of high-dimensional vectors, and (b) large-sized clusters that capture significant amount of space, thus leading to highly overlapping clusters. Both of these factors affect the quality of any object grouping method and make pruning harder. We justify these observations using two experiments, as outlined below. In turn, these findings motivate the design of CSSIA.

Fig. 3 shows two histograms of the distribution of distances to a random query, at the original n-dimensional space (n=100) and for the m-dimensional projection (m=2) respectively, for a data set of 1M tweets (as described in Sect. 7). As we can see, the distance distribution is much narrower in the n-dimensional space, something that is bound to happen as the variance of the score is relatively low, equal to 0.0046. On the other hand, the histogram for $m = 2$ is much wider, with its variance being more than double of the n-dimensional one at 0.01. This difference in variance greatly affects the quality of any grouping method applied to the underlying data, thus affecting any index that relies on that grouping as well.

Fig. 4a shows the average diameter of hybrid clusters, for an increasing number of clusters (x-axis). Fig. 4b shows the percentage of hybrid clusters that enclose a random query. In both cases, we compare the n-dimensional vectors with the m-dimensional (m=2) ones. For the n-dimensional vectors, the rate of decrease of the cluster diameter is almost negligible if more than 5K clusters are created (Fig. 4a). At the same time, the overall percentage of clusters that enclose the query is between 55% and 60% for up to 40K clusters (Fig. 4b). On the other hand, for m=2, we observe that the rate of decrease in overlap is much greater, with the percentage of clusters enclosing the query being close to 0% for 5K clusters or more.

Consequently, the hybrid clusters in n dimensions will have a lot of overlap, something expected due to the low variance of the distribution of distances and the large area that is covered in the n-dimensional space. This will make pruning clusters harder for CSSI. In contrast, the pruning in CSSIA is improved because the representations of hybrid clusters will have smaller overlap.

5.2 Index Construction

CSSIA adopts the same hybrid clustering scheme as CSSI, as explained in Sect. 4.1. However, the cluster representations in the semantic domain are instead based on *the projected space*. More

Algorithm 3 Query processing CSSIA

```
function CSSIA($\mathcal{H}$, $q$, $k$)
    $\mathcal{R} \leftarrow \emptyset$                                   ▷ Result set of size $k$
    $U \leftarrow \infty$                  ▷ Distance to current $k$-NN in original space
    $U' \leftarrow \infty$                 ▷ Distance to current $k$-NN in projected space
    sort $\mathcal{H}$ based on $L'(q, C)$ ascending
    for each hybrid cluster $C \in \mathcal{H}$ do
        if $L'(q, C) \geq U'$ then
            break                                   ▷ Revised Pruning Property 1
        for each element $e \in A$ of $C$ (let $o = e.o$) do
            if $q$ is not enclosed in cluster $C$ then
                $bound \leftarrow \lambda \cdot e.ds + (1 - \lambda) \cdot e.dt$
                if $d(q, C) - bound > U$ then
                    break                           ▷ Pruning Property 2
            if $d(q, o) < U$ then
                $\mathcal{R} \leftarrow \mathcal{R} \cup \{o\}$
                if $\mathcal{R}.size > k$ then
                    $\mathcal{R} \leftarrow \mathcal{R} \setminus \{o'\}, o' = \arg\max_{o_i \in \mathcal{R}} d(q, o_i)$
                if $\mathcal{R}.size = k$ then
                    $U \leftarrow \max_{o_i \in \mathcal{R}} d(q, o_i)$
                    $U' \leftarrow \max_{o_i \in \mathcal{R}} d'(q, o_i)$
    return $\mathcal{R}$
```

concretely, the cluster centroid C_i^t is computed by averaging all the m-dimensional projected data objects in O. The cluster radius R_i^t is also computed based on the projections, that is: $R_i^t = \max\{d_t'(o, C_i^t)\}$, where the distance function $d_t'()$ refers to the projected space.

Regarding the ordering of data objects within each cluster, we use the identical ordering as in CSSI. Thus, distances are computed in the original n-dimensional space. In summary, CSSIA organizes the data objects of a cluster in the same way as CSSI. The difference is in the way clusters are represented, namely based on the projected (m-dimensional) space. In turn, this affects the way pruning is performed, as indicated next.

5.3 Revisiting the Pruning Properties

Intra-cluster pruning (pruning property 2) is performed in the exact same way as in CSSI. This means that we use the distances in the original n-dimensional space. Regarding inter-cluster pruning, since the cluster is represented in m-dimensional space, we need to use the distances $d'()$ in the projected space to prune clusters. Therefore, we replace Equation 3 with the following one which defines a semantic lower bound in the projected space:

$$d_t'(q, o) \geq d_t'(q, C^t) - R^t = L'(q, C^t, R^t) \quad (5)$$

Based on this, we define $L'(q, C)$ in accordance with Equation 4, only replacing the distance $d_t()$ with $d_t'()$ as above. Therefore, we replace pruning property 1 as defined in Lemma 4.4 with the condition: $L'(q, C) \geq d'(q, o_{nn})$, where the right part of the inequality corresponds to $d'(q, o_{nn}) = \lambda \cdot d_s(q, o_{nn}) + (1 - \lambda) \cdot d_t'(q, o_{nn})$, but it is computed based on $d_t'()$ in the projected space. As a result, inter-cluster pruning is performed in the projected space.

5.4 Query Processing

Algorithm 3 describes the query processing procedure for CSSIA. To begin with, all hybrid clusters are ordered based on their lower bound $L'(q, C)$ with respect to q, in an ascending manner. We also set two types of bounds to infinity, one for each pruning property (U' for inter-cluster and U for intra-cluster pruning). Then, we iterate over each hybrid cluster C in H. If the lower bound for C is greater than the inter-cluster bound U', the cluster is pruned and none of its objects will be accessed. For each cluster that is not pruned, we access all of its objects from the outermost to the innermost ones. If q is not spatially covered by C, we check whether pruning property 2 (intra-cluster) applies for each accessed object in C. If that is not the case (i.e., $d(q, C) - bound \leq U$), we have to compute the distance between o and q. If $d(q, o)$ is less than the current intra-cluster upper bound U, then o is added to the list of neighbors. If that list grows larger than k, the furthest neighbor is discarded and both U and U' are updated.

6 COMPLEXITY AND INDEX MAINTENANCE

In this section, we analyze the space and time complexity of our algorithms (Sect. 6.1), and we discuss how index maintenance is performed for dynamic data (Sect. 6.2).

6.1 Space and Time Complexity

Space Complexity. The main factors are (a) the representations of the spatio-textual data objects (spatial coordinates and semantic vectors), (b) the representations of the hybrid clusters, and (c) the priority queue $\mathcal{R}$ which is of size k (typically $k \ll |O|$). Regarding the data objects, for a data set O of spatio-textual objects, we consider for each data object $o \in O$ its spatial 2D representation, the n-dimensional semantic representation and the distances of each object to the two centroids, therefore we need $(n + 4) \cdot |O|$ space.

For the $K = K_s \cdot K_t$ hybrid clusters in CSSI, we use $(n + 2)$ dimensions for the centroid plus the two radii, so $(n+4) \cdot K$ space. Assuming that K is much smaller than $|O|$, we have:

$$(n + 4) \cdot |O| + (n + 4) \cdot K + k = O(n \cdot |O|)$$

which means that space is linear to the size of the data set and the dimensionality of the semantic vectors. For CSSIA, we need additional $(m + 4) \cdot K$ space for each hybrid cluster. Also, we keep the low-dimensional projections of data objects, which is an additional $m \cdot |O|$. Despite that, the complexity is again $O(n \cdot |O|)$, since $m \ll n$.

Query Time Complexity. The cost is expressed as number of floating point operations. The query algorithm (a) sorts the hybrid clusters based on q, (b) computes distances between q and the subset of the objects that cannot be pruned, and (c) adds/removes objects to/from the priority queue $\mathcal{R}$. The cost of sorting the K hybrid clusters is $O(K \log K) \cdot (n + 2)$ for CSSI. The cost of computing the distances is in worst case $O(n \cdot |O|)$, since we have $|O|$ objects with $(n + 2)$ dimensions. Also, in worst case, all objects may be inserted to the priority queue, so additional $|O| \cdot \log k$. Thus, for CSSI:

$$K \log K \cdot (n + 2) + n \cdot |O| + |O| \cdot \log k = O(n \cdot (|O| + K \log K))$$

For CSSIA, the sorting cost is $K \log K \cdot (m + 2)$ and the cost of the priority queue is $|O| \cdot k$, because when the queue is updated we traverse all k objects to compute both $d(q, o_{nn})$ and $d'(q, o_{nn})$.

Indexing Time Complexity. The time complexity of index creation basically consists of the execution of K-Means (twice), deriving the semantic vector for each object, the execution of PCA, and the assignment of objects to hybrid clusters. K-means converges fast, and can in practice be considered linear for a small fixed number of iterations i and small number of clusters, i.e., $O(i \cdot K_s \cdot |O|)$ and $O(i \cdot K_t \cdot m \cdot |O|)$. Embeddings are stored in a lookup-table using word as key, ensuring fast access for

Alg.	Space/Time	Complexity
CSSI	Space (Index)	$O(n \cdot \lvert O \rvert)$
CSSIA	Space (Index)	$O(n \cdot \lvert O \rvert)$
CSSI	Time (Query)	$O((n + \log k) \cdot \lvert O \rvert + n \cdot K \log K)$
CSSIA	Time (Query)	$O((n + k) \cdot \lvert O \rvert + m \cdot K \log K)$
CSSI	Time (Index)	$O(n \cdot K \cdot \lvert O \rvert)$
CSSIA	Time (Index)	$O(n \cdot K \cdot \lvert O \rvert)$

Table 2: Space and time complexity of `CSSI` and `CSSIA`.

Parameter	Values
Twitter Data set size $\lvert O \rvert$	5M, 10M, 16M, 35M
Yelp Data set size $\lvert O \rvert$	0.5M, 1M, 2.5M, 5M
Dimensions of original space n	100
Dimensions of reduced space m	1, 2, 3, 5, 7, 9, 11, 13, 20, 30
Number of nearest neighbors k	5, 10, 25, 50, 100
Multiplier f	0.1, 0.3, 0.5, 0.7, 0.9
Balancing parameter λ	0 − 1, step: 0.1, def: 0.5

Table 3: Parameters ranges and their default values.

creating the semantic vector for a document, i.e., $O(|O|)$. The scikit-learn implementation of PCA makes use of a randomized SVD algorithm [20] that is very efficient when the number of principal components m is small, as in our case, i.e., $O(m \cdot n \cdot |O|)$. Finally, the assignment step has complexity $O(n \cdot K \cdot |O|)$ for `CSSI` and $O((m + n) \cdot K \cdot |O|)$ for `CSSIA`, but $m + n \approx n$. Putting all together, the dominant cost is $O(n \cdot K \cdot |O|)$. The results are summarized in Table 2.

6.2 Inserts, Deletes, Updates

In the case of dynamic data, our indexes are robust to object insertions, deletions or updates. Insertions correspond to the case where new data objects are added to the index, whereas deletions correspond to some objects becoming obsolete (e.g., a point of interest, such as a restaurant no longer operates). Updates typically involve a modification in the textual description of the objects, however it is also possible to have updates of the spatial coordinates.

Regarding insertions, a new data object o_{ins} needs to be assigned to a hybrid cluster. In turn, this means that o_{ins} needs to be assigned to both a spatial and a semantic cluster. In both cases, the selected cluster is the one with minimum distance (spatial or semantic) between o_{ins} and the cluster centroid C. In case the distance between C and o_{ins} is larger than the current radius R, we additionally update the radius (although that is not something that is expected to happen frequently). As a result, o_{ins} is assigned to a hybrid cluster.

Object deletions are handled by removing the object that needs to be deleted from the hybrid cluster that it belongs to. However, in the infrequent case that the deleted object o_{del} is the one that determines the cluster's radius (i.e., the farthest point from its cluster centroid), we need to update the cluster's radius. Again, this is common both for spatial as well as for semantic clusters. Finally, an update can be thought of as a deletion followed by an insertion.

The above approach to handle dynamic data is an incremental approach that avoids the cost of re-building the clusters for every update. For most object insertions/deletions/updates, especially when they follow the data distribution, this incremental process does not cause any performance degradation, since the clusters are not affected. However, after many object insertions/deletions/updates, it is possible that the data distribution has changed significantly, so in this case we need to rebuild the clusters.

7 EXPERIMENTAL EVALUATION

In this section, we evaluate the performance of our algorithms `CSSI`/`CSSIA` using fairly large, real-world data sets.

7.1 Experimental Setup

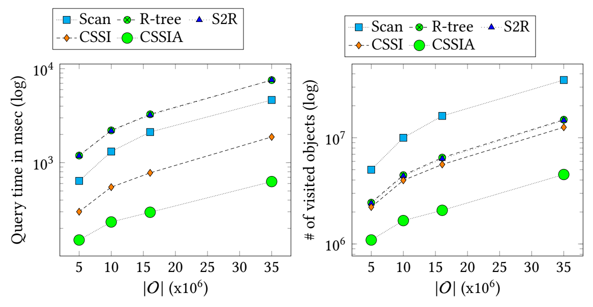
Code and Platform. Our algorithms are memory-resident and implemented in C++ using g++ ver. 11.4.0, while data set pre-processing is implemented in Python3, whereas index creation in Rust. Our code is publicly available[3]. The implementations of the metric indexes comes from the authors of [48].[4] The experiments are run on a server with dual 12-core 2.30GHz Intel Xeon Gold 5118 CPUs, with 128GB RAM, using Ubuntu 22.04.3.

Algorithms. We examine the efficiency and quality of five indexing methods, including the two methods that we propose (`CSSI` and `CSSIA`). The first one is a linear scan algorithm (denoted Scan) that calculates the distances between the query and all the objects in the data set, included because in high dimensions linear scan often outperforms index-based algorithms. The second one is an index-based algorithm (denoted R-tree) that builds an R-tree on the spatial coordinates of each object, with semantic vectors located in the leaves, practically a spatial-only index. It performs a best-first k-NN search [22] where *mindist* is calculated based on the assumption that in worst case semantic distance is zero, as there can always be a semantic vector with semantic distance equal to zero located in some non-visited leaf node. Moreover, we compare against the state-of-the-art algorithm [8] (denoted S2R). Last, but not least, we compare against two approaches for multi-metric indexing, DESIRE [48] and RR*-tree [16].

Data sets and Index creation. We use two data sets: Twitter and Yelp. The former consists of geo-tagged tweets from the US, written in English, that were collected using the public Twitter API[5]. The location is given by metadata attached to the tweet, provided by the API. The Yelp data set[6] consists of user reviews from 11 metropolitan areas, all reviews being for a particular business (e.g., cafe, restaurant, hotel, shop) with known location. Of the approx. 7M reviews in the data set, we use 5M. Notice that our largest data set is approximately two orders of magnitude larger than what is used in related work [8, 35]. In both data sets, the spatial coordinates are normalized in $[0, 1] \times [0, 1]$.

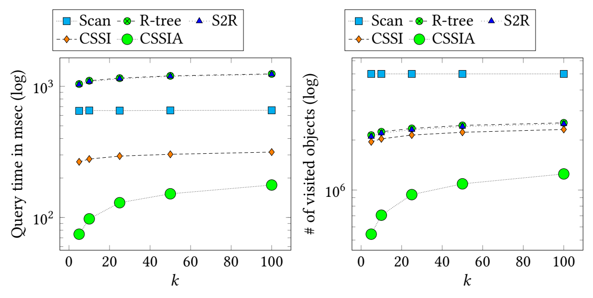
Regarding pre-processing and index creation, the textual part of each tweet or Yelp review is embedded in a n-dimensional space (also mentioned as original space) and is represented as a semantic vector. We use pre-trained vectors by the authors of Glove [32] with n=100, which is the dimensionality used in [8] too. To produce a single semantic vector per object, we average the embedding of each word in the tweet or review. Terms that do not exist in the vocabulary and common stop-words are dropped. Every tweet or review that includes fewer than 3 words is also dropped. With respect to clustering, we apply K-Means to 10% of the data set to obtain centroids and radii (using a 10% sample

[3]https://github.com/noervaag/CSSI
[4]https://github.com/ZJU-DAILY/DESIRE
[5]https://developer.twitter.com/en/docs/twitter-api
[6]https://www.yelp.com/dataset



(a) Query time. **(b) Visited objects.**

Figure 5: Scalability (Twitter).



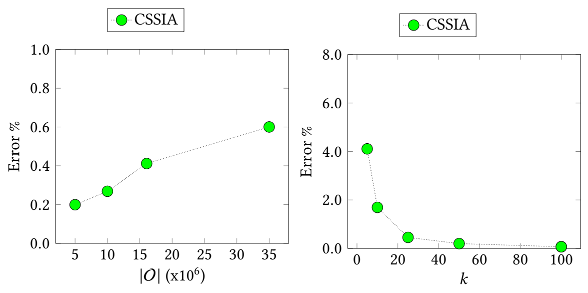
(a) Query time. **(b) Visited objects.**

Figure 6: Varying k (Twitter).

gave similar results as using the whole data set). Then, we assign the remaining 90% of the objects to the nearest cluster, updating both centroids and radii, if needed. For the computation of PCA, we use the randomized SVD provided in scikit-learn [31] (based on the method of Halko et al. [19]), which is very efficient when the number of principal components is much smaller than data set size, as in our case.

Parameters. One significant parameter is the number of clusters for a given data set. Having too many clusters induces overhead for data management of their descriptions, whereas too few clusters have a negative effect on pruning. In our experiments, we set the number of clusters based on the formula: $K_s = K_t = \sqrt{|O| \cdot 0.01} \cdot f$, which was shown experimentally to produce a reasonable number of clusters for different values $|O|$ (e.g., for the default setup, this gives a total of 4,489 hybrid clusters). The multiplier f can be used to further control the number of clusters, and we evaluate different values of f. It is important to note that our goal is not to find the optimal number of clusters for each field, but to introduce a parameter that can increase or decrease the granularity of the underlying partitioning that is obtained. We vary the size of the data set $|O|$, the projection dimensionality m, the value of k-NN, and the parameter λ of the distance function. Table 3 lists the parameters.

Metrics. The main metric is query execution time. Also, we report the number of visited (accessed) objects by each algorithm, which indicates how well pruning is performed. For our algorithms CSSI/CSSIA, we additionally report the number of objects pruned during intra- and inter-cluster processing. Also, we report the error rate of CSSIA measured as the number of objects in the result set of the exact algorithm(s) that are not in the result set of the approximate algorithm, divided by the result set size k.



(a) Varying $|O|$. **(b) Varying k.**

Figure 7: Error of CSSIA (Twitter).

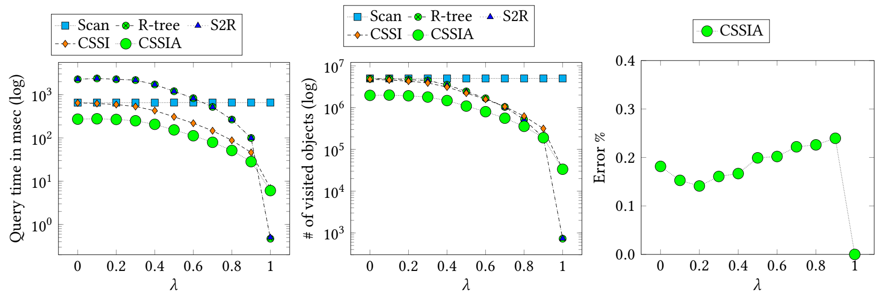
Queries. For each experiment, we randomly select 100 objects from the data set as queries and we report the average values. For the error measurements of CSSIA, we use 5,000 queries because of the error rate being very low.

7.2 Comparative Evaluation on Twitter Data

Fig. 5 shows the scalability of the algorithms with data set size, from 5M to 35M objects. In Fig. 5a, the query time is depicted in log scale, whereas the number of visited (accessed) objects by each algorithm is shown in Fig. 5b. Overall, CSSIA is shown to be significantly faster than CSSI, typically x2–x3 times faster, and this gain increases for larger data sets (notice the log scale). CSSIA owes its superior performance to the better pruning, as it needs to access the fewest data objects. Scan accesses all data objects, followed by the index-based algorithms: R-tree and S2R, which also access a large number of objects. However, R-tree is slower than Scan, because of the overhead for traversing the index structure and less efficient memory access (the tree-based index-structures do not benefit from efficient prefetching from memory to the extent that linear scan can do). Interestingly, S2R also performs as bad as R-tree, as it accesses almost the same number of objects. Despite its enhancement of index nodes with semantic bounding boxes, S2R cannot prune large portions of the index, as discussed in Sect. 2. The reason is that it follows a spatial-first approach, organizing index nodes based on spatial coordinates and then adds to them semantic bounding boxes which may cover the entire space, as there may exist no correlation of spatial location with semantic vectors. As can be seen also in [8], the difference between S2R and R-tree decreases with increasing cardinality, and for our significantly larger data sets (the largest data set used in [8] contains only 250K objects), the difference in performance is only marginal.

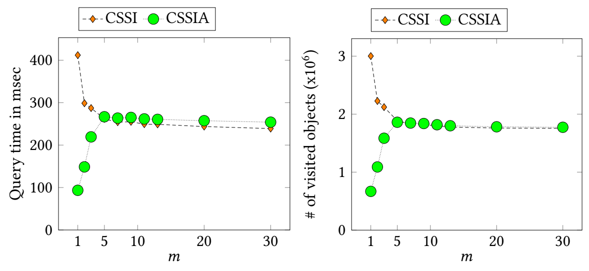
Fig. 6 shows the effect of the value of k, the number of retrieved nearest neighbors. Increasing values of k after a certain point have a minor effect on query time and visited objects, therefore all lines become almost straight for $k > 50$. Notice that for low values of k, which are typical in most applications, CSSIA achieves a significant performance gain, by accessing much fewer data objects.

Fig. 7 shows the error of CSSIA when varying $|O|$ (Fig. 7a) and k (Fig. 7b). For clarity, if the algorithm retrieves $k-1$ correct nearest neighbors and 1 false neighbor, the error should be $\frac{1}{k} \cdot$ 100%. For the default value of $k = 50$, having 1 false results in 50 would mean 2% error. As will be shown, the error of CSSIA is typically much lower. In particular, in Fig. 7a, for all data set sizes, the error is always smaller than 1%, indicating that for most of the queries CSSIA returns the correct result and only misses



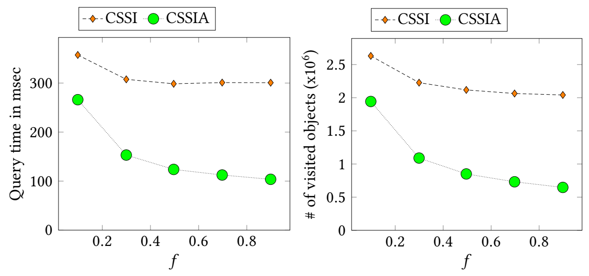
(a) Query time. **(b) Visited objects.** **(c) Error.**

Figure 8: Varying λ (Twitter).



(a) Query time. **(b) Visited objects.**

Figure 9: Varying m (Twitter).



(a) Query time. **(b) Visited objects.**

Figure 10: Varying f (Twitter).

one object in few queries. Even for small values of k (Fig. 7b), the error is at most 4%.

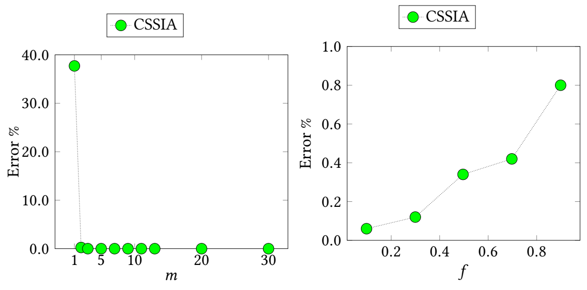
Fig. 8 shows the effect of the balancing factor λ. Recall that for $\lambda = 0$ the problem becomes equivalent to k-NN search of high-dimensional vectors, while for $\lambda = 1$ it is reduced to spatial k-NN. Interestingly, for small values of λ, our algorithms outperform the competitors. Obviously, Scan is not affected by λ and appears as a straight line. In particular, the index-based methods exhibit worse performance than Scan (as both S2R and R-tree rely on spatial indexing or spatial-first indexing). For high values of λ our algorithms and the index-based solutions have similar behavior. It is also noteworthy that only for $\lambda > 0.7$ do the index-based solutions outperform Scan. Regarding the error of CSSIA, this is always smaller than 0.3% and for the special case of spatial k-NN the error is equal to zero, indicating that CSSIA returns the correct result.

7.3 Sensitivity Analysis using Twitter Data

Fig. 9 shows the effect of the projection dimensionality m on both CSSI and CSSIA. Our first observation is that CSSI performs better for values of m up to 10, where its performance is stabilized. This is explained by the fact that the clustering used by CSSI produces better clusters for $m > 2$, and this affects the performance of query processing. Also, it is worth pointing out that for small values of m, i.e., smaller than 5, CSSIA is faster than CSSI, while for larger values their performance becomes comparable. For low values of m, CSSIA prunes significantly more objects than CSSI, which verifies the intuition of Sect. 5.1. Also, for $m = 2-5$, it is evident that pruning gets worse for CSSIA, and the lines for both algorithms converge for $m = 5$. That is the case because the more principal components that are used, the larger the projected space gets and, thus, the variance of the distances in that space approaches the relatively low variance value of the n-dimensional one.

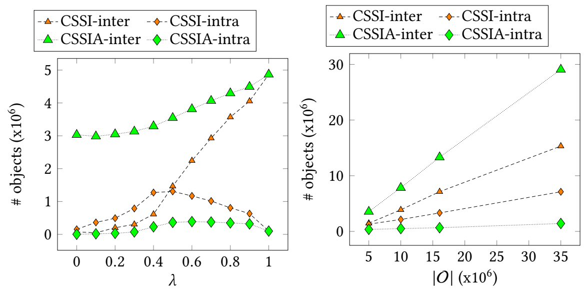
Fig. 10 presents the effect of the parameter f, which determines the number of hybrid clusters. Using larger values for f results in more hybrid clusters. In principle, having more clusters leads to more fine-grained and compact clusters, which leads to better pruning and improved performance. This holds as long as the number of clusters does not exceed a specific number, i.e., in case of too many clusters this may have a negative effect on performance due to (a) the cost of sorting, and (b) the extra cost associated with examining these clusters. CSSI does not improve for larger values of f because after some point its pruning does not improve. Eventually the larger overhead outweighs the negligible gain from pruning and CSSI becomes slower. Instead, CSSIA improves its pruning even for the largest values of f, thus improving its performance. These findings are justified by inspecting the number of visited objects.

Fig. 11 shows the error of CSSIA when increasing m and f. Regarding the value of projection dimensionality m, the error is large (close to 40%) only for the special case of $m = 1$. This indicates that projecting to one-dimensional values is insufficient for building a qualitative index. For $m > 2$, the error of CSSIA is very small (smaller than 1%). Combined with the result of Fig. 9a, it seems that using $m = 2$ is the best choice for CSSIA, as it results in low processing time and low error. Fig. 11b shows that for all values of f the error is smaller than 0.8%. Larger values of f result in higher number of clusters and this seems to increase the error of CSSIA. The rationale behind the default value $f = 0.3$ is that it offers the best performance for CSSI, while CSSIA also performs well enough.



(a) Varying m. (b) Varying f.

Figure 11: Error of CSSIA (Twitter).



(a) Varying λ. (b) Varying $|O|$.

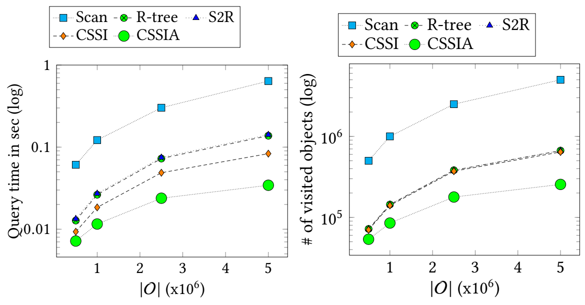
Figure 12: Pruning (Twitter).

Fig. 12 reveals how well the two pruning properties of our algorithms perform. For each algorithm (say CSSI), we report the number of pruned objects due to intra-cluster pruning (CSSI-intra) and due to whole clusters being pruned (CSSI-inter). By adding these two numbers and number of visited objects (Fig. 5b and Fig. 8b), we always get back $|O|$, the cardinality of the data set. All figures show that CSSIA relies a lot more on inter-cluster pruning (i.e., prunes entire clusters more frequently) than CSSI. In contrast, in CSSI both inter-cluster and intra-cluster pruning seem to contribute equally to the cumulative pruning achieved.

7.4 Comparative Evaluation on Yelp Data

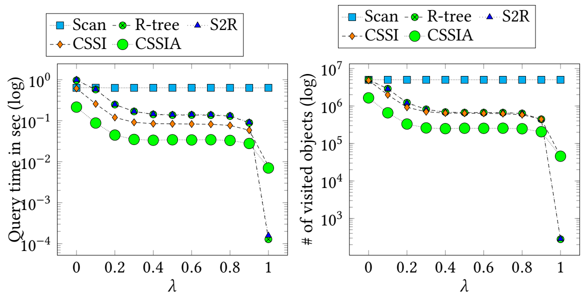
Fig. 13 shows the performance of all algorithms when increasing the size of the Yelp data set from 500K to 5M data objects. Notice the use of log-scale on the y-axis. Again, our algorithms outperform all competitors. However, an interesting observation is that the index-based algorithms (R-tree and S2R) outperform the Scan algorithm, whereas the opposite was observed in the case of Twitter data. This is attributed to the fact that in Yelp the spatial locations form dense clusters in some cities, and this favors the algorithms that follow a spatial-first indexing approach. This shows that the index-based algorithms, including the state-of-the-art (S2R), outperform Scan for certain data sets only (when there is strong spatial clustering). In contrast, our algorithms rely on hybrid clusters, thus always outperforming the competitors (both Scan and index-based).

Fig. 14 shows how changing the value of λ affects the performance of query processing. Interestingly, for reasonable values of λ (i.e., larger than 0 and smaller than 1) our algorithms outperform all competitors. For $\lambda = 1$, the problem is reduced to spatial k-NN and combined with the strong spatial clustering of the Yelp data set, this results in better performance for the



(a) Query time. (b) Visited objects.

Figure 13: Scalability (Yelp).



(a) Query time. (b) Visited objects.

Figure 14: Varying λ (Yelp).

index-based algorithms. On the other extreme, for $\lambda = 0$, CSSIA is the best-performing algorithm, while CSSI behaves similarly as Scan, and the index-based algorithms are worse than Scan. Finally, in the case of Yelp, the error of CSSIA is always smaller than 0.2%.

7.5 Cost of Index Creation

The indexing cost for both CSSI and CSSIA is almost identical and can be split into: the cost of PCA, the cost of K-Means, and the formation of hybrid clusters.

Fig. 15 shows the time for index creation using a sample size of 10% for different sizes ($|O|$) of the Twitter data set. Even when $|O|$=35M, indexing as a whole only takes around 4, 500 seconds (approx. 75 minutes). The two most time-consuming parts, K-Means and the formation of hybrid clusters, can easily be parallelized, thus significantly reducing the cost. Note that the reason behind non-linear scaling in the chart is that we increase the number of clusters with data set size, according to the formula in Sect. 7.1.

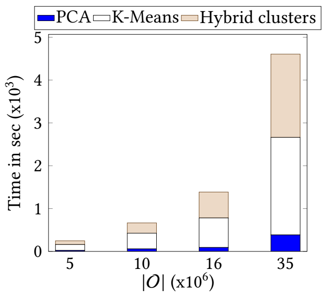


Figure 15: Index creation (Twitter).

7.6 Effect of Inserts and Updates

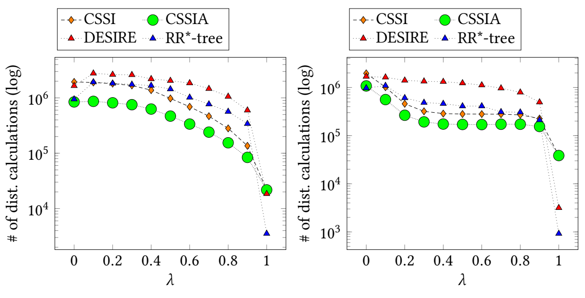
In order to study the efficiency of index after inserts, we compare query performance: 1) using an index created based on the whole

	10M	15M	20M	35M
CSSI-Full	3,926,072	5,523,278	7,342,543	12,565,492
CSSI-Partial	3,938,545	5,565,072	7,370,274	12,673,968
CSSI Increase	0.318%	0.757%	0.378%	0.863%
CSSIA-Full	1,301,884	1,825,084	2,570,055	4,485,555
CSSIA-Partial	1,334,685	1,857,023	2,601,536	4,658,100
CSSIA Increase	2.520%	1.750 %	1.225%	3.847%

Table 4: Effect of inserts on index efficiency.

# updates:	0	0.5M	1.5M	2.5M
CSSI (# objects)	2,011,321	2,008,944	2,056,239	2,003,551
CSSIA (# objects)	852,628	871,090	898,889	887,690
CSSIA (Error %)	0.27	0.26	0.26	0.26

Table 5: Effect of updates on index efficiency.



(a) Twitter. **(b) Yelp.**

Figure 16: Comparison to multi-metric indexing.

data set as described in Sect. 4.1 (Full) vs. 2) an index created based on 5M objects, and the remainder objects (up to 35M) inserted afterwards as described in Sect. 6.2 (Partial). Table 4 shows the number of visited objects for both cases, for various data set sizes, for both CSSI and CSSIA. As can be observed, the increase in query cost is only marginal even after a high number of inserts, indicating the index is resilient to subsequent inserts.

To study the efficiency of index after updates, we compare query performance after a certain number of updates (delete followed by insert, as described in Sect. 6.2, thus keeping data set size constant). Table 5 shows the number of visited objects and the error rate after a number of update operations, on a data set of size $|O|$=5M. As can be observed, the query cost and error rate remain almost unchanged. This shows that the approach is robust to inserts/deletes.

7.7 Comparison to Multi-metric Indexing

Fig. 16 presents a comparison against state-of-the-art methods for multi-metric indexing: RR*-tree [16] and DESIRE [48]. We used data sets of 1M objects, as this was the largest that could be run with the implementation of the competitors. We measure the number of distance calculations as an objective measure, mainly because DESIRE operates in secondary storage, and we vary the parameter λ. We included the RR*-tree as it has been shown in [48] to need fewer distance calculations than DESIRE. For DESIRE and the RR*-tree, the number of distance calculations is the sum of distances calculated for each metric space. For CSSI, the number of distance calculations is number of objects visited

	Avg. Cluster Diameter	Time (in sec)
K-means	0.310	24.66
HAC (Ward)	0.323	210.18
HAC (Complete)	0.321	194.62

Table 6: Effect of clustering.

multiplied by two (one distance calculation for spatial space, one for semantic space).

As can be seen, our algorithms are consistently better than both competitors. The main reason is that DESIRE performs a k-NN in a single metric space, and then uses the radius of the k-th object to perform a range query over the other metric space (to ensure correctness), which is inefficient compared to our hybrid approach. Interestingly, only for the corner case of $\lambda = 1$, i.e., spatial-only k-NN, the competitors are better than our algorithms.

7.8 Effect of Clustering

Finally, we performed an experiment to demonstrate the suitability of K-means as the clustering method for our approach. The objective is to have a scalable and efficient method that produces compact spherical clusters, i.e., having low diameter. To this end, we compare against two versions of hierarchical agglomerative clustering (HAC), namely *Ward* and *Complete or Max-link*. In this experiment, we had to use a smaller sample of the data set, i.e., 1%, since HAC run out of memory for our default value. We evaluate the average diameter of the clusters, as a measure of quality, and the time needed to perform the clustering. In Table 6, we show the results for the default setup ($f = 0.3$). We observe that K-means produces slightly more compact clusters, i.e., having lower diameter on average, and almost one order of magnitude faster than HAC.

8 CONCLUSIONS AND FUTURE WORK

In this paper, we presented a novel approach for semantic similarity search over spatio-textual data. The gist of our approach is the construction of hybrid clusters, which combine spatial with semantic clusters, that constitute our indexing structure. We proposed two query processing algorithms, CSSI which is provably correct and outperforms the state-of-the-art, and CSSIA which boosts the performance of query processing at the expense of slightly inaccurate results. Our algorithms outperform the competitors using two real-world, large-scale data sets.

In our future work, we intend to explore other query types that combine spatial with semantic retrieval and can exploit our indexing based on the hybrid clusters. Moreover, making our approach applicable in a big data setting using parallel processing algorithms is a challenging research direction.

ACKNOWLEDGMENTS

The research work was supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the "First Call for H.F.R.I. Research Projects to support Faculty members and Researchers and the procurement of high-cost research equipment grant" (Project Number: HFRI-FM17-81), and by the Horizon Europe R&I programme EMERALDS under the GA No. 101093051.

The authors would like to thank the anonymous reviewers for their fruitful comments that helped in clarifying the technical details of our approach.

REFERENCES

[1] Kevin S. Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. 1999. When is "nearest neighbor" meaningful?. In *Proceedings of ICDT*. 217–235.

[2] Benjamin Bustos, Sebastian Kreft, and Tomás Skopal. 2012. Adapting metric indexes for searching in multi-metric spaces. *Multim. Tools Appl.* 58, 3 (2012), 467–496.

[3] Xin Cao, Gao Cong, Tao Guo, Christian S. Jensen, and Beng Chin Ooi. 2015. Efficient processing of spatial group keyword queries. *ACM Trans. Database Syst.* 40, 2 (2015), 13:1–13:48.

[4] Xin Cao, Gao Cong, Christian S. Jensen, and Beng Chin Ooi. 2011. Collective spatial keyword querying. In *Proceedings of SIGMOD*. 373–384.

[5] Jing Chen, Jiajie Xu, Chengfei Liu, Zhixu Li, An Liu, and Zhiming Ding. 2017. Multi-objective spatial keyword query with semantics. In *Proceedings of DASFAA*. 34–48.

[6] Lisi Chen, Gao Cong, Christian S. Jensen, and Dingming Wu. 2013. Spatial keyword query processing: An experimental evaluation. *PVLDB* 6, 3 (2013), 217–228.

[7] Lisi Chen, Shuo Shang, Chengcheng Yang, and Jing Li. 2019. Spatial keyword search: A survey. *Geoinformatica* (2019).

[8] Xinyu Chen, Jiajie Xu, Rui Zhou, Pengpeng Zhao, Chengfei Liu, Junhua Fang, and Lei Zhao. 2020. S^{2}R-tree: a pivot-based indexing structure for semantic-aware spatial keyword search. *GeoInformatica* 24, 1 (2020), 3–25.

[9] Zhida Chen, Lisi Chen, Gao Cong, and Christian S. Jensen. 2021. Location- and keyword-based querying of geo-textual data: A survey. *VLDB J.* 30, 4 (2021), 603–640.

[10] Paolo Ciaccia and Marco Patella. 2000. The M^2-tree: Processing Complex Multi-Feature Queries with Just One Index. In *Proceedings of DELOS Workshop (ERCIM Workshop Proceedings, Vol. 01/W001)*. ERCIM.

[11] Gao Cong and Christian S. Jensen. 2019. Spatio-textual data. In *Encyclopedia of Big Data Technologies*, Sherif Sakr and Albert Y. Zomaya (Eds.). Springer.

[12] Gao Cong, Christian S. Jensen, and Dingming Wu. 2009. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB* 2, 1 (2009), 337–348.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*. 4171–4186.

[14] Ronald Fagin, Amnon Lotem, and Moni Naor. 2001. Optimal Aggregation Algorithms for Middleware. In *Proceedings of PODS*.

[15] Ian De Felipe, Vagelis Hristidis, and Naphtali Rishe. 2008. Keyword search on spatial databases. In *Proceedings of ICDE*. 656–665.

[16] Maximilian Franzke, Tobias Emrich, Andreas Züfle, and Matthias Renz. 2016. Indexing multi-metric data. In *Proceedings of ICDE*. 1122–1133.

[17] Tao Guo, Xin Cao, and Gao Cong. 2015. Efficient algorithms for answering the m-closest keywords query. In *Proceedings of SIGMOD*. 405–418.

[18] Tao Guo, Kaiyu Feng, Gao Cong, and Zhifeng Bao. 2018. Efficient selection of geospatial data on maps for interactive and visualized exploration. In *Proceedings of SIGMOD*. 567–582.

[19] Nathan Halko, Per-Gunnar Martinsson, Yoel Shkolnisky, and Mark Tygert. 2011. An Algorithm for the Principal Component Analysis of Large Data Sets. *SIAM J. Sci. Comput.* 33, 5 (2011), 2580–2594.

[20] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. 2011. Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. *SIAM Rev.* 53, 2 (2011), 217–288.

[21] Alexander Hinneburg, Charu C. Aggarwal, and Daniel A. Keim. 2000. What is the nearest neighbor in high dimensional spaces?. In *Proceedings of VLDB*. 506–515.

[22] Gísli R. Hjaltason and Hanan Samet. 1999. Distance Browsing in Spatial Databases. *ACM Trans. Database Syst.* 24, 2 (1999), 265–318.

[23] H. V. Jagadish, Beng Chin Ooi, Kian-Lee Tan, Cui Yu, and Rui Zhang. 2005. iDistance: An adaptive B$^+$-tree based indexing method for nearest neighbor search. *ACM Trans. Database Syst.* 30, 2 (2005), 364–397.

[24] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.

[25] Georgios Kalamatianos, Georgios John Fakas, and Nikos Mamoulis. 2021. Proportionality in spatial keyword search. In *Proceedings of SIGMOD*. 885–897.

[26] Zhisheng Li, Ken C. K. Lee, Baihua Zheng, Wang-Chien Lee, Dik Lun Lee, and Xufa Wang. 2011. IR-Tree: An efficient index for geographic document search. *IEEE Trans. Knowl. Data Eng.* 23, 4 (2011), 585–599.

[27] Ahmed R. Mahmood, Ahmed M. Aly, and Walid G. Aref. 2018. FAST: Frequency-aware indexing for spatio-textual data streams. In *Proceedings of ICDE*. 305–316.

[28] Ahmed R. Mahmood and Walid G. Aref. 2019. *Scalable processing of spatial-keyword queries*. Morgan & Claypool Publishers.

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

[30] Tomás Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of NIPS*. 3111–3119.

[31] Fabian Pedregosa et al. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[32] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of EMNLP*. 1532–1543.

[33] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of NAACL-HLT*. Association for Computational Linguistics, 2227–2237.

[34] Zhihu Qian, Jiajie Xu, Kai Zheng, Wei Sun, Zhixu Li, and Haoming Guo. 2016. On efficient spatial keyword querying with semantics. In *Proceedings of DASFAA*. 149–164.

[35] Zhihu Qian, Jiajie Xu, Kai Zheng, Pengpeng Zhao, and Xiaofang Zhou. 2018. Semantic-aware top-k spatial keyword queries. *World Wide Web* 21, 3 (2018), 573–594.

[36] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings EMNLP-IJCNLP*.

[37] João B. Rocha-Junior, Orestis Gkorgkas, Simon Jonassen, and Kjetil Nørvåg. 2011. Efficient processing of top-k spatial keyword queries. In *Proceedings of SSTD*. 205–222.

[38] Yufan Sheng, Xin Cao, Yixiang Fang, Kaiqi Zhao, Jianzhong Qi, Gao Cong, and Wenjie Zhang. 2023. WISK: A Workload-aware Learned Index for Spatial Keyword Queries. In *Proceedings of SIGMOD*.

[39] Jiabao Sun, Jiajie Xu, Kai Zheng, and Chengfei Liu. 2017. Interactive spatial keyword querying with semantics. In *Proceedings of CIKM*. 1727–1736.

[40] Panagiotis Tampakis, Dimitris Spyrellis, Christos Doulkeridis, Nikos Pelekis, Christos Kalyvas, and Akrivi Vlachou. 2021. A novel indexing method for spatial-keyword range queries. In *Proceedings of SSTD*. 54–63.

[41] Yao Tian, Tingyun Yan, Xi Zhao, Kai Huang, and Xiaofang Zhou. 2023. A Learned Index for Exact Similarity Search in Metric Spaces. *IEEE Trans. Knowl. Data Eng.* 35, 8 (2023), 7624–7638.

[42] Subodh Vaid, Christopher B. Jones, Hideo Joho, and Mark Sanderson. 2005. Spatio-textual indexing for geographical search on the Web. In *Proceedings of SSTD*. 218–235.

[43] Xiang Wang, Ying Zhang, Wenjie Zhang, Xuemin Lin, and Wei Wang. 2015. AP-Tree: Efficiently support continuous spatial-keyword queries over stream. In *Proceedings of ICDE*. 1107–1118.

[44] Dingming Wu, Gao Cong, and Christian S. Jensen. 2012. A framework for efficient spatial web object retrieval. *VLDB J.* 21, 6 (2012), 797–822.

[45] Jiajie Xu, Jing Chen, and Lihua Yin. 2020. Multi-objective spatial keyword query with semantics: a distance-owner based approach. *Distributed and Parallel Databases* 38 (2020), 625–647.

[46] Junye Yang, Yong Zhang, Xiaofang Zhou, Jin Wang, Huiqi Hu, and Chunxiao Xing. 2019. A Hierarchical Framework for Top-k Location-Aware Error-Tolerant Keyword Search. In *Proceedings of ICDE*. 986–997.

[47] Yong Zhang, Yu Chen, Junye Yang, Jin Wang, Huiqi Hu, Chunxiao Xing, and Xiaofang Zhou. 2021. Clustering Enhanced Error-tolerant Top-k Spatio-textual Search. *World Wide Web* 24, 4 (2021), 1185–1214.

[48] Yifan Zhu, Lu Chen, Yunjun Gao, Baihua Zheng, and Pengfei Wang. 2022. DESIRE: An Efficient Dynamic Cluster-based Forest Indexing for Similarity Search in Multi-Metric Spaces. *Proc. VLDB Endow.* 15, 10 (2022), 2121–2133.