Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Tore Apeland Fossland

Semantic Similarity Search over Spatio-textual Data

Master's thesis in Computer Science
Supervisor: Kjetil Nørvåg
June 2023

NTNU
Norwegian University of
Science and Technology

Tore Apeland Fossland

Semantic Similarity Search over Spatio-textual Data

Master's thesis in Computer Science
Supervisor: Kjetil Nørvåg
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

This master's thesis explores the challenges associated with indexing and searching spatio-textual data, a rapidly expanding category of multi-dimensional objects with diverse applications. Existing methods for spatio-textual data search often struggle to capture the intricate semantic relationships within textual data. To overcome this limitation, this thesis presents DualiDistance, a novel algorithm that leverages *word embeddings* to enhance the accuracy and effectiveness of search operations. Building upon the foundational principles of the iDistance indexing method, DualiDistance integrates spatial and textual information into dual indexes to tackle some of the principal challenges in semantic similarity searches. The approach goes beyond exact term matching by incorporating semantic relevance and tries to avoid a spatial bias, which can result in more accurate search results.

Sammendrag

Denne masteroppgaven utforsker utfordringene knyttet til indeksering og søk i spatio-tekstuelle data, en raskt voksende kategori av flerdimensjonale data med varierte bruksområder. Eksisterende metoder for søk over spatio-tekstuell data sliter ofte med å fange de intrikate semantiske sammenhengende i tekstdata. For å unngå denne begrensningen presenterer denne oppgaven DualiDistance, en ny algoritme som tar i bruk *word embeddings* for å forbedre nøyaktigheten og effektiviteten av søkeoperasjoner i tekstdata. Ved å bygge videre på de grunnleggende prinsippene til indekseringsmetoden iDistance, fletter DualiDistance sammen spatial og tekstuell informasjon i doble indekser for å unngå en spatial bias som man ser hos andre relaterte metoder.

Aknowledgements

We would like to thank to our thesis advisor, Kjetil Nørvåg, for their guidance and for providing access to their code repository and datasets. Their code repository served as a great starting point for our work as well as providing benchmarking capabilities when testing our algorithms.

Tore Apeland Fossland
Trondheim, 12th June 2023

Contents

Acronyms

ANN Approximate Nearest Neighbors.

BERT Bidirectional Encoder Representations from Transformers.

ELMo Embeddings from Language Models.

FA Fagin's Algorithm.

GloVe Global Vectors for Word Representation.

I/O Input/Output.

KNN k-Nearest Neighbors.

LDA Latent Dirichlet Allocation.

LSH Locality-Sensitive Hashing.

MBB Minimum Bounding Boxes.

MBR Minimum Bounding Rectangles.

NN Nearest Neighbor.

PCA Principal Component Analysis.

t-SNE t-Distributed Stochastic Neighbor Embedding.

TA Threshold Algorithm.

1 Introduction

This master's thesis focuses on semantic similarity search over spatio-textual data and the underlying theory that enables them. This chapter presents the motivation and objectives of this research. We will begin by providing background information and explaining the motivation for undertaking this work. Next, we will define the problem formulation. Finally, we will define the goals and research questions that will guide the work and outline the structure of the thesis.

1.1 Background

Today we are witnessing a tremendous increase in the sheer amount, diversity, and speed at which digital information is being generated. A substantial portion of the available data consists of multidimensional information, encompassing various types of data such as spatial coordinates, textual content, and temporal data. Among these, the intersection of spatial and textual data, known as spatio-textual data, has drawn considerable attention due to its broad range of applications. From geotagged social media posts to location-based service reviews, spatio-textual data is rich in contextual information that provides insights into users' behaviors, preferences, and sentiments. Thus, efficient search methods for this type of data are crucial for many applications. The integration of spatial and textual information introduces numerous challenges for similarity search. Traditional methods for spatio-textual search often fall short of capturing the semantic relationships within textual data. These methods primarily rely on keyword-based search, which may not consider the nuanced similarities between words and can lead to inadequate search results.

1.2 Motivation

In this thesis, our motivation is to address the challenges of semantic similarity search over spatio-textual data. We aim to capture the semantic relationships between words in a high-dimensional space by utilizing word embeddings, enabling more contextually accurate and comprehensive search capabilities compared to traditional keyword-based methods. Word embeddings allow us to represent the textual description of a spatio-textual object as a dense, high-dimensional semantic vector. By leveraging similarity functions on these vectors, we can identify semantically similar textual descriptions, leading to more accurate and contextually sensitive search results. The main challenges we face are the high dimensionality of the semantic vectors and the integration of spatial

and textual information. The high-dimensional nature of spatio-textual data poses computational and efficiency challenges for similarity search. Additionally, effectively incorporating both spatial proximity and semantic textual similarity into a unified index adds another layer of complexity. To tackle these challenges, we introduce the novel algorithm DualiDistance. This algorithm is inspired by the iDistance indexing method, originally designed for high-dimensional similarity search. We extend the principles of iDistance into the dual spaces of spatial and textual data, allowing us to partition and index the spatio-textual data space separately. By doing so, we aim to enable faster retrieval of relevant data objects during query processing, providing an efficient and effective dual search mechanism.

1.3 Problem Formulation

This master's thesis addresses the issue of semantic similarity search over spatio-textual data. The task involves selecting, from a set of spatio-textual objects $P = \{p_1, p_2, ..., p_n\}$ each possessing a geographical location $(p_i.x, p_i.y)$ and a textual annotation $p_i.text$, the k objects that minimize the distance function $d(q, p)$, considering both semantic and spatial aspects. The distance function $d(q, p)$ embodies two key elements. The first one is $d_s(q, p)$, which measures the spatial distance between the coordinates of the query $(q.x, q.y)$ and those of a particular object $(p.x, p.y)$. The calculation of $d_s(q, p)$ employs a normalized form of the Euclidean distance, normalized by D_{max_s}, the greatest Euclidean distance between any two items in the data set

$$d_s(q, p) = \frac{\sqrt{(q.x - p.x)^2 + (q.y - p.y)^2}}{D_{max_s}}$$

The second element is the semantic distance $d_t(q, p)$, derived from the comparison between $q.text$ and $p_i.text$. Semantic distance is defined using word embeddings, which allow us to transform the textual description of a spatio-textual object p into a highly-descriptive n-dimensional vector $V = \{v[1], v[2], \ldots, v[n]\}$. Semantic distance $d_t(q, p)$ is determined as the Euclidean distance between the normalized semantic vectors of q and p, normalized by the largest potential Euclidean distance D_{max_t}

$$d_t(q, p) = \frac{\sqrt{(V_q[1] - V_p[1])^2 + (V_q[2] - V_p[2])^2 + \ldots + (V_q[n] - V_p[n])^2}}{D_{max_t}}$$

We propose the following equation for the total weighted distance function for two spatio-textual objects q and p

$$d(q, p) = \lambda \cdot d_s(q, p) + (1 - \lambda) \cdot d_t(q, p) \tag{1}$$

Here, $\lambda \in [0, 1]$ is an application-specific variable that adjusts the relative impact of the spatial and semantic components on the overall distance function.

Problem 1. (Semantic Spatio-textual Similarity Search) Given a set of spatio-textual objects $\{p_i\} \in P$ and a query object q, the objective is to find the k entities $P_k = \{p_1, \ldots, p_k\}$ such that

$$d(q, p_i) \leq d(q, p_j), \quad \forall p_i \in P_k \text{ and } \forall p_j \in P - P_k.$$

where $d(q, p)$ represents the distance between objects q and p. We introduce an exact algorithm to address the k-nearest neighbors problem above in Section 4.4.

1.4 Goals and Research Questions

Goal *Develop indexing and searching methods that improve the performance of similarity searches in spatio-textual data.*

The primary objective of this thesis is to address the limitations of previous approaches to semantic similarity search over spatio-textual data. By developing scalable and efficient semantic similarity search techniques, this thesis aims to enhance the performance of similarity searches and contribute to the broader goal of facilitating more efficient and effective analysis of spatio-textual data in various applications. To achieve this, the study focuses on designing and implementing novel methods for indexing and searching large spatio-textual datasets, while maintaining good performance, generalizability to new datasets, and adaptability to a range of use cases. To achieve this goal, the research will focus on the following research questions.

Research question 1 *What are the limitations of existing indexing and searching techniques for spatio-textual data, and how do they impact the efficiency and effectiveness of similarity searches?*

Research question 2 *How can novel indexing and searching methods be designed to address the challenges associated with semantic spatio-textual search?*

Research question 3 *What performance improvements can be achieved by designing new algorithms, and how do they compare to existing methods in terms of retrieval efficiency and effectiveness?*

1.5 Thesis Structure

The rest of this paper is organized as follows:

- Chapter 2 explores the state-of-the-art indexes and algorithms for semantic search over spatio-textual data.
- Chapter 3 introduces the theory, tools, and methods necessary to understand the work.

- Chapter 4 presents the novel spatio-textual index and algorithm DualiDistance and its accompanying methods.
- Chapter 5 details and discusses the experiments and results obtained from applying the proposed indexes and algorithms on a large dataset.
- Chapter 6 concludes the thesis and describes possible directions for future research.

2 Related Work

This Chapter provides a review of the existing literature on spatial keyword search and the generation of semantic representations for text. We then explore the state-of-the-art methods for semantic spatio-textual search. By highlighting the notable limitations of current approaches, the Chapter underscores the importance of developing new and improved methods.

2.1 Spatial Keyword Search

In recent years, spatial keyword search has become increasingly important, with a growing number of location-based devices and services requiring both spatial and textual relevance between a query and objects in the dataset[4, 5, 7, 15]. This approach combines geographic location and textual matching to provide users with more accurate and meaningful results than each domain might achieve independently. Initial efforts in the field concentrated on Spatial Keyword Search employing boolean constraints, which depends on precise keyword matching[4]. Nonetheless, this approach might yield unsatisfactory results due to its inflexible matching requirements. In order to tackle this problem, alternative methods have been proposed that permit more lenient matching criteria, accepting partial keyword matches and exhibiting a higher tolerance for spelling errors[4, 8, 26, 27]. However, a common limitation among these methods is their reliance on the syntactic matching of query keywords to text. This means they primarily focus on the exact matching between search terms and textual content without considering semantic relationships and contextual nuances. This limitation prevents them from retrieving objects that are synonyms but literally different from query keywords. As a result, there is a growing need for spatial keyword search methods incorporating semantic representations to better capture the meaning and intent behind user queries.

2.2 Semantic Representations of Text

Word embeddings are a state-of-the-art technique for creating semantic representations of text. They are a powerful tool for capturing the underlying meaning and intent of words and phrases within a text, addressing the limitations of syntactic matching in spatial keyword search methods. Incorporating word embeddings into a traditional spatial keyword search can significantly improve the quality of search results by considering semantic relationships between query terms and documents[6]. In this Section, we discuss various word embedding techniques and their potential applications in enhancing spatial

keyword search. Word2Vec[16] is a widely-used word embedding technique that learns continuous vector representations for words in a large corpus of text. The core idea behind Word2Vec is that words with similar meanings tend to appear in similar contexts. This approach represents words as high-dimensional vectors in a way that preserves their semantic relationships, allowing for the computation of similarity between words using cosine similarity or other distance measures. Global Vectors for Word Representation (GloVe)[18] is another popular word embedding method. GloVe learns word vectors by capturing the global co-occurrence information of words in a corpus. The technique optimizes a weighted least squares objective function that minimizes the difference between the dot product of word vectors and the logarithm of their co-occurrence probabilities. While traditional word embeddings like Word2Vec and GloVe have demonstrated success in various natural language processing tasks, they still have limitations, particularly in capturing context-dependent word meanings. Context-aware word embeddings, such as Embeddings from Language Models (ELMo) and Bidirectional Encoder Representations from Transformers (BERT), address this issue by generating embeddings that take into account the surrounding context of words within a sentence. ELMo[19] is a context-aware word embedding method that uses deep bidirectional language models to generate word representations. ELMo leverages a pre-trained, multi-layer bidirectional Long Short-Term Memory (LSTM) network to capture both left-to-right and right-to-left context. The final ELMo embeddings are a weighted combination of the hidden layer representations from the LSTM network. BERT[9] is another context-aware word embedding technique based on the transformer architecture[24]. BERT is pre-trained using a masked language model objective, which allows it to learn bidirectional context by predicting masked words in a sentence based on their surrounding context. The model also employs a next-sentence prediction task, which enables it to learn relationships between sentences. BERT has achieved state-of-the-art performance on a wide range of NLP tasks and is a good candidate for word embedding generation.

2.3 Semantic Spatio-textual Search

The S2R-tree[6] represents the state-of-the-art in semantic spatio-textual searches, addressing the same problem as this thesis. Essentially, it expands upon the R-tree structure, discussed in 3.7.1, integrating spatial coordinates with low-dimensional semantic vectors to represent the text. These vectors are derived from projecting high-dimensional word embeddings into an m-dimensional space using a pivot-based method. As a result, each vector is represented as an m-dimensional vector, with the value of m being as low as 2. The vector thus represents the distance of the specific vector from the m pivots in high-dimensional space. The S2R-tree is built on the spatial layer of coordinates, with each leaf further developed into an R-tree indexing the m-dimensional representations as the semantic layer. Index nodes are then extended with m-dimensional Minimum Bounding Boxes (MBB) of semantic vectors, allowing only parts relevant to the query to be accessed. However, its main limitation arises from its spatial-first approach, where the index primarily focuses on organizing data based on spatial coordinates. Our experimental

findings indicate that its effectiveness is comparable to a basic R-tree implementation that solely indexes the spatial domain.

The NIQ-tree[20] and its extension, LHQ-tree[21], are multi-level indexing structures that also adopt the spatial-first approach. These structures use a Quadtree at the top level, discussed in 3.7.1, to index data objects based on their spatial coordinates. In the second level, objects within each leaf node are indexed using topic relevance and iDistance[13]. The topic relevance is derived using Latent Dirichlet Allocation (LDA), discussed in Section 3.7.2, to create a probabilistic topic model that captures semantic information. Finally, at the last level, n-gram inverted lists are built to retrieve objects based on the presence of specific n-grams within them. While the NIQ-tree and LHQ-tree share similarities with our work, our approach integrates spatial and textual information using word embeddings, which differ from the topic modeling approach employed in the NIQ-tree and LHQ-tree.

Sun et al. [22] propose an interactive spatial keyword querying method with semantics. They also employ LDA to extract semantic representations from the data. In their approach, they address the challenges of short queries and semantic ambiguity by incorporating user interactions and feedback in query results. This differs from our approach, which does not rely on user interaction. User feedback in spatio-textual querying has limitations as it relies on users' understanding and interpretation of queries, which can result in inaccurate or ambiguous feedback and less precise query results. Additionally, incorporating user feedback adds an extra step of interaction, potentially slowing down the query process, and scalability challenges arise when collecting and processing feedback from a large number of users due to the associated time and computational demands.

2.4 Limitations of Existing Methods

Despite significant advancements in semantic spatio-textual similarity search techniques, existing methods continue to face significant limitations related to scalability, efficiency, and accuracy. One key limitation of these methods is the dominance of the spatial dimension in their indexing structures. The spatial-first approach seems to restrict the potential for exploiting semantic similarities for efficient search and retrieval. When the spatial dimension dominates the indexing process, the effectiveness is comparable to that of indexing methods that only consider the spatial domain, such as a basic R-tree. Therefore, there is a need for a more balanced integration of spatial and textual information.

Current approaches predominantly rely on low-dimensional semantic vectors or topic modeling techniques to capture the semantic meaning of queries. However, these methods have inherent limitations. Low-dimensional semantic vectors, obtained by projecting high-dimensional word embeddings into a lower-dimensional space, can result in a significant loss of semantic information, thereby hindering precise semantic capture and contextual understanding of queries. On the other hand, topic modeling approaches provide a probabilistic interpretation of semantic information but may struggle to capture the fine-grained nuances of individual semantic contexts, particularly in large and complex

datasets.

Furthermore, incorporating user feedback to enhance semantic relevance introduces computational complexity and potential inconsistencies due to subjective interpretations. Incorporating user interaction can impact system scalability, as processing feedback from a large number of users poses time and computational challenges. In designing DualiDistance, we aim to address these limitations. We focus on developing an indexing structure that integrates spatial and textual dimensions equally, manages high-dimensional semantic representations efficiently, and provides precise results without relying on user interaction.

3 Background Theory

3.1 Symbols

The primary symbols used in this paper are summarized in Table 3.1.

Table 3.1: Description of symbols used in the algorithms

Symbol	Description
q	Query object
qid	The id of the query object
p	Data object
k	Number of nearest neighbor objects required by the query
λ	Spatio-textual weighting parameter
T	iDistance B+-tree
c	iDistance stretch factor
S	Set of k nearest neighbors
r	Radius of a sphere
Δr	The change of r
P	Set of data partitions
O	Set of reference points
P_i	The ith partition
O_i	The ith reference point
PQ	Priority queue for candidate nodes
DB	The dataset
DS	The dataspace
$distmax_i$	Maximum radius of partition P_i
$dist(p1, p2)$	Metric function returns the distance between objects p_1 and p_2
$idist(d)$	Returns the iDistance based on $d = dist(p1, p2)$
$querydist(q)$	Query radius of q
$sphere(q, r)$	Sphere of radius r and center q
$furthest(S, q)$	Sphere of radius r and center q

3.2 Spatio-textual Data

Spatio-textual data refers to data that contains both spatial and textual information. A spatio-textual object, denoted as p, is a data object that encapsulates both spatial and textual information. The spatial information is represented by a geographical location, characterized by coordinates $(p.x, p.y)$. The textual information of the object, represented as $p.text$, comprises a textual annotation associated with the object. This annotation may consist of a string of words, terms, or other types of text data relevant to the object. Spatio-textual data can be derived from structured data sources like databases of addresses and associated textual information, or unstructured sources like social media posts containing location information and accompanying text.

3.3 Distance Metrics

A key aspect of similarity search is the use of various distance metrics to quantify the similarity between data objects. This Section reviews commonly used distance metrics for data with spatial and textual components.

3.3.1 Spatial Distance Metrics

The Minkowski distance - The Minkowski distance measures the distance between two n-dimensional vectors $x = \{x_1,...,x_n\}$ and $y = \{y_1,...,y_n\}$. It is defined as:

$$\ell_p(x, y) = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p} \tag{3.1}$$

Two of the most well-known variants are ℓ_1 (Manhattan distance) and ℓ_2 (Euclidean distance), where the latter is most relevant for our purposes.

The Manhattan distance - The Manhattan distance measure the distance between two n-dimensional vectors $x = \{x_1,...,x_n\}$ and $y = \{y_1,...,y_n\}$. It is defined as:

$$\ell_1(x, y) = \sum_{i=1}^{n} |x_i - y_i| \tag{3.2}$$

The Euclidean distance - The Euclidean distance measure the distance between two n-dimensional vectors $x = \{x_1,...,x_n\}$ and $y = \{y_1,...,y_n\}$. It is defined as:

$$\ell_2(x, y) = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2} \tag{3.3}$$

3.3.2 Textual Similarity Metrics

Textual Similarity Metrics are quantitative measures used to evaluate the similarity or relatedness between two text documents or strings based on their content. Two commonly used textual similarity metrics are Cosine Similarity and Jaccard Similarity.

Cosine Similarity measures the cosine of the angle between two non-zero vectors in a term frequency space. There are various ways of transforming sentences into vectors, one example being word embeddings (**??**). It is a widely used metric for comparing the similarity between documents represented by term frequency vectors. The formula for Cosine Similarity between two vectors A and B is given by:

$$CosineSimilarity(A, B) = \frac{A \cdot B}{\|A\|\|B\|} \tag{3.4}$$

Jaccard Similarity, also known as the Jaccard Coefficient or Jaccard Index, is a metric used to measure the similarity between two sets. It is defined as the ratio of the size of the intersection of the sets to the size of their union. For two sets A and B, the Jaccard Similarity can be calculated as:

$$JaccardSimilarity(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{3.5}$$

3.3.3 Hybrid Distance Metrics

Hybrid Similarity Metrics are designed to evaluate the similarity or relatedness between multi-dimensional data objects, such as spatio-textual data described in Section 3.2. These metrics take into account both spatial and textual dimensions, combining the individual similarity measures for each dimension into a unified metric. Typically, hybrid similarity metrics are formulated as a weighted combination of spatial and textual similarity metrics. The following is an example definition of a hybrid distance metric. Given two spatial-textual objects p_i and p_j, with spatial coordinates (x_i, y_i) and (x_j, y_j) and textual descriptions T_i and T_j, the hybrid distance metric $d_h(p_i, p_j)$ is defined as

$$d_h(p_i, p_j) = \lambda \cdot d_s(p_i, p_j) + (1 - \lambda) \cdot d_t(T_i, T_j) \tag{3.6}$$

where $\lambda \in [0, 1]$ is a weighting parameter, $d_s(p_i, p_j)$ represents the spatial distance between the two objects, and $d_t(T_i, T_j)$ represents the textual distance between their descriptions.

3.4 Features

This Section is from the specialization report in the course TDT4501. A feature of a data object is a measurable attribute or characteristic of a phenomenon. The set of features for a data object should be informative, discriminating and independent to achieve good performance. This means that a feature should contain relevant information about the data, differentiate between different classes or groups within the data and not correlate with other features. Since features are usually numerical, the set of features that defines a data object is stored in a feature vector. An increase in dimensionality implies an increase in the number of features used to describe the data. For example, in cancer research, age and the current stage of the spread can be used as features to decide a cancer patient's

prognosis. The two features are the dimensions of a feature vector. However, other factors can give information to the prognosis, like the type of tumor and comorbidities. By adding these features to the feature vector, we are increasing the dimensionality of our data.

3.5 Impact of High Dimensionality

The term data dimensionality denotes the count of distinct features or attributes characterizing each data object in a dataset, explained in Section 3.4. The dimensionality can also increase dramatically due to the use of word embeddings, discussed in Section 2.2, where words are mapped into high-dimensional vector spaces. The influence of data dimensionality is particularly noteworthy in similarity search methods, which often grapple with high-dimensional data. This increased complexity leads to a phenomenon known as the *curse of dimensionality*. With the rise in dimensionality, data becomes increasingly sparse and requires significant computational resources for processing and analysis. Such sparsity challenges conventional search methods, which are often designed for dense data. Interpreting similarity or closeness between data objects also becomes more challenging with high dimensionality. In high-dimensional spaces, distances between data objects can become almost equidistant, complicating the distinction between the nearest and the furthest neighbors. Therefore, researchers adopt varied strategies to manage high-dimensional data. Some focus on optimizing indexing algorithms for lower-dimensional data, while others strive to mitigate the intricacies of indexing high-dimensional data. A common strategy to cope with high dimensionality involves employing alternative vector similarity metrics or using dimensionality reduction techniques. Dimensionality reduction strives to convert high-dimensional data into a representation with fewer dimensions while preserving essential information. The most straightforward technique involves selecting a relevant subset of features from the total feature set. A more detailed exploration of dimensionality reduction techniques can be found in Section 3.6.

3.6 Dimensionality Reduction

Dimensionality reduction methods can be used to avoid the problem stated in Section 3.5. These techniques attempt to reduce the complexity of high-dimensional data while preserving its essential structure and relationships. It aims to transform the original high-dimensional data into a lower-dimensional representation, which can facilitate more efficient processing, visualization, and storage. This Section provides an overview of some popular dimensionality reduction techniques, including t-Distributed Stochastic Neighbor Embedding (t-SNE)[23], Linear Discriminant Analysis, and Principal Component Analysis (PCA). The t-SNE algorithm can effectively capture complex relationships, making it useful for visualizing high-dimensional data. However, it is not well-suited for similarity searches as it is specifically used for visualization purposes only. Linear Discriminant Analysis is a supervised technique that focuses on maximizing class separation using class label information. However, for similarity searches on spatio-textual data, which

does not have well-defined class labels or relationships, PCA is a better option. PCA is an unsupervised method that captures the maximum variance in the data, preserving the inherent structure of the data without relying on class labels. It achieves this by transforming a set of variables that may be related to each other into a smaller set of uncorrelated variables called principal components. These components are arranged in order of importance, with the first components capturing the most significant patterns of variation in the original data.

3.7 Indexing Methods

Indexing methods are data structures and algorithms designed to organize and store data in a way that enables efficient retrieval, modification, and deletion operations. They play a crucial role in reducing the search space and improving the performance of nearest neighbor search algorithms. In the context of high-dimensional spatio-textual data, various indexing methods have been proposed. This Section explores the most common one-dimensional indexes of spatial and textual data.

3.7.1 Spatial Indexing

Spatial indexing methods are specifically designed for organizing and searching data objects based on their spatial attributes, such as coordinates. These enable the efficient processing of spatial queries. R-trees[12], kd-trees[2], and quad-trees[11] are considered some of the most effective methods utilized in spatial indexing. R-trees utilize a hierarchical tree structure optimized for multi-dimensional data, encapsulating data objects within Minimum Bounding Rectangles (MBR). Each node in the tree is linked to a MBR. The data objects are stored in the leaf nodes. If a node isn't a leaf node, it contains additional MBRs. Importantly, all objects that are encapsulated within a larger parent MBR are also included within one of its child MBRs. Kd-trees, or k-dimensional trees, are binary tree structures for organizing k-dimensional objects. Utilizing splitting hyperplanes, each non-leaf node divides the space into two half-spaces, with left and right hyperplane objects represented in the respective left and right subtrees. This recursive process results in a binary tree partitioning space based on the k-dimension object coordinates. Quad-trees, primarily used for two-dimensional spatial data, recursively partition a space into four quadrants. Each node symbolizes a square area, with non-leaf nodes having four children representing the respective quadrants of the square. The partitioning continues until each node encapsulates at most one object or reaches a specified depth level.

3.7.2 Textual Indexing

Textual indexing focus on organizing and searching data based on their textual attributes, such as keywords, terms, or semantic relationships. These enable efficient processing of textual queries, including keyword search, phrase search, and semantic search. Inverted indexes are data structures that store a mapping from words to their locations in a set of documents. By listing the documents in which a particular term appears, inverted

indexes facilitate efficient keyword-based search, allowing for quick retrieval of relevant documents. This technique is commonly used in search engines and large-scale text databases.

Suffix trees[25], on the other hand, are trie-based data structures that store all the suffixes of a given text. By enabling efficient pattern matching and substring searches, suffix trees are particularly useful for applications that require complex string operations, such as computational biology, data compression, and natural language processing. Suffix trees can also be adapted to handle multiple texts or documents through generalized suffix trees, further extending their applicability.

Latent Dirichlet Allocation (LDA)[3] is a generative probabilistic model that is widely used for extracting abstract topics from collections of documents, thereby capturing the semantic structure of the data. In the context of semantic similarity search over textual data, LDA can be used to create representations of textual data that capture the underlying semantic content. LDA perceives each document as a mixture of topics, with each topic being a distribution over the vocabulary of the text. It starts by assigning every word in a document to a random topic, then iteratively updates these assignments based on the prevalence of topics within the document and the prevalence of words within the topics. As the algorithm converges, it creates a stable set of topics each characterized by a particular set of words. These topics form the semantic representation of the document, capturing the underlying semantic content. Therefore, LDA enables the conversion of complex textual data into more manageable topic-based representations that preserve the core semantic information.

Word embeddings are vector representations of words that capture their semantic meanings in the context of textual indexing, and are generated based on the contextual usage of words in large corpora. The generated vectors preserve semantic relationships, enabling more accurate and context-aware search capabilities within textual indexes. They will be discussed in detail in Section 2.2 since they are a key factor in the performance of our method.

3.8 B+-Tree

Space partitioning methods divide the search space into non-overlapping regions, each containing a subset of the data objects. By recursively partitioning these regions, a hierarchical data structure is constructed. Common space partitioning methods are described in Section 3.7.1. While space partitioning is commonly used for nearest neighbor search, some methods use alternative data structures such as B+-trees. The B+-tree[1] is a balanced tree data structure designed for efficient storage and retrieval of large datasets in block-oriented storage systems. Compared to binary search trees, the B+ tree is especially advantageous due to its high fanout, which refers to the number of pointers to child nodes in a node. Typically, the fanout of a B+ tree is on the order of 100 or more, significantly reducing the number of Input/Output (I/O) operations required to locate elements in the tree. The keys within each node are sorted in ascending order, and the tree is balanced so that all leaf nodes are at the same level. Nearest neighbor search

using B+-trees is done by mapping high-dimensional data objects to a one-dimensional space. This mapping enables the data objects to be efficiently stored in the tree and accessed using a standard tree traversal search algorithm. To find the nearest neighbor to a query point, the query object q is first mapped to one-dimensional space. A range search is then executed in the B+-tree around the mapped query object to retrieve a set of candidate points. The distances between the query object and the candidate objects are then computed, and the object with the smallest distance is returned as the nearest neighbor. By leveraging the properties of B+-trees and the one-dimensional mapping technique, this approach may provide a fast and accurate method for nearest-neighbor searches over large datasets.

3.9 Nearest Neighbor Search Algorithms

Nearest Neighbor (NN) search is a form of proximity search that focuses on identifying the object in a given dataset that is closest or most similar to another specified object. This closeness is typically quantified using a dissimilarity function where larger values denote less similarity. Consider a set of objects $DB = p_1, p_2, ..., p_n$ located in a d-dimensional space DS. Given a query object q in DS, the task of NN search is to find the object in DS that is closest to q. A direct extension is the k-Nearest Neighbors (KNN) search, which identifies the k closest objects to the query object instead of just the singly closest.

3.9.1 Exact Nearest Neighbors

The linear search algorithm, also known as the exhaustive search or brute-force search, is a simple and straightforward method for exact nearest neighbor search. The algorithm iterates over all objects in the dataset and computes the distance between each object and the query point. The object with the smallest distance is returned as the nearest neighbor. Algorithm 1 is a simple implementation of a linear nearest neighbor search over a dataset DB and a query object q. It keeps track of the current nearest neighbor p^* relative to q and its distance $d_{\min}$. Finally, p^* is returned when all objects in DS have been checked.

The time complexity of algorithm 1 is $O(n)$, as it requires computing the distance between q and each object in DB. The space complexity is $O(1)$, as the algorithm only needs to store $d_{\min}$ and p^*.

3.9.2 K-Nearest Neighbor Search

The KNN search is an extension of the nearest neighbor search in Section 3.9.1. Instead of finding the sole nearest neighbor p^*, KNN search aims to identify the k nearest neighbor objects to q. The problem can be defined as follows. Given a set of objects $\{p_i\} \in P$ and a query object q, the objective is to find the k entities $P_k = \{p_1, \ldots, p_k\}$ such that

$$d(q, p_i) \leq d(q, p_j), \quad \forall p_i \in P_k \text{ and } \forall p_j \in P - P_k.$$

Algorithm 1 Linear Search Algorithm for Exact Nearest Neighbor Search

Require: DS, q
Ensure: p^*
1: $p^* \leftarrow$ None
2: $d_{\min} \leftarrow \infty$
3: **for** $i \leftarrow 1$ to n **do**
4: $d_i \leftarrow d(p_i, q)$
5: **if** $d_i < d_{\min}$ **then**
6: $p^* \leftarrow p_i$
7: $d_{\min} \leftarrow d_i$
8: **end if**
9: **end for**
10: **return** p^*

In contrast to exhaustive linear scanning of the entire dataset, KNN search typically employs a strategy of incrementally expanding the search radius around the query object until the KNN are identified. The KNN search, as outlined in algorithm 2, employs an incremental approach in identifying the K nearest neighbors to a query object q. Starting with a query sphere, *sphere(q, r)*, centered at q with a small initial radius, it maintains a result set holding objects that constitute the K nearest neighbors. This radius expands incrementally until the K nearest neighbors to q are found.

In KNN search, data is often divided into distinct partitions or subsets using a partitioning algorithm, *partitions* in Algorithm 2, like KMeans.Instead of considering every object in the dataset, the algorithm examines only the objects from partitions intersecting with *sphere(q, r)*, determined by the function $intersects(p, q, r)$. This significantly reduces the number of distance computations required compared to a linear approach. The time complexity of the search is reduced to $O(m)$, where m denotes the number of objects within the partitions that intersect with the search sphere.

3.9.3 Approximate Nearest Neighbors

As the scale and dimensionality of data increase, it can become computationally prohibitive to conduct an exact nearest neighbor search. To alleviate this, a more computationally efficient paradigm known as Approximate Nearest Neighbors (ANN) search has been proposed. The main principle of ANN is to identify a set of k data objects in the dataset that are similar to the query object, although not necessarily the exact closest. In this context, an acceptable trade-off exists between the accuracy of results and computational efficiency. While it does not guarantee to provide the exact nearest neighbors, it offers an approximate result that is often indistinguishable in practical applications. Several techniques can be employed for ANN searches. One method is Locality-Sensitive Hashing (LSH), which simplifies the data by grouping similar items together to speed up the search. The more similar the two items are, the more likely they will be grouped together, which helps reduce the search space and time. LSH utilizes hash functions to group

Algorithm 2 K-Nearest Neighbor Search with a Search Radius

Require: Dataset $D = p_1, p_2, \ldots, p_n$, Query object q, Number of neighbors K, r, Partition function *partitions*(DS), Intersection function *intersects*(p, q, r)
Ensure: K nearest neighbors within radius r, N

```
N ← ∅
r ← 0
P ← partitions(DS)
while N ≤ k do
    Increase search radius r
    for each partition p in P do
        if intersects(p, q, r) then
            for each member m in p do
                d_i ← dist(m, q)
                if |N| < K then
                    Add m to N
                else
                    Find farthest object p_f in N
                    If d_i < d(p_f, q), replace p_f with m in N
                end if
            end for
        end if
    end for
end while
return N
```

similar items together. While these hash functions aim to assign similar items to the same bucket, there is still a possibility of dissimilar items colliding or similar items being placed in different buckets, resulting in an approximation rather than an exact result. Another method is to introduce an error tolerance parameter to make the nearest neighbor search approximate. This parameter specifies the maximum acceptable deviation from the exact nearest neighbors. By allowing a small tolerance for the search results to be within a certain distance from the exact nearest neighbor, the search process can be significantly accelerated, especially in large and high-dimensional datasets. This relaxation of precision strikes a balance between speed and accuracy, ensuring that the results remain acceptable while achieving faster query performance.

3.10 Integrating Rankings

Integrating Rankings is a technique employed for combining ranked lists from different domains. Given a set of objects, each object has m grades associated with m distinct criteria; the goal is to efficiently integrate these object rankings based on the provided grades. The grade for each object's i-th criterion is represented by x_i. Grades are

assumed to be within the range of $0 \leq x_i \leq 1$, where a higher value of x_i indicates better performance according to the i-th criterion. The input consists of m sorted lists, where each list represents one criterion. The i-th list is sorted according to x_i. The main objective is to find the top k objects that best satisfy all the criteria. Given the m sorted lists of objects and their grades based on different criteria, the challenge is to devise an efficient algorithm that can integrate these rankings to identify the top k objects. The algorithm should consider the trade-offs between accuracy, computational complexity, and resource utilization while achieving the desired ranking integration. The following is a presentation of the main algorithms for integrated rankings[10]. The Naive Algorithm is an obvious and simple method for integrating sets into a top-k candidate set. It evaluates each item in the dataset by computing the overall grade of every object and returns the top k answers. The Naive Algorithm is easy to implement but comes with a large computational complexity.

3.10.1 Fagin's Algorithm

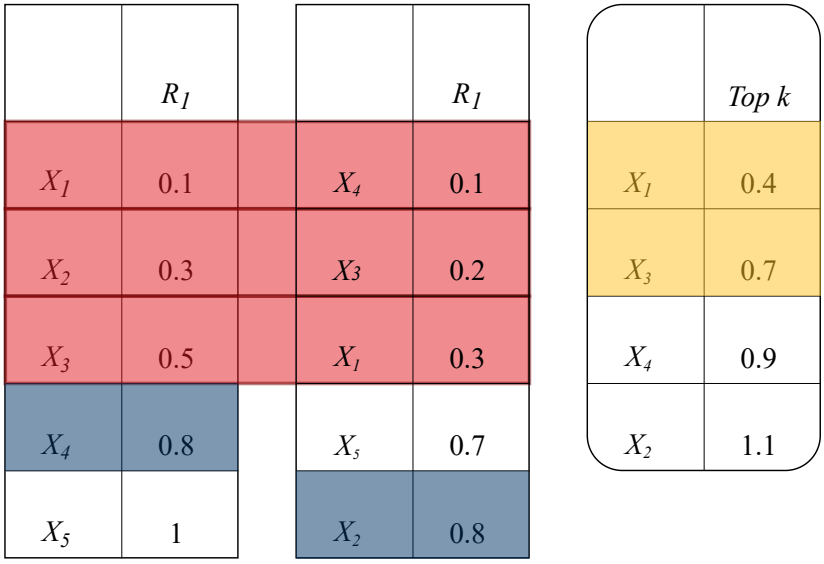


Figure 3.1: Fagin's algorithm with $k = 2$

Fagin's Algorithm (FA)[10] is a more efficient method for top-k queries, utilizing sorted access to each list. The algorithm carries out sorted access in parallel to each of the m sorted lists L_i and stops when there are at least k objects, each of which has been seen in all the lists. Next, for each object R that has been seen, it retrieves all of its fields $x_1, ..., x_m$ by random access and computes $F(R) = F(x_1, ..., x_m)$. Finally, it returns the top k answers. FA guarantees to find the exact top-k results but may require accessing a large number of objects in each list. To understand why FA guarantees the exact top-k results, consider an object R that was not seen with grades $x_1, ..., x_m$, and an object seen in every

sorted list S with grades $y_1, ..., y_m$. Since R was not seen, it implies that $x_i \leq y_i$ for each i. As a result, the function value for R is $F(R) = F(x_1, ..., x_m) \leq F(y_1, ..., y_m) = F(S)$. Therefore, R cannot be part of the top-k results. Figure 3.1 demonstrates this with $k = 2$, shown in green. After the third iteration, shown in red, there are two objects, X_1 and X_3, that have been seen in all the sorted lists. The algorithm terminates and returns the current Top-k objects seen. It is important to note that every object that has been seen in at least on of the sorted sets is considered, not just the objects that have been seen in all sets. To consider every object random accesses is performed to get every grade, shown in blue.

3.10.2 Treshold Algorithm

Threshold Algorithm (TA)[10] is another alternative that leverages a threshold value to stop the search early when the top k objects have been confidently identified. The algorithm performs sorted access in parallel to each of the m sorted lists L_i. As each object R is seen under sorted access, it retrieves all of its fields $x_1, ..., x_m$ by random access and computes $F(R) = F(x_1, ..., x_m)$. If this is one of the top k answers so far, it is stored in a data structure. For each list L_i, let $\hat{x}_i$ be the grade of the last object seen under sorted access. The threshold value t is then defined as $F(\hat{x}_1, ..., \hat{x}_m)$. When k objects have been seen whose grade is at least t, the algorithm terminates, and the top-k objects are returned. TA usually requires fewer accesses than FA while still guaranteeing the exact top-k results. The Threshold Algorithm is correct for every monotone aggregate function F. Moreover, TA is optimal in a very strong sense, as it is as good as any other algorithm on every instance, within a constant factor, except for pathological algorithms that make wild guesses. This optimality ensures that TA provides efficient and accurate top-k query results in a wide range of scenarios. TA can be adapted to create an approximation algorithm for situations where obtaining approximately top-k answers is sufficient rather than exact top-k results. Given a parameter, $\varepsilon > 0$, an ε-approximation of top-k answers is a collection of k objects $R_1, ..., R_k$ so that for each object R not among them, $(1+\varepsilon) \cdot F(R_i) \geq F(R)$. To transform TA into an approximation algorithm, the stopping rule is simply modified. The algorithm halts when k objects have been seen whose grade is at least $\frac{t}{1+\varepsilon}$. This adjustment enables the efficient retrieval of approximately top-k answers, catering to situations where an exact top-k result is not strictly required. There are also methods for scenarios where sorted access or random access to the lists is restricted, but these are not relevant for our purposes.

3.11 The iDistance Method

In iDistance[13], an novel approach is proposed for effective KNN search within high-dimensional metric spaces. We allocate this Section to the iDistance method as it constitutes a crucial component of the novel similarity search techniques proposed in this thesis. An in-depth examination of the iDistance method allows us to highlight its key features, advantages, and the rationale behind employing it as the foundation for our

solutions. We will discuss the query processing part of the iDistance algorithm in Section 4.3 instead, since our implementation will be slightly different than the original.

3.11.1 Overview

The primary motivation behind iDistance is to reduce the search space, enabling efficient similarity search. iDistance achieves this by splitting the data space into non-overlapping partitions and using distance-based indexing to facilitate fast search operations. The method is founded on three key observations. First, the similarity or dissimilarity between data objects can be assessed utilizing a reference point. Second, data objects can be sorted according to their distances relative to a reference point. Lastly, metric distances are one-dimensional values. The transformation of high-dimensional data into a single-dimensional space allows for the utilization of existing single-dimensional indexes, such as the B+-tree in the iDistance algorithm. This transformation facilitates KNN searches by converting them into simpler one-dimensional range searches, leveraging the efficiency and effectiveness of established index structures. Given a collection of data objects DB in a d-dimensional metric space DS and a related distance function $dist$, let $P = \{p_1, p_2, p_3\}$ represent three data objects in DS. The distance function $dist$ has the following properties.

$$dist(p_1, p_2) = dist(p_2, p_1) \quad \forall p_1, p_2 \in DB \tag{3.7}$$
$$dist(p_1, p_1) = 0 \quad \forall p_1 \in DB \tag{3.8}$$
$$0 < dist(p_1, p_2) \quad \forall p_1, p_2 \in DB; p_1 \neq p_2 \tag{3.9}$$
$$dist(p_1, p_3) \leq dist(p_1, p_2) + dist(p_2, p_3) \quad \forall p_1, p_2, p_3 \in DB \tag{3.10}$$

Although other distance functions can be applied to iDistance, we consider the Euclidean distance described in Section 3.3. A high-dimensional database can be divided into partitions with reference objects O_i for data partition P_i. Data object p in the partition can be related to O_i concerning its distance to it, $dist(O_i, p)$. Using the triangle inequality, we see that

$$dist(O_i, q) - dist(p, q) \leq dist(O_i, p) \leq dist(O_i, q) + dist(p, q). \tag{3.11}$$

In the scenario where we use a search radius defined by $querydist(q)$, we aim to find all data objects p that satisfy $dist(p, q) \leq querydist(q)$. For all such objects p, they must therefore meet the condition

$$dist(O_i, q) - querydist(q) \leq dist(O_i, p) \leq dist(O_i, q) + querydist(q). \tag{3.12}$$

In partition P_i, we only need to consider candidate objects p whose distance from the reference point, $dist(O_i, p)$, is constrained by this inequality, which typically defines an annulus around the reference point. Let $distmax_i$ be the distance between O_i and the furthest object in partition P_i. If $dist(O_i, q) - querydist(q) \leq distmax_i$, then partition P_i must be searched for candidate objects. Otherwise, this partition can be excluded

from consideration. Figure 4.1 illustrates a search process involving three partitions P_1, P_2, and P_3. When considering the query object q and query radius r, it is necessary to search within partitions P_1 and P_2, while partition P_3 can be pruned.

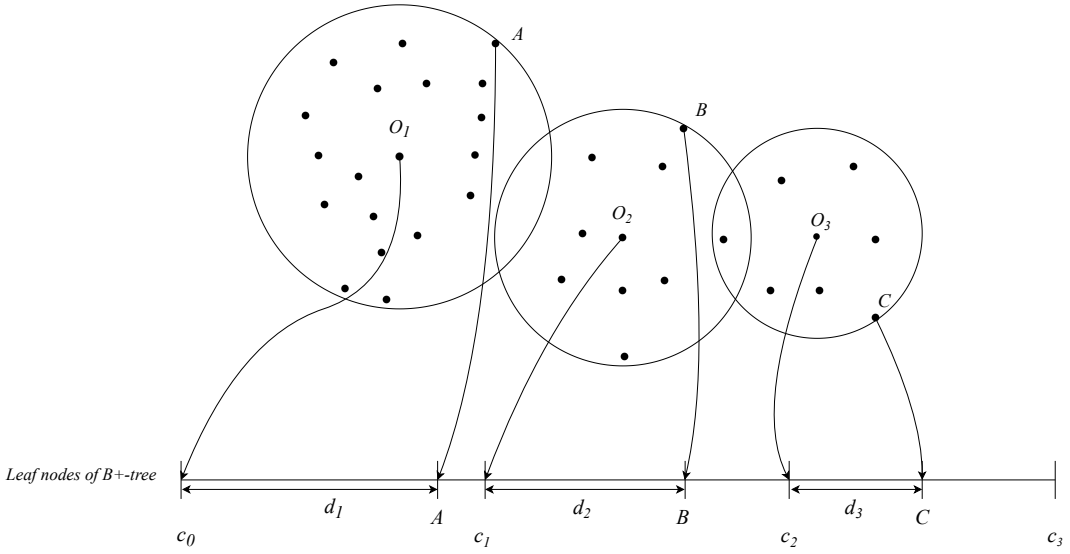


Figure 3.2: The iDistance Index Method

3.11.2 Index Structure

In iDistance, high-dimensional objects are transformed into objects in a single-dimensional space using a three-step algorithm. First, the high-dimensional data space is split into a set of partitions. Second, a reference point is identified for each partition. Suppose there are N partitions $P = \{P_0, P_1, \ldots, P_{N-1}\}$ and their corresponding reference objects $\{O_0, O_1, \ldots, O_{N-1}\}$. Finally, all data objects are represented in a single-dimensional space. A data object p is assigned an index key y given by the equation

$$y = i \times c + \text{dist}(p, O_i) \tag{3.13}$$

Here, i represents the index of the partition p belongs to, c is a constant used to stretch the data ranges, and $\text{dist}(p, O_i)$ represents the distance between p and O_i. Essentially, c serves to partition the single-dimensional space into regions so that all objects in partition P_i are mapped to the range $[i \times c, (i + 1) \times c)$. c must be set sufficiently large to avoid

overlap between the index key ranges of different partitions. This concept is illustrated in Figure 3.2. Two data structures are employed in iDistance. First, a B+-tree, described in Section 3.8, is used to index the transformed objects to facilitate speedy retrieval. Second, an array is used to store the N data space partitions and their respective reference points, which helps determine the data partitions that need to be searched during query processing.

3.11.3 Data Partitioning and Clustering

The data space is divided into partitions, each with an associated reference point. Various reference point selection and partitioning strategies are employed in the iDistance paper. The partitioning strategies are split into space-based partitioning and data-based partitioning respectively. Space-based partitioning involves dividing the data space into equal partitions using geometric patterns and assigning reference objects based on the structure of the partitions. Data-based partitioning, on the other hand, focuses on identifying and leveraging the inherent clustering or correlation within the data. The reference objects are selected based on the properties of the clusters, such as their centers or edges. The study highlights the importance of selecting a partitioning strategy that minimizes partitioning sphere overlap, in order to promote high levels of pruning. They find that data-based partitioning is more suitable for real-life data, as it often exhibits clustering or correlation naturally. In this context, the K-means clustering algorithm was adopted, with the number of clusters serving as a tuning parameter. The authors provide an extensive experimental evaluation of iDistance, demonstrating its effectiveness, and propose a cost model that can be utilized for query optimization.

4 Dual Semantic Similarity Search

In this Chapter, we provide a comprehensive overview of our proposed method for performing dual index semantic similarity searches, trying to address the limitations identified in Section 2.4. We first explore the utilization of the K-Means algorithm and the optimization of the original iDistance implementation. The main part of this Chapter details our contribution, a Dual Spatio-textual iDistance algorithm, DualiDistance. This encompasses various aspects such as index construction, the implementation of the common top-K using the threshold algorithm, and the employment of inter- and intra-cluster pruning strategies. Finally, we discuss the rationale and decision-making process that motivated the development of DualiDistance.

4.1 K-Means with Maximum Radii Calculation

Algorithm 3 is an iterative algorithm based on K-Means[14] designed to partition a given dataset DB into K distinct partitions. The algorithm's output includes the partitions P, the set of reference points O, and the set of maximum cluster radii $distmax$. To begin, O is initialized either randomly or using a heuristic method, while $distmax$ is initialized with zeros. The algorithm then enters a loop that continues until convergence is achieved. Within this loop, O_i for each partition P_i is calculated as the mean of all objects in DB assigned to P_i. Next, for each data object p in DB, the distance $d(p, O_i)$ from p to each point of O is calculated. p is then assigned to the closest reference point O_i, and its partition assignment in P is updated accordingly. Once convergence is reached, the algorithm calculates the maximum radii for each partition by setting the corresponding $distmax_i$ to be the maximum distance from any object in partition P_i to the reference point O_i. Finally, the algorithm returns the partitions P, the reference points O, and the maximum radii $distmax$.

4.2 Optimizing the iDistance Algorithm

By implementing the iDistance algorithm from scratch, we are able to improve upon the method as we see fit. The B+-tree implementation is provided by Gregory Popovitch[1]. To optimize the iDistance algorithm for our specific needs, we can incorporate several enhancements that may help boost its efficiency and effectiveness. The iDistance algorithm, as presented in the original paper, makes use of recursion. This might lead to a considerable call stack for large datasets and potentially cause stack overflow issues.

[1] https://github.com/greg7mdp/parallel-hashmap

Algorithm 3 K-Means Clustering Algorithm with Maximum Radii Calculation

Require: DB, K
Ensure: P, O, $distmax$
1: $O \leftarrow \emptyset$
2: $R \leftarrow \emptyset$
3: **while** not convergence **do**
4: **for** $i = 1$ to K **do**
5: $O_i \leftarrow Mean(DB, P_i)$
6: **end for**
7: **for** each $p \in DB$ **do**
8: $O_{close} \leftarrow ClosestO(O, p)$
9: $AddToPartition(P_{close}, p)$
10: **end for**
11: **end while**
12: **for** $i = 1$ to K **do**
13: $distmax_i \leftarrow MaximumDistance(P_i, O_i)$
14: **end for**
15: **return** P, O, $distmax$

To prevent this, we can employ iterative techniques that mimic recursion but avoid the drawbacks associated with deep recursive function calls. This modification not only circumvents potential stack overflow problems but we also found that it enhances the algorithm's execution speed. Pruning is a critical aspect of the iDistance algorithm that can significantly reduce the search space by eliminating irrelevant partitions and objects within relevant partitions. To further improve efficiency, we can employ novel pruning techniques that better identify and disregard irrelevant search space. Similarly, we can improve efficiency with early termination techniques. The iDistance algorithm also executes the searches of the spatial dimension and textual dimension sequentially. To speed up the search performance, we could leverage parallel processing techniques. By dividing the dataset into spatial and textual indexes and processing them concurrently, we could cut down the search time and improve the algorithm's scalability.

4.3 Optimized One-Dimensional iDistance

This Section describes our implementation of the iDistance algorithm and the changes we have made to the original. The optimized One-Dimensional iDistance algorithm performs indexing and k-nearest neighbors search over either spatial or textual data objects. It partitions the data using the K-Means implementation in Algorithm 3 and leverages the B+ tree structure described in 3.8 to perform an exact k-nearest neighbor search. In Algorithm 4, the dataset DB is first partitioned into K disjoint partitions, P, using Algorithm 3. Each partition is assigned a reference point O_i, which is the mean value of all objects in P, and radius $distmax_i$, which is the distance to the most distant object

in p_i. The iDistance, *idist*, is then computed for each data object using equation 3.13 and the stretch constant c. The iDistance values and the ids of the corresponding data objects are then inserted into the B+ tree T.

Algorithm 4 iDistance: Nearest Neighbor Index

Require: DB, K, c
Ensure: T
1: $T \leftarrow \emptyset$
2: P, O, $distmax \leftarrow$ K-Means(DB, K)
3: **for** each $P_i \in P$ **do**
4: **for** each $p \in P_i$ **do**
5: $d_p \leftarrow dist(O_i, p)$
6: $iDistance_p \leftarrow idist(d, c)$
7: Insert key $iDistance_p$ and value p into T
8: **end for**
9: **end for**
10: **return** T

4.3.1 Query Processing

Algorithms 8-6 provide an overview of the iDistance k-Nearest Neighbors (KNN) search process. The iDistance KNN search algorithm operates similarly to a simple KNN search, described in Section 3.9.2. It starts by searching within a small sphere of radius r and incrementally enlarges the search space until k nearest neighbors are found. The search stops when the distance from the query point q to the most remote object in the answer set S, is less than or equal to the current search radius r. Before considering the main iDistanceKNN algorithm it is crucial to understand three key methods, namely *SearchInward*, *SearchOutward*, and *LocateLeaf*.

SearchInward inspects a leaf node's entries towards the left, determining their inclusion in the K nearest neighbors and updating the results as needed. Taking Figure 4.1 as an example, the *SearchInward* routine examines towards the left sibling in partition $P1$ as indicated by arrow A, while *SearchOutward* examines towards the right sibling as per arrow B. For partition $P2$, only *SearchInward* is used to examine towards the left sibling from $distmax_2$, following the direction of arrow *C*. *LocateLeaf*, a standard B+-tree traversal routine, identifies a leaf node based on a search value. Due to the similarities of the two algorithms, we will focus on explaining *SearchInward*. Algorithm 5 demonstrate SearchInward, and displays the relevant changes necessary for SearchOutward in parentheses. The original implementation performs a recursive nearest neighbor search within a partitioned dataset. The algorithm operates by utilizing a priority queue, PQ, the current node ce, a search limit $iValue$, the B+-tree T, a query object q and the current partition P_c. It returns the current node ce, which will be used to start the search from where we left off if the partition is not exhausted. At the start, the algorithm checks if the current node ce is the first entry of T. If this is not the case, the algorithm then

Algorithm 5 iDistance: Original SearchInward (SearchOutward)

Require: PQ, ce, $iValue$, T, q, P_c
Ensure: ce
1: **if** ce is not first (last) entry in T **then**
2: $score \leftarrow dist(ce, q)$
3: $w \leftarrow PQ_{top}$
4: **if** $score < w.score$ **then**
5: Add ce to PQ
6: **end if**
7: **if** $ce.distance > iValue$ ($ce.distance < iValue$) **then**
8: ce = MoveLeft(ce) (MoveRight(ce))
9: SearchInward(PQ, ce, $iValue$, T, q, P_c) (SearchOutward(PQ, cn, $iValue$, T,q), P_c)
10: **end if**
11: **if** $ce \notin P_c$ **then**
12: $ce \leftarrow T.end$
13: **end if**
14: **end if**
15: **return** ce

calculates the distance between the ce and the q, referred to as *score*. The algorithm then find the worst element in the priority queue, PQ_top. If the score of ce is better than PQ_top, the algorithm adds ce to the priority queue. Next, the algorithm examines whether the $dist(ce, P)$ is less than the search limit $iValue$. If it is, it means that we are still in the search annulus and so the algorithm shifts one entry to the left by calling the *MoveLeft* or *MoveRight* function. Following this, the algorithm recursively invokes itself with the updated parameters. When $iValue$ is reached, it checks whether ce is still within the current partition P_c. If we reached the end of the partition, the current entry is set to the endpoint of the tree structure $T.end$, marking the completion of the search within this partition. Finally, the algorithm returns ce as a pointer for further search later, or as a flag marking the partition P_c as exhausted.

Algorithm 6 show our new *SearchInward* implementation, which adopts an iterative approach. This can offer several benefits over the recursive approach outlined in the original iDistance paper. Recursive calls result in additional overhead because each recursive call results in the addition of a new layer to the stack, consuming memory. This overhead can be significant, especially for large datasets with high dimensionalities, ultimately causing slower execution times. In worst-case scenarios, excessive recursion can lead to stack overflow errors. An iterative approach avoids these issues. Each iteration operates within the same stack frame, rather than creating a new one for each recursive call. This translates to less memory usage, less overhead for function calls, and generally a faster execution time. It also reduces the risk of stack overflow errors. In the first version of our *SearchInward* implementation, we stored document identifiers in the lists

Algorithm 6 iDistance: New SearchInward (SearchOutward)

Require: cn, $iValue$, p
Ensure: nd
1: Initialize current node distance nd
2: **while** Current node cn is not first (last) node in T **do**
3: Update cn one entry to the left (right)
4: Update nd to be to the key value of cn
5: **if** cn is not a member of partition p **then**
6: End of partition, $cn \leftarrow$ end of T
7: Break loop
8: **else if** cn is member of partition p **then**
9: $score \leftarrow dist(cn, q)$
10: Add cn to PQ if $score$ is better than the score of w
11: **end if**
12: **if** $nd < iValue$ ($nd >= iValue$) **then**
13: Break loop
14: **end if**
15: **end while**
16: **return** cn

lp and rp. This proved itself to be highly inefficient, and we opted to store the document iterators for the B+-tree instead. An iterator is a pointer-like object that can be used to iterate over the elements of a container, like our B+-tree. When using an iterator, you can easily access the documents and their neighbors again without having to perform a new search operation. In contrast, using document identifiers requires us to perform a lookup operation every time we increase r we want to continue searching where we left off in *SearchInward* and *SearchOutward*.

Algorithm 7 operates using several parameters, the priority queue PQ, the query identifier id_q, the left and right pointer lists lp and rp, the boolean flag list $oflag$, the search radius r, and the stretch factor c. The priority queue PQ stores potential nearest neighbors. id_q represents the identifier of the query object q. The left and right pointers, lp and rp respectively, are used to navigate the tree structure during the search operation. $oflag$ is a boolean flag indicating whether a partition has been searched or not. r is the search radius and c is a constant used in the calculation of the iDistance.The algorithm commences by setting q to be id_q, representing the query point. It then iterates through each partition P_i of the partition set P. For each partition, it calculates the distance d_q from the reference point of the partition O_p to the query point q. The steps that follow depend on the state of $oflag_p$ (the operation flag for partition p). If $oflag_p$ is $FALSE$ it means that partition P_i has not been searched before, and we have to consider three different scenarios. First, if the query point, q, is within the partition P_i, we traverse the partition to locate the node where q is stored. We then conduct both *SearchInward* and *SearchOutward*. For instance, in the first iteration depicted in figure 4.1, we only

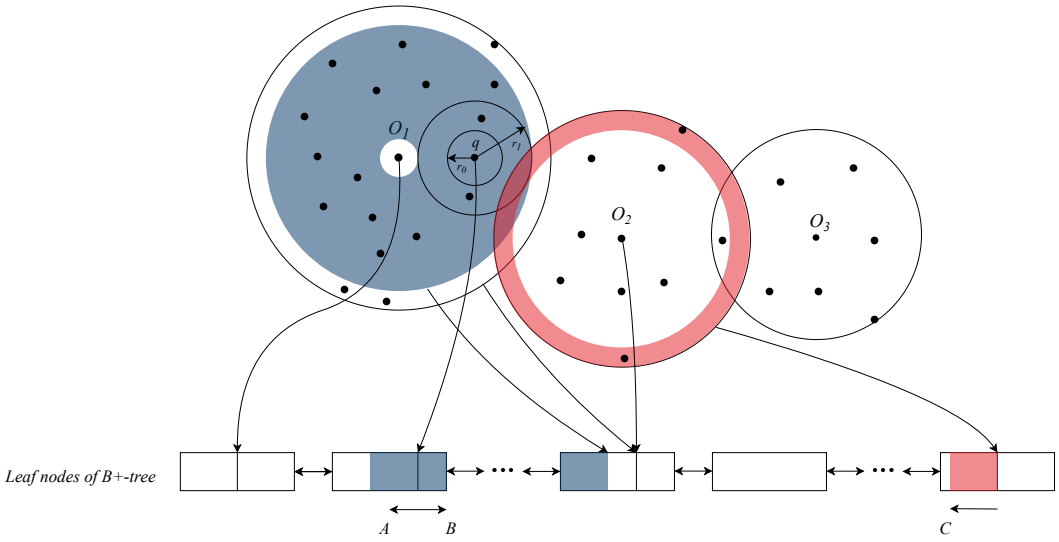


Figure 4.1: The iDistance Search Method

inspect partition $P1$, using q to navigate down the B+-tree. The second scenario arises when q is outside the partition, but the query sphere $sphere(q, r)$ intersects with the partition. Two spheres q and O_i intersects if $dist(q, O_i) < radius_q + radius_{O_i}$. In this situation, we only perform *SearchInward*. This is illustrated when $sphere(q, r)$ intersects with partition $P2$ in figure 4.1, which contains the reference point O_2. The third and final scenario is when the partition P_i doesn't intersect with the query sphere. In this case, there's no need to examine the partition, as shown with partition $P3$ in figure 4.1.

The main search Algorithm 8 runs Algorithm 7 with an increasingly larger search radius r. When the furthest object in the set S is within the current search radius r and there are at least k objects in the set, the K nearest neighbors have been identified. At this point, all other objects still not considered have a distance d from q larger than querydist. Consequently, any further expansion of the query sphere is unnecessary, guaranteeing 100% accuracy of the iDistance result.

4.4 Dual Spatio-textual iDistance

In the following Section, we introduce DualiDistance, our multi-dimensional extension of the iDistance algorithm designed for conducting efficient semantic similarity searches

Algorithm 7 iDistance KNN search algorithm: SearchO

Require: PQ, id_q, lp, rp, $oflag$, r, c, λ

$q \leftarrow id_q$
for each P_i of P **do**
  $d_q \leftarrow dist(O_p, q, \lambda)$
  if not $oflag_p$ **then**
    if $q \in P_i$ **then**
      $oflag_i \leftarrow TRUE$
      $iDistance_q \leftarrow idist(d_q, c, i)$
      $n \leftarrow LocateLeaf(T,\ iDistance_q)$
      $lp_i \leftarrow SearchInward(PQ, n,\ idistance_q - r,\ q,\ P_i)$
      $rp_i \leftarrow SearchOutward(PQ, n,\ idistance_q + r,\ q,\ P_i)$
    else if $sphere(O_p, distmax_p)$ intersects $sphere(q, r)$ **then**
      $oflag_i \leftarrow TRUE$
      $n \leftarrow LocateLeaf(T,\ distmax_i)$
      $lp_i \leftarrow SearchInward(PQ, n,\ idistance_q - r,\ q,\ P_i)$
    end if
  else
    if $lp_i \neq T.end$ **then**
      $lp_i \leftarrow SearchInward(PQ, lp_i,\ idistance_q - r,\ q,\ P_i)$
    end if
    if $rp_i \neq T.end$ **then**
      $rp_i \leftarrow$ SearchOutward$(PQ, rp_i,\ idistance_q + r,\ q,\ P_i)$
    end if
  end if
end for

on high-dimensional spatio-texutal data. Unlike the original iDistance, DualiDistance is capable of retrieving documents that meet both spatial and textual criteria. We also detail the integration of the threshold algorithm, which combines the dual indexes to find the optimal set of spatio-textual objects for a given query. Figure 4.2 depicts the separate dual search process and its subsequent combination.

4.4.1 Index Construction

Efficient search processes require a data structure that integrates spatial and textual aspects. Instead of performing spatial-fist indexing, we advocate for a joint strategy that does not favour one dimension since the spatial-centric approach seems to provide little information about their semantic relation to the query. To overcome this, we suggest a "dual" index that captures both the spatial and textual facets individually, finding objects that are alike in terms of both their spatial proximity and semantic similarity. The dual index also incorporates a weighting parameter, λ. λ provides the ability to adjust the emphasis between spatial proximity and textual similarity. The implementation of

Algorithm 8 iDistance KNN main search algorithm: iDistanceKNN

Require: id_q, k, T, λ, c, Δr
Ensure: S
1: $q \leftarrow id_q$
2: $S \leftarrow \emptyset$
3: $PQ(k) \leftarrow \emptyset$
4: $r \leftarrow 0$
5: $flag \leftarrow$ FALSE
6: lp, $rp \leftarrow T.end$
7: $oflag \leftarrow FALSE\ \forall P_i \in P$
8: **while** $flag$ is FALSE **do**
9: Increment r with Δr
10: Find the worst candidate w in PQ
11: **if** $dist(w, q, \lambda) <$ r and $|S|$ is k **then**
12: $flag$ is set to $TRUE$
13: **end if**
14: searchO(PQ, id_q, lp, rp, $oflag$, r, c, λ)
15: **end while**
16: **return** S

this parameter allows dynamic adjustments for each query without any changes to the underlying data structures.

In this Section we provide an outline of the dual index for DualiDistance as implemented in Algorithm 9. The algorithm starts with the initialization of two B+-trees, T_s and T_t, which will be used to store the spatial and textual indexes, respectively. The partitions, denoted by P_s and P_t, reference points, O_s and O_t for spatial and textual data respectively, and the maximum distances from the reference points to any point within the partitions $distmax_s$ and $distmax_t$, are derived from Algorithm 3. The next phase involves creating the iDistance indexes. For each partition in P_s and for each member data object p within the partitions, the algorithm computes the distance d_s from the reference point O_i for partition P_i to p. This distance is then used to calculate the iDistance value $iDistance_p$ using Equation 3.13. The pair $(iDistance_p, p)$ is subsequently inserted into the spatial index B+-tree T_s. This process is mirrored for the textual partitions P_t. Upon completion of these steps, the DualiDistance index construction process yields two B+-trees T_s and T_t that serve as independent spatial and textual indexes.

4.4.2 Threshold Algorithm For Dual Indexes

In the implementation of DualiDistance, we use the Threshold Algorithm (TA), presented in Algorithm 10, in the main query search function. The incorporation of TA into the DualiDistance main search function allows for the simultaneous exploration of both spatial proximity and textual similarity while maintaining computational efficiency by identifying and returning the top-k relevant documents as early as possible. The search

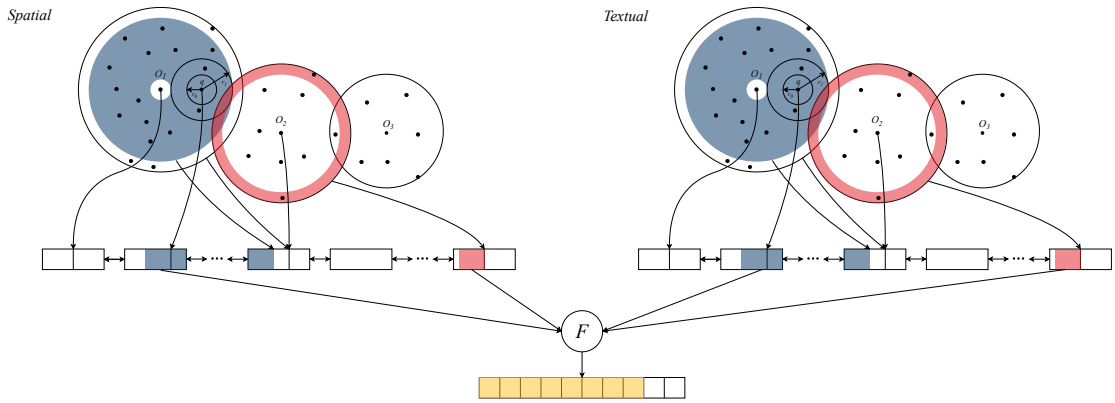


Figure 4.2: The DualiDistance Search Method

operation works by gradually enlarging a search radius r. At each step, it produces sets of documents that are sorted based on their spatial and textual distances relative to the query object q. These sets are then processed by TA to find the optimal k-nearest neighbors in relation to q. For every iteration over the search radius r, the spatial and textual search functions traverse their respective B+-trees to generate sorted sets of documents, *spatialDocs* and*textualDocs*. Each of the sets contains documents sorted by their spatial and textual distances from the query q. When Algorithm 10 is run, for every pair of current documents p_s and p_t, the algorithm gets their respective spatial and textual distances, and calculates for each their combined distance *score* as a weighted sum, using the weight parameter λ. If the calculated score, known as *score*, is among the top-k scores, the present document, either *spatialDoc* or *textualDoc*, is added to the priority queue PQ. Simultaneously, the threshold value t is dynamically updated, computed as the combined score of the last examined pair, p_s and p_t. The algorithm stops early and returns $TRUE$ when it encounters a pair of documents from *spatialDoc* and *textualDoc* whose combined scores is better than the value of $treshold$, and the worst document in PQ is closer to q than the current search radius r. If it is not able to terminate early, it does an exhaustive search and returns wether there exists k documents who is closer to q than r. The search process repeats, with the search radius incremented at each pass, until one of the steps above returns $TRUE$. Documents that have been assessed in previous iterations are tracked in a set *consideredDocuments*, thereby avoiding the need to recompute their scores in subsequent iterations.

4.4.3 Query Processing

Algorithm 11 outlines the query processing in DualiDistance. At the start, the algorithm sets up various variables for execution, such as the query identifier id_q, the left and right pointers for the spatial and textual search lp_s, rp_s, lp_t, rp_t, and the reference point flags for each section ($oflag_s$, $oflag_t$). It also initializes the search radius r and assigns a termination flag $flag$ to $FALSE$. Document sets for the spatial and textual

Algorithm 9 DualiDistance: Nearest Neighbor Index

Require: DB, K, c
Ensure: T_s, T_t
1: T_s, $T_t \leftarrow \emptyset$
2: P_s, O_s, $distmax_s \leftarrow$ K-Means(DB, K)
3: P_t, O_t, $distmax_t \leftarrow$ K-Means(DB, K)
4: **for** each $P_i \in P_s$ **do**
5: **for** each $p \in P_i$ **do**
6: $d_s \leftarrow dist(O_i, p)_s$
7: $iDistance_p \leftarrow idist(d_s, c)_s$
8: Insert key $iDistance_p$ and value p into T_s
9: **end for**
10: **end for**
11: **for** each $P_i \in P_t$ **do**
12: **for** each $p \in P_i$ **do**
13: $d_t \leftarrow dist(O_i, p)_t$
14: $iDistance_p \leftarrow idist(d_s, c)_t$
15: Insert key $iDistance_p$ and value p into T_t
16: **end for**
17: **end for**
18: **return** T_s, T_t

indexes $Docs_s$, $Docs_t$ are set up, along with a group of already evaluated documents $consideredDocuments$, a priority queue $PQ(k)$ to contain k candidate nodes, and the final result set S. In every iteration of the main loop, the search radius r is increased by a set value Δr. Following that, the functions $searchO_s$ and $searchO_t$ are called upon to identify candidate nodes in the spatial and textual index, respectively, within the present search radius. Once the candidate nodes are identified, the Threshold Algorithm is applied. This algorithm evaluates the considered candidate nodes from both the spatial and textual fields, including their distance from the query. If the threshold condition is achieved, the algorithm performs early termination. After the search ends, the top k elements in the priority queue are moved to the final result set S. Thus, the algorithm ultimately provides S, which comprises the exact k-nearest neighbors to the given query in terms of both spatial and textual distances.

4.4.4 Inter- and Intra-cluster Pruning

The DualiDistance algorithm employs two significant pruning strategies to improve search efficiency, namely inter-cluster pruning and intra-cluster pruning. The purpose of inter-cluster pruning is to reduce the number of clusters that need to be searched. Clusters that are distant from the query are eliminated from the search process. This is the case when the search sphere does not contain nor intersect with a partition sphere. This process significantly decreases the search space and can improve the query response

Algorithm 10 Threshold Algorithm

Require: $spatialDocs$, $textualDocs$, k, PQ, q, $consideredDocuments$, λ, r
Ensure: Top-k documents found or need to expand the search radius

```
1:  spatialIterator ← spatialDocs.begin
2:  textualIterator ← textualDocs.begin
3:  threshold ← 0
4:  while spatialIterator ≠ spatialDocs.end AND textualIterator ≠ textualDocs.end
    do
5:    spatialDoc ← spatialIterator
6:    if spatialDoc ∉ consideredDocuments then
7:      Insert spatialDoc into consideredDocuments
8:      score ← a * spatialDoc.score + (1 − a) * semanticScore(q, spatialDoc)
9:      Add to PQ if better: (spatialDoc, score)
10:   end if
11:   textualDoc ← textualIterator
12:   if textualDocument ∉ consideredDocuments then
13:     Insert textualDoc into consideredDocuments
14:     score ← λ * spatialScore(q, doc) + (1 − λ) * textualDoc.score
15:     Add to PQ if better: (textualDoc, score)
16:   end if
17:   threshold ← a * spatialDoc.score + (1 − a) * textualDoc.score
18:   if threshold ≥ PQ.top AND PQ.size ≥ k AND PQ.top < r then
19:     return TRUE
20:   end if
21:   Increment spatialIterator, textualIterator
22: end while
23: if PQ.size ≥ k AND PQ.top < r then
24:   return TRUE
25: else
26:   return FALSE
27: end if
```

Algorithm 11 DualiDistance: Nearest Neighbor Search

Require: id_q, k, T_s, T_t, λ, c, λr, Δr
Ensure: S
1: $q \leftarrow id_q$
2: $S, Docs_s, Docs_t, consideredDocuments \leftarrow \emptyset$
3: $PQ(k) \leftarrow \emptyset$
4: $r \leftarrow 0$
5: $flag \leftarrow$ FALSE
6: lp_s, $rp_s \leftarrow T_s.end$
7: lp_t, $rp_t \leftarrow T_t.end$
8: $oflag_s \leftarrow FALSE\ \forall P_i \in P_s$
9: $oflag_t \leftarrow FALSE\ \forall P_i \in P_t$
10: $r \leftarrow 0$
11: **while** $flag$ is FALSE **do**
12: Increment r with Δr
13: $searchO_s(Docs_s, id_q, lp_s, rp_s, oflag_s, r)$
14: $searchO_t(Docs_t, id_q, lp_t, rp_t, oflag_t, r)$
15: $flag = TresholdAlgorithm(Docs_s, Docs_t, k, PQ, q, consideredDocuments, \lambda, r)$
16: **end while**
17: Add the top k members of the priority Queues PQ to S
18: **return** S

time considerably. While inter-cluster pruning helps reduce the number of clusters that need to be considered, intra-cluster pruning is done inside each cluster that is being searched. It aims to reduce the number of data points that need to be searched within a cluster. Intra-cluster pruning operates on the premise that within a cluster, not every data point will be relevant to a given query. When performing a range query inside a cluster, inter-cluster pruning operates under the assumption that only data entries within a specific annulus of each cluster sphere, determined by the query radius, need to be examined. Figure 4.1 shows both types of pruning. Partitions $P1$ and $P2$ prunes all the colorless document, and is a result of intra-cluster pruning. $P3$, on the other hand, is an example of inter-cluster pruning since it is never considered in the search.

4.4.5 Motivations Driving DualiDistance

We found two main ways to address the limitations of existing methods described in Section 2.4. Firstly, high-dimensional word embeddings can be employed to overcome the limitations of low-dimensional semantic vectors and topic modeling techniques. High-dimensional embeddings preserve the semantic richness of textual data and capture nuanced semantic relationships more accurately. By utilizing these embeddings, semantic spatio-textual searches can be conducted with improved accuracy and contextual relevance. Secondly, a dual indexing strategy can be implemented to address the spatial-first bias observed in existing techniques. This approach involves separating the spatial and textual

indexes, ensuring equal emphasis on both dimensions during the indexing process. By treating the spatial and textual components independently, the indexing structure can avoid inherent biases and provide a balanced and effective approach to indexing large spatio-textual datasets.

During the initial stages of our search for inspiration in designing a novel similarity search algorithm, our attention was captivated by the iDistance technique. This choice was driven by the algorithm's unique approach to addressing various challenges regarding high-dimensional data. It is also used in some of the related work discussed in Section 2.3. The iDistance method accomplishes this by dividing the data space into separate regions based on reference points, and this partitioning reduces the portion of the dataset that needs to be examined in a search operation. Another key factor in our decision was the dimensionality reduction mechanism of iDistance. By converting multi-dimensional objects into one-dimensional values, iDistance simplifies the data structure and makes querying more efficient since the demand for costly distance calculations is reduced. This characteristic made iDistance an attractive foundation for our new algorithm, considering the complexity of our data.

Integrating search results from two separate indexes presents challenges, as our empirical experiments demonstrate that a naive approach would be computationally infeasible. We were then left with two viable options, namely the TA and Fagin's Algorithm (FA). Ultimately, we opted for the Threshold Algorithm (TA) as the preferred choice for implementing DualiDistance. This decision was primarily driven by the TA's ability to directly utilize the precomputed distance scores stored in the sorted sets of documents. Each object or document in our sorted spatial and textual lists already has a precomputed distance score from the main searches. The TA is able to take advantage of these precomputed scores by using them directly in its threshold calculation at each iteration. In contrast, FA would require either recalculating the distances or storing them in a separate data structure. This leads to significant additional computational costs, particularly in high-dimensional datasets. These factors make TA a superior choice compared to FA for our specific requirements.

The decision to make DualiDistance an exact method, rather than an approximate one, primarily stemmed from the importance of accuracy in retrieving the most relevant results in spatial-textual searches. Exact nearest neighbor methods ensure the returned results are indeed the closest to the query in the context of both spatial and textual distances, which is crucial in numerous applications. Approximate Nearest Neighbors (ANN) methods, on the other hand, trade off a degree of accuracy for increased efficiency. While this may be acceptable in some contexts, it can lead to sub-optimal results, particularly when high accuracy is of high importance. When it comes to comparing approximate nearest-neighbor methods to current approaches, the challenge lies in the lack of a universal standard to quantify their performance. The inherent trade-off between speed and accuracy in these methods makes it difficult to compare them directly to other techniques.

5 Experimental Evaluation

Chapter 5 presents a comprehensive assessment of the performance of our proposed method through a series of experiments. The Chapter begins with an overview of the experimental setup, including details about the coding platform, algorithms used, dataset, selected parameters, evaluation metrics, and the nature of the queries employed. We then conduct a comparative evaluation to examine the impact of changing key variables such as k and λ on performance. The latter part of the Chapter focuses on a sensitivity analysis, investigating the robustness of our method by varying parameters Δr and f. This experimental analysis serves to examine the effectiveness and efficiency of our approach over a range of parameters.

5.1 Experimental Setup

In this Section, we outline our experimental setup. We provide the methodologies and parameters used to assess our proposed algorithm to facilitate the replication of our results.

5.1.1 Code and Platform

The algorithms in this study are implemented in C++ using g++ version 12.2.0, while the data set preprocessing and partitioning are implemented using Python3. The experiments are run on a remote DigitalOcean Droplet, using Ubuntu 22.10. The droplet is connected to a local Visual Studio Code instance through an ssh connection. The local Visual Studio Code instance is run on a MacBook Pro M1. The droplet has 8GB Memory, 4 Intel Xeon 2nd Generation Scalable processors, and an 80 GB disk. The processors are clocked at a base frequency of 2.50 GHz and a max turbo frequency of 3.90 GHz.

5.1.2 Algorithms

The study evaluates the performance and effectiveness of five distinct indexing methods, including our proposed method DualiDistance. We first make use of a linear scan algorithm, fittingly called *Scan*. This algorithm computes distances between the query and all objects within the dataset. It is included due to its tendency to outperform index-based algorithms in high-dimensional spaces. Secondly, we employ an index-based algorithm known as the *R-tree*, which constructs an R-tree using the spatial coordinates of each object, thereby functioning as a purely spatial index. Lastly, our methods are benchmarked against the state-of-the-art algorithm *S2R* discussed in Section 2.3.

5.1.3 Dataset

The dataset used in this master's thesis is provided by our supervisor. It is composed of 500.000 geotagged tweets collected from across the United States. Each tweet object contains a high-dimensional word embedding and a spatial coordinate with x and y components. All tweets in the dataset are written in English and acquired using the public Twitter API[1], and their spatial coordinates have been normalized in $[0, 1] \times [0, 1]$. In terms of data preparation and index creation, the text from each tweet is transformed into a 100-dimensional semantic vector representation using the Glove model. To create a unique semantic vector for each tweet, the average of the individual word embeddings is computed. Words that do not feature in the predefined vocabulary or are recognized as common stop-words are excluded from this process. Furthermore, tweets comprising less than three words are discarded, as they are deemed to offer minimal value due to an overabundance of stopwords or unfamiliar terms. Regarding clustering, we leverage the K-Means algorithm to derive the cluster centroids, radii, and members. We are able to apply the K-Means algorithm on the whole dataset, thanks to its relatively small size. This approach ensures that every data object directly contributes to the formation of cluster centroids and radii, maximizing the representativeness and accuracy of our clustering. However, for future studies involving larger datasets, we may need to adjust this method due to computational considerations. In future scenarios, it would be more efficient to perform K-Means clustering on a representative sample of the data, and then assign the remaining data objects to the nearest clusters. This method still maintains reasonable accuracy while drastically improving computational efficiency. The implementations used for K-Means come from Python3's scikit-learn[17] library.

5.1.4 Parameters

The choice of parameters plays a crucial role in our work, with the number of clusters for a dataset being particularly significant. Over-clustering can lead to excessive granularity and fragmentation of data which can potentially inflate computational requirements. On the contrary, under-clustering can result in overly generalized groups, which could hamper pruning effectiveness. Hence, the challenge in applying clustering algorithms lies in striking a delicate balance between these extremes. Choosing the appropriate number of clusters is largely determined by the specific characteristics of the dataset and the overall objectives of the work. In our experiments, we determined the number of clusters using the formula

$$K_s = K_t = \sqrt{|O| \cdot 0.01 \cdot f}$$

This formula seems to yield a suitable number of clusters for various data set sizes. The multiplier f can be adjusted to further control the number of clusters, and we conducted experiments to evaluate different values of f. It is important to clarify that our objective is not to determine the exact optimal number of clusters for each attribute. Instead, our goal is to introduce a parameter that enables us to adjust the level of granularity in the

[1] https://developer.twitter.com/en/docs/twitter-api

resulting partitioning scheme. Beyond that, we alter the number of k-nearest neighbors, Δr, and the balancing parameter of the distance function λ. Table 5.1 encapsulates the values of the examined parameters.

Parameter	Values
Twitter Data set size $\|O\|$	500k
Dimensionality of the original space n	100
Number of nearest neighbors k	5, 10, 25, 50, 100
Δr	0 - 0.04, step: 0.005, def: 0.01
Multiplier f	0.1, 0.3, 0.5, 0.7, 0.9
Balancing parameter λ	0 - 1, step:0.1, def: 0.5

Table 5.1: Parameters ranges and their default values.

5.1.5 Metrics

In our experiments, we assess the performance of the algorithms in Section 5.1.2 using a set of metrics designed to evaluate their efficiency, adaptability, and computational overhead. These metrics provide insights into the practical applications of the algorithms and enable comparative analysis. We primarily assess the performance of each algorithm based on its query execution time. The query execution time is a measure of the time taken by an algorithm from query execution to the final result retrieval. A shorter execution time indicates a more efficient algorithm.

In addition to execution time, we evaluate the effectiveness of pruning strategies by tracking the total number of accessed objects during query execution. This metric indicates the number of documents examined or processed by the algorithms to retrieve the result. A lower value suggests more effective pruning. We also keep track of the number of objects accessed by the Threshold Algorithm (TA). This metric provides valuable insights into how many objects TA evaluates in order to stop the search early and identify the top-k relevant documents. By monitoring TA accesses, we gain a deeper understanding of its efficiency within the broader framework of DualiDistance. This information becomes particularly meaningful when compared to the total accesses made by DualiDistance. It indicates whether we retrieve an excessive number of documents before executing the TA, which could lead to unnecessary computational overhead. Ideally, we aim for a scenario where the number of retrieved objects closely aligns with the requirements of TA, avoiding the expenditure of resources on fetching and evaluating unnecessary documents. It is important to note that DualiDistance is an exact method, so there is no need to calculate or report an error rate. The algorithm produces exact results and ensures the retrieval of the optimal k documents.

5.1.6 Queries

In our experimental setup, we conduct a series of runs, each employing a distinct object from our dataset as a query. For each run, we select 20 objects at random, ensuring a wide variety of test conditions and a robust evaluation of the performance of each algorithm. This process helps to eliminate any potential biases that might be introduced by using a fixed set of query objects. For each query, we track a variety of metrics that provide us with insight into the algorithm's performance and efficiency. These metrics include the algorithm and query that is used, the number of nearest neighbors requested, and the weighting factor used to balance the importance of spatial and textual distances. We also measure several time-related metrics, such as the time taken for index construction, the query execution time, and the total time from indexing to result retrieval. To gauge the efficiency of our pruning strategy, we record the number of documents accessed during the execution and the number of documents pruned. The result values we present are the mean values from the 20 runs for each query.

5.2 Comparative Evaluation

We perform a comprehensive evaluation of our proposed method in comparison to established algorithms. This evaluation involves systematically varying key parameters, namely k and λ, to analyze their influence on the performance of each method. By conducting this assessment, we gain valuable insights into the strengths and weaknesses of our approach relative to existing solutions.

5.2.1 Varying k

By examining Figure 5.3 and 5.4, we can assess the influence of the number of retrieved nearest neighbors, k, on the performance of the Scan, R-tree, S2R-tree, and DualiDistance algorithms. These figures reveal distinct patterns and shed light on how different values of k affect the efficiency of these algorithms. The Scan algorithm, which compares every document in the dataset to a given query, exhibits consistent performance regardless of the value of k. This is expected, as its approach evaluates every document independently of k. As k increases, the performance gap between the scan algorithm and other algorithms diminishes. The Scan algorithm, which visits all documents, does not incur significant additional overhead when handling larger k values. As a result, its query time can even become competitive with other algorithms in such scenarios.

The R-tree, a spatial indexing algorithm, demonstrates a slight increase in the number of visited objects as k increases. However, its query time remains stable, indicating efficient pruning of non-relevant objects and the ability to retrieve a larger result set without significantly impacting query time. Surprisingly, the S2R-tree and R-tree exhibit nearly identical performances, raising doubts about the added effectiveness of the S2R-tree's semantic layer. Despite the additional complexity and theoretical advantages of the S2R-tree, their performances closely resemble each other. In contrast, DualiDistance shows an increase in both query time and the number of visited objects as k grows. Of

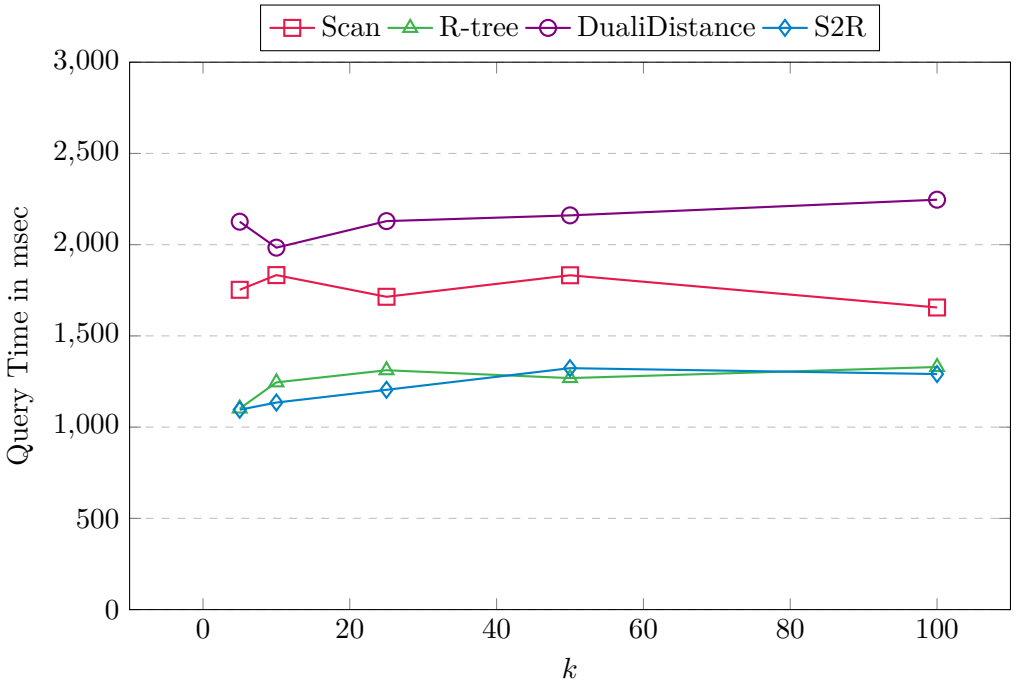


Figure 5.1: Varying k

concern is the discrepancy between the number of documents fetched by DualiDistance and the number of documents accessed by its threshold algorithm. This mismatch suggests an over-retrieval of documents, potentially leading to unnecessary computational overhead and impacting overall performance.

5.2.2 Varying λ

Figure 5.3 and 5.4 evaluate the effect of the balancing factor λ. λ serves as a control parameter that governs the balance between spatial and semantic search. Specifically, for $\lambda = 0$, the problem reduces to k-nearest neighbor (k-NN) search of high-dimensional semantic vectors, while for $\lambda = 1$, it is reduced to spatial k-NN. The density of documents in different dimensions plays a crucial role in the effect of varying λ. Spatial data exhibits a denser distribution, with many documents originating from the same or nearby locations. On the other hand, high-dimensional semantic vectors display greater sparsity and wider distribution. The Scan algorithm, known for its brute-force approach of evaluating every document in the dataset, maintains consistent query time and visits the same number of objects across different λ values. This is expected, as the Scan algorithm does not discriminate between spatial and semantic relevance. While this method may seem computationally heavy and less efficient for high λ values, where the density of spatial data can speed up queries, it performs surprisingly well for low λ values. The sparsity of semantic vectors becomes more dominant in this scenario, and the Scan algorithm's

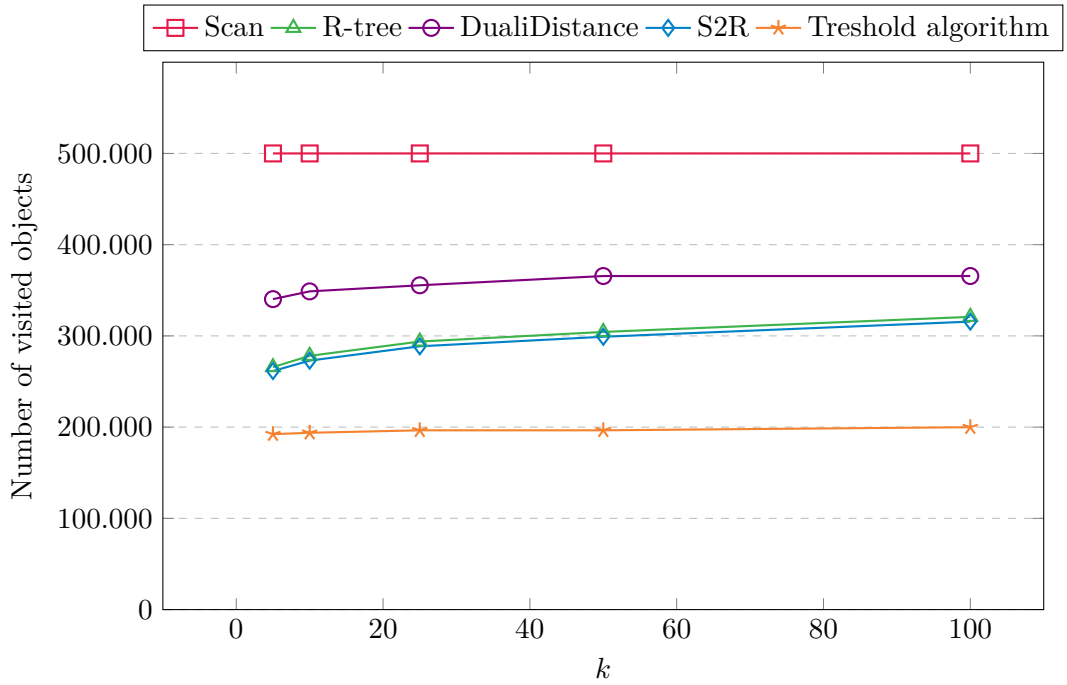


Figure 5.2: Varying k

exhaustive scanning strategy proves effective. This is consistent with earlier findings, where the Scan Algorithm becomes an increasingly viable choice as the dimensionality increases. Both the R-tree and S2R-tree algorithms exhibit notable reductions in query time and the number of visited objects as λ increases, transitioning from sparse semantic search to denser spatial search. These algorithms leverage spatial indexing to efficiently prune non-relevant regions of the search space, resulting in improved performance for higher λ values. The S2R-tree and R-tree algorithms heavily rely on spatial indexing to facilitate efficient spatial queries. However, as λ decreases and the problem becomes more semantically oriented, the effectiveness of spatial indexing diminishes. Notably, the performance of the S2R-tree algorithm, despite its increased complexity in handling semantics, closely resembles that of the purely spatial R-tree. This similarity raises doubts about the potential advantages of its semantic integration. DualiDistance possesses a distinctive characteristic, where the search and retrieval of documents are not dependent on λ. This approach, while providing consistency, also may lead to a consistently higher query time across different λ values. The complexity of managing dual indexes within DualiDistance may contribute to this behavior. Interestingly, DualiDistance consistently fetches almost twice as many documents as those accessed by its associated threshold algorithm. This suggests potential over-fetching and inefficiency in early pruning that warrant further investigation. The threshold algorithm in DualiDistance employs λ for computing the total document score and is an important factor when determining which documents should be part of the final result. The algorithm exhibits a significant decrease

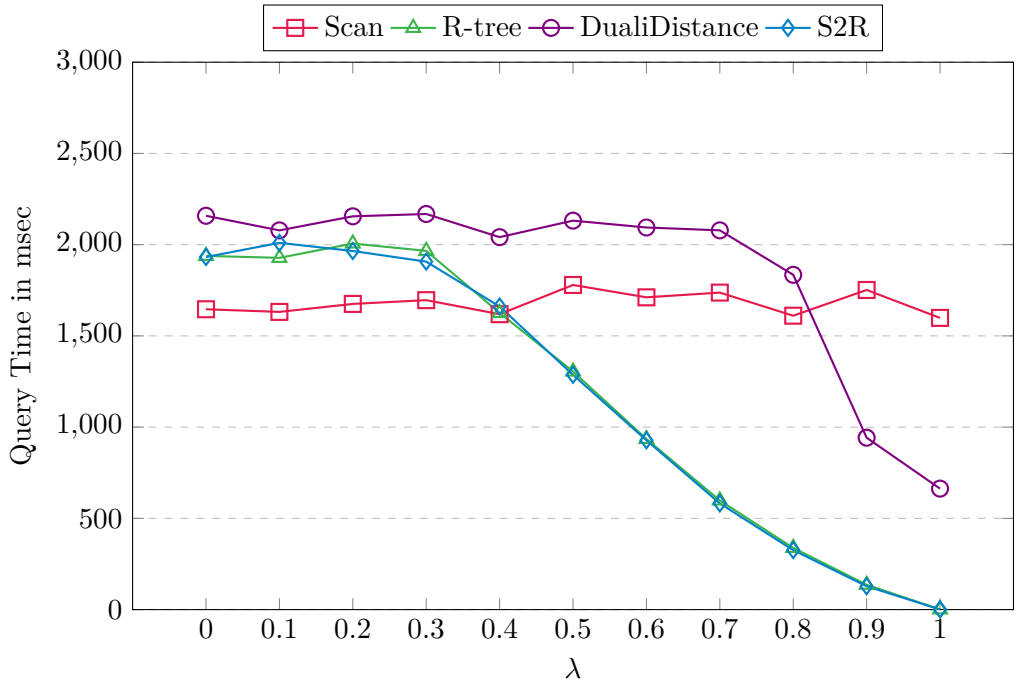


Figure 5.3: Varying λ

in the number of visited objects as λ transitions from 0 to 1. This reduction indicates the algorithm's proficiency in spatial search surpasses that of semantic search, reflecting the inherent challenges in exploring high-dimensional spaces. In datasets with spatially dense distributions, characterized by many documents sharing identical coordinates, the process of populating the priority queue with suitable candidates becomes notably easier as the value of λ increases. With a higher concentration of documents in the same spatial location, the search process may require fewer iterations, enabling the earlier identification of the top-k nearest neighbors. As λ decreases and the focus shifts to semantics, the influence of sparsity in high-dimensional semantic vectors becomes more significant.

5.3 Sensitivity Analysis

The following is a sensitivity analysis to explore the behavior of our proposed method DualiDistance under different conditions. By varying key parameters, specifically Δr and f, we investigate the impact of these variations on the overall performance of our model. This helps us understand the robustness and generalizability of our method across diverse applications.

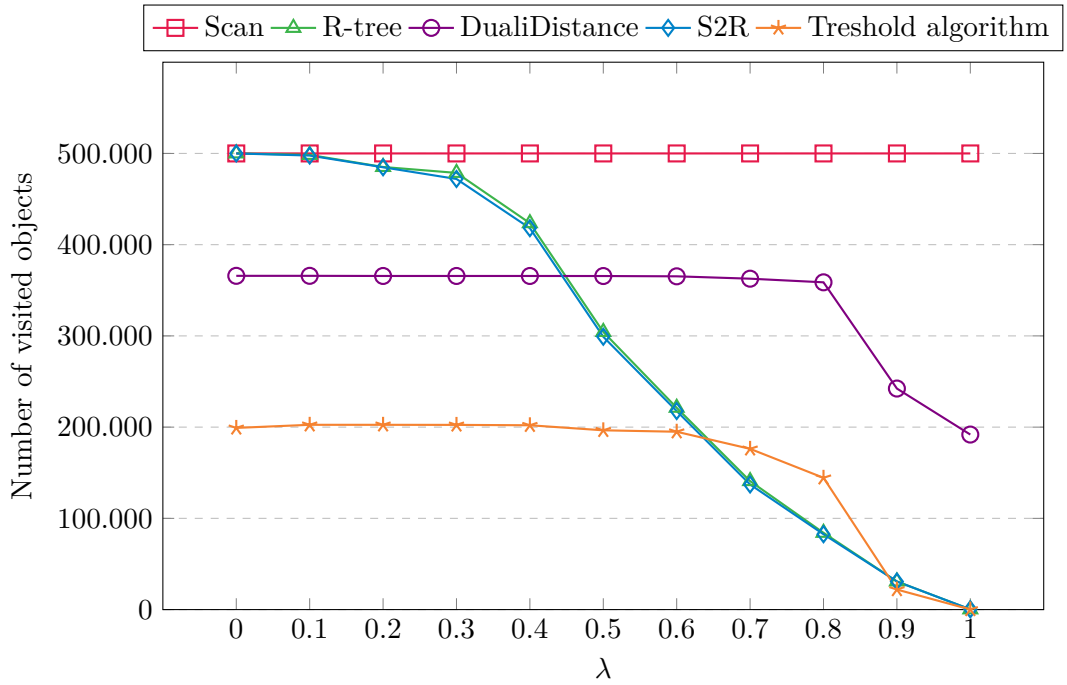


Figure 5.4: Varying λ

5.3.1 Varying Δr

The parameter Δr in our DualiDistance algorithm dictates the rate of the incremental extension of the search radius r. Its effect on the algorithm's performance is observable in two primary areas, namely the query time and the number of visited objects during the search process. Our sensitivity analysis for Δr is encapsulated in Figures 5.5 and 5.6. As shown in Figure 5.5, the query time exhibits a general trend to decrease with larger Δr values. This observation makes some intuitive sense considering the design of our algorithm. With a larger Δr, each step of the search covers a broader space, potentially reducing the time to reach the nearest neighbors. It may also reduce the number of iterations the main search loop needs to perform.

However, one must interpret these results with caution due to the behavior of our threshold algorithm implementation. Our threshold algorithm implementation Algorithm 10 has a termination condition in line 18 that is dependent on the search radius r, which is modulated by Δr. If Δr is set too high, the initial search radius becomes overly large at the start, which might lead to early termination of the search loop before confirming the exact k nearest neighbors. Therefore, a larger Δr might speed up the query time but could potentially compromise the accuracy of the search.

Moving onto Figure 5.6, we observe that the number of visited objects by the DualiDistance algorithm remains relatively consistent across varying values of Δr. The stability in the number of visited objects by the DualiDistance algorithm across different Δr values can be attributed to the algorithm's design. DualiDistance incrementally

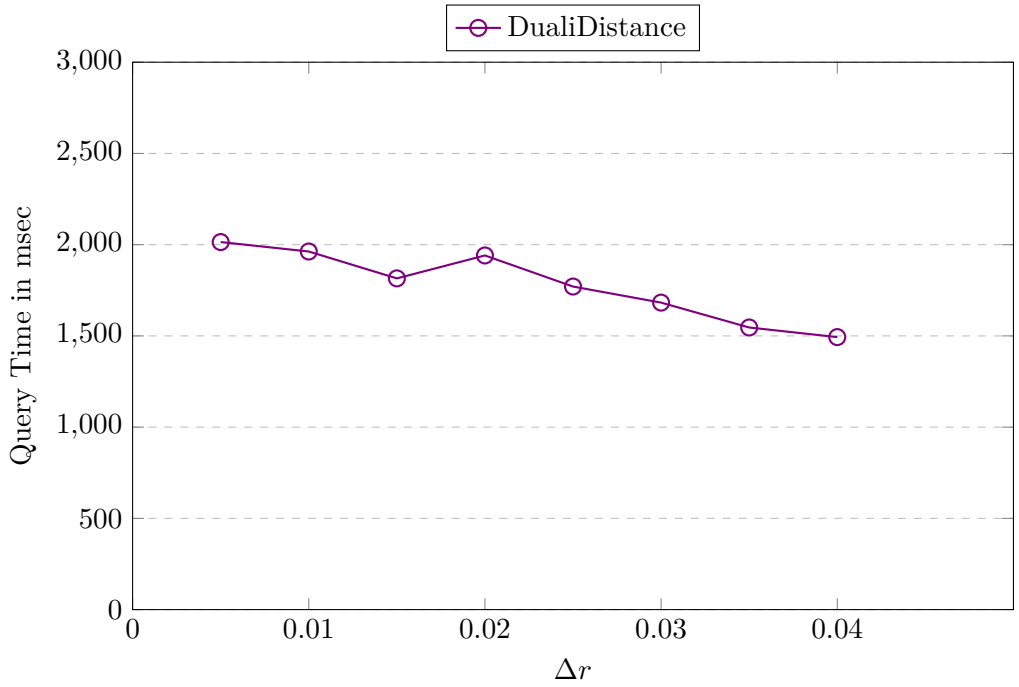


Figure 5.5: Varying Δr

expands the search radius r and explores the documents within that radius in each index. While a larger Δr allows the search to cover a wider area in each step, DualiDistance is specifically designed to find the exact k nearest neighbors. Consequently, the algorithm continues running until it has identified k documents that are closer to the query object than the current search radius. Regardless of the specific value of Δr, a similar set of documents may be likely to be considered in both indices, leading to a relatively constant count of visited objects. In contrast, the count of visited objects for the threshold algorithm declines significantly with larger Δr values. This suggests that with an enlarged initial search radius, the threshold algorithm can meet its termination condition sooner, thereby visiting fewer objects. However, as noted earlier, this increased efficiency might come at the expense of reduced search precision. Determining an appropriate value for Δr is a balancing act. On the one hand, a higher value can hasten the query time but may risk diminishing search accuracy due to premature termination of the threshold algorithm. On the other hand, a lower value, although preserving the integrity of the search, could lead to longer query times. Since we require the exact k-nearest neighbors, we need to choose a conservative value for Δr that ensures exact results at the cost of query performance.

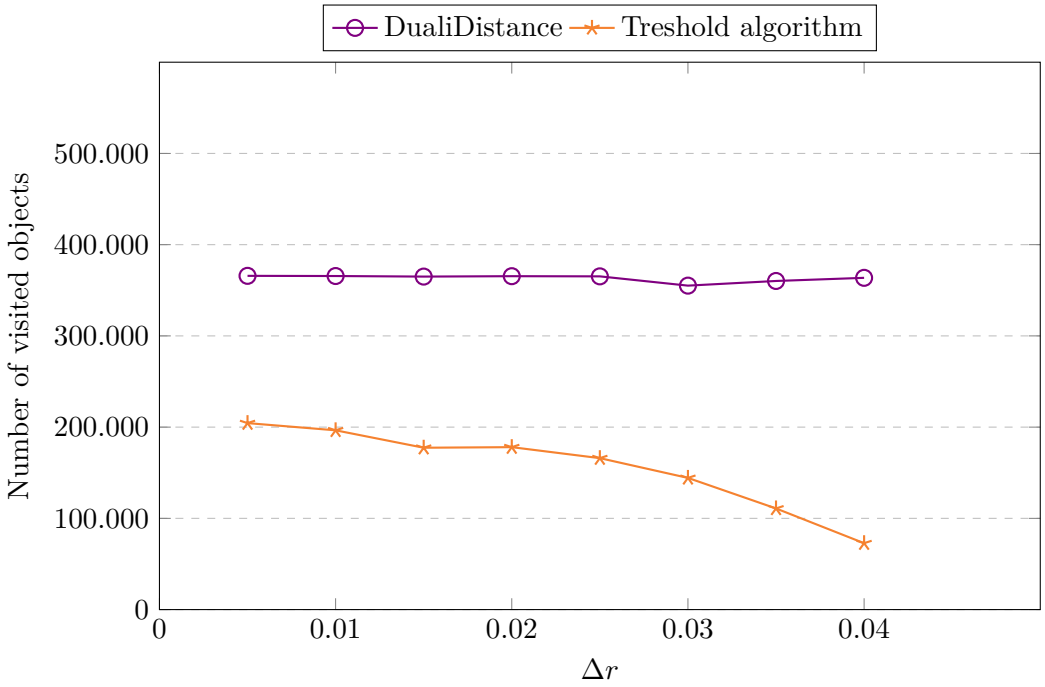


Figure 5.6: Varying Δr

5.3.2 Varying f

The parameter f only plays a significant role in the DualiDistance algorithm, which utilizes clustering to partition the data. As depicted in Figure 5.7, increasing f leads to an increase in query time. This behavior aligns with our understanding that higher f values generate a greater number of clusters, which adds complexity to the search process and hence extends the query time. An interesting observation is that the query time does not exhibit a linear increase but rather levels off at higher values of f in the DualiDistance algorithm. This suggests that the algorithm's design possesses a certain degree of resilience, effectively managing the complexity associated with a larger number of clusters. Meanwhile, as shown in Figure 5.8, the number of visited objects by DualiDistance also increases as f grows. DualiDistance partitions the data space into multiple clusters and, for each search query, the algorithm conducts a dual search in both spatial and textual index structures. As the number of clusters grows, the data space is divided into smaller segments. Even though each individual cluster becomes smaller, the total number of clusters the algorithm might need to traverse during a search increases. The algorithm doesn't access all documents in a cluster indiscriminately during its dual search in both spatial and textual index structures. It specifically examines the relevant documents in each cluster that belongs to the same cluster as the query, or those located within an annulus defined by the intersection of the query's search sphere and the cluster sphere. The necessity to navigate more clusters, induced by their increased number, also necessitates more annuluses to be searched. This can potentially lead to a larger number

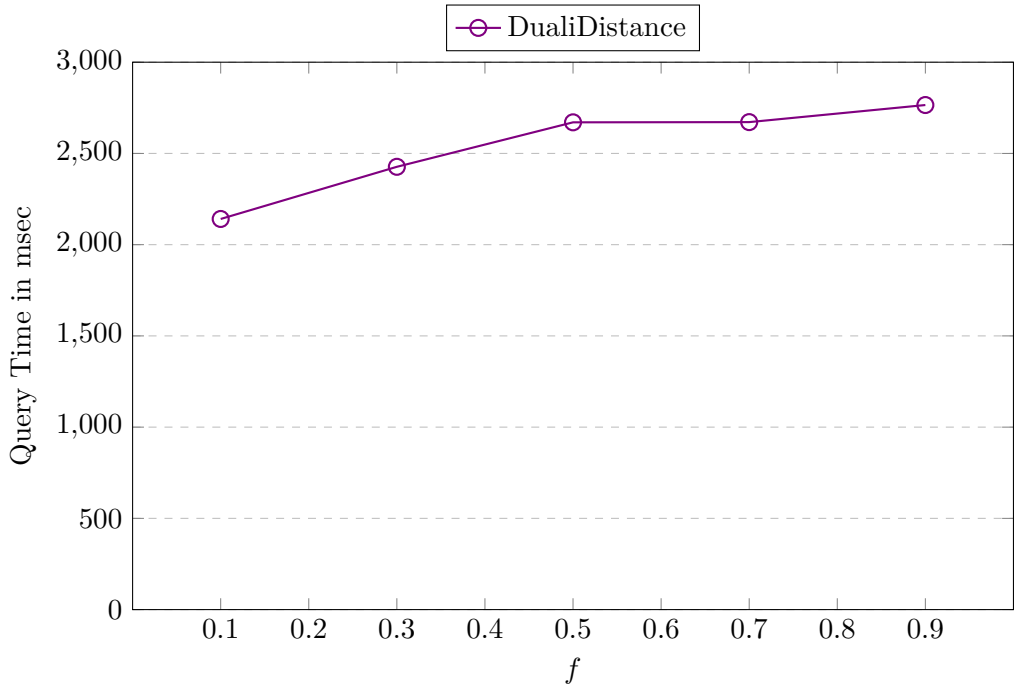


Figure 5.7: Varying f

of documents being considered. The transition process between searching each cluster can also incur an overhead leading to a performance dip.

Interestingly, Figure 5.8 demonstrates that the number of objects visited by the threshold algorithm seems to follow a different trend, decreasing as f increases. This divergence implies that the threshold algorithm may be more efficient at managing an increasing number of clusters. In the threshold algorithm, two sorted lists of spatial and textual distances are processed in parallel. Each list contains documents, sorted by their spatial or textual distance to the query document. At every iteration, the algorithm chooses the document closest to the query from each list, ensuring that it has not been previously examined. The threshold, which is derived from the distances of the two current documents, is then updated accordingly. When we increase the number of clusters, the documents within the data space become more finely partitioned. This tighter grouping of documents in each cluster leads to smaller distances between documents within the same cluster, which means that the documents in the sorted lists could have lower scores. Therefore, sorted lists containing documents with lower scores may lead to a situation where the threshold value becomes greater than the combined distance score of the worst candidate in the result set faster. This may allow the algorithm to more efficiently prune irrelevant documents whose combined distances are greater than the threshold. Consequently, the number of accessed objects would decrease as the number of clusters increases.

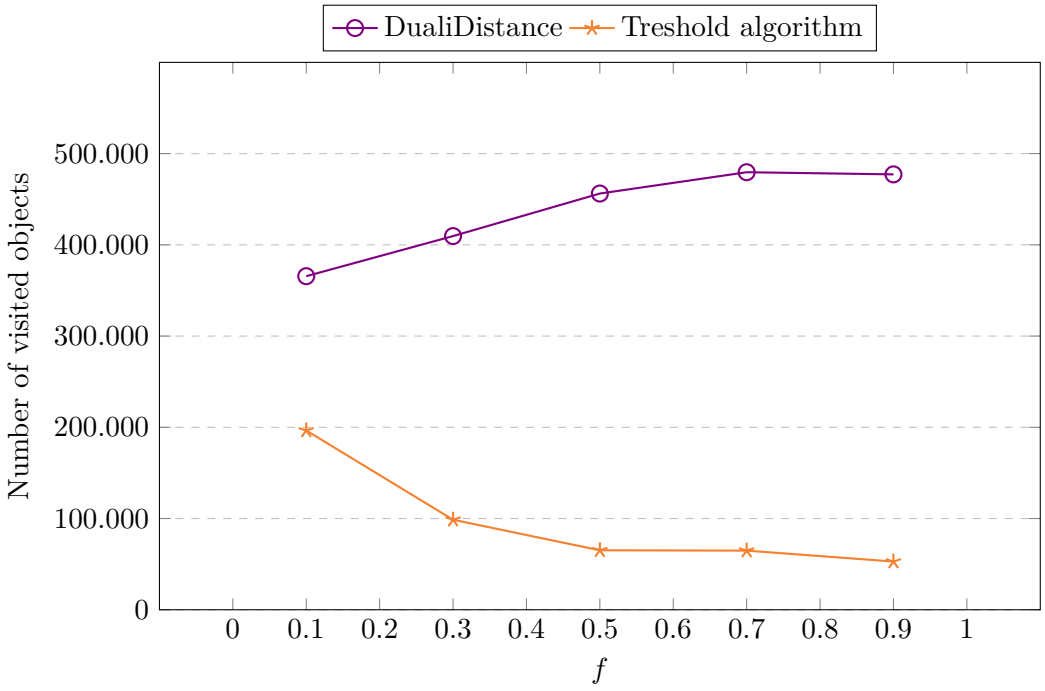


Figure 5.8: Varying f

6 Conclusion and Future Work

6.1 Conclusion

This thesis set out with the aim of improving the performance of semantic similarity searches over spatio-textual data. To achieve this objective, we focused on developing new indexing and searching methods that could effectively manage and extract valuable insights from large-scale spatio-textual datasets. Throughout this research journey, we pursued answers to four core research questions, and in this conclusion, we will assess how our endeavors have fared against the original goals and questions.

Goal *Develop indexing and searching methods that improve the performance of similarity searches in spatio-textual data.*

We identified significant limitations in existing semantic spatio-textual search methods, including spatial-first bias and limited semantical capabilities. To address these challenges, we proposed the utilization of high-dimensional word embeddings and a dual indexing strategy, resulting in the development of the DualiDistance method. However, in practical experiments, DualiDistance did not outperform existing methods and introduced new complexities such as document over-retrieval and computational load for larger numbers of clusters. While we feel our efforts to innovate and implement a new method point in the right direction, the expected performance improvements were not fully realized. Nevertheless, these findings may provide some insight for future work in semantic spatio-textual search techniques.

Research question 1 *What are the limitations of existing indexing and searching techniques for spatio-textual data, and how do they impact the efficiency and effectiveness of similarity searches?*

Through our literature review, we identified the key limitations of existing indexing and searching techniques for spatio-textual data. One key limitation of these methods is the dominance of the spatial dimension in their indexing structures. When the spatial dimension dominates the indexing process, the effectiveness is comparable to that of indexing methods that only consider the spatial domain, such as a basic R-tree. Another limitation is that current approaches predominantly rely on low-dimensional semantic vectors or topic modeling techniques to capture the semantic meaning of queries. Our experiments also showed that the additional low-dimensional semantic layer in the S2R-tree did not provide any clear performance advantage over the simpler R-tree. Topic modeling approaches provide a probabilistic interpretation of semantic information but

may struggle to capture the fine-grained nuances of individual semantic contexts. Lastly, incorporating user feedback to enhance semantic relevance introduces scalability issues and potential inconsistencies due to subjective interpretations.

Research question 2 *How can novel indexing and searching methods be designed to address the challenges associated with semantic spatio-textual search?*

To address the limitations of existing spatio-textual data indexing and searching techniques, two main concepts should be adopted. Firstly, high-dimensional word embeddings can better capture semantics and context in queries than low-dimensional semantic vectors or topic modeling techniques. Secondly, a dual indexing strategy that separates spatial and textual indexes can mitigate the spatial-first bias in current methods. This balanced approach could enhance accuracy, context relevance, and efficiency in managing large spatio-textual datasets.

Research question 3 *What performance improvements can be achieved by designing new algorithms, and how do they compare to existing methods in terms of retrieval efficiency and effectiveness?*

To address Research Question 3, we proposed the DualiDistance method, which integrates the design ideas suggested in response to Research Question 2. DualiDistance utilizes high-dimensional word embeddings to provide a more comprehensive and context-aware representation of semantics in spatio-textual data. Moreover, it employs a dual indexing strategy with separate indexes for spatial and textual data, avoiding the spatial-first bias inherent in existing methods. Despite these promising design choices, the performance improvements of DualiDistance did not outperform the existing methods in our experiments. Several reasons may explain this outcome. DualiDistance tends to retrieve more documents than the threshold algorithm needs to stop the search process early. When the number of clusters increases, DualiDistance requires more computational resources to handle the increased complexity. There might be ways to change the implementation so that higher amounts of clusters do not incur a significant dip in performance. Lastly, the choice of the Δr parameter represents a trade-off between speed and accuracy. A higher value can speed up query time but may compromise search accuracy due to premature termination of the algorithm. Since we require exact results, we had to set the value of Δr conservatively.

6.2 Future Work

The findings and experiences of this study reveal several potential directions for future research aimed at improving the performance of semantic spatio-textual search.

One important direction for further exploration is the investigation of alternative indexing techniques, whether unified or dual, to address the spatial-first bias observed in existing methods. While the dual indexing approach employed by DualiDistance shows promise, there is potential for the development of innovative indexing methods that

not only balance the spatial and textual dimensions but also leverage their interplay to achieve efficient pruning early in the search process.

Additionally, exploring alternative techniques for semantic representation in spatio-textual data is a worthwhile pursuit. While high-dimensional word embeddings offer rich semantic representations, their computational complexity due to high dimensionality poses challenges. Research into alternative methods that provide equally robust semantic capabilities without the computational load could greatly enhance spatio-textual data search performance.

Lastly, refining the DualiDistance method itself holds the potential for improving its effectiveness. Addressing issues such as document over-retrieval and computational load associated with increasing cluster numbers is a challenge worth tackling. Optimizing the stopping criteria of the threshold algorithm or exploring new techniques for early stopping could potentially enhance performance without compromising result accuracy.

Bibliography

Bayer, R., & McCreight, E. (1970). Organization and maintenance of large ordered indices. *Proceedings of the 1970 ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control*, 107–141.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, *18*(9), 509–517.

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *The Journal of Machine Learning Research*, *3*(Jan), 993–1022.

Chen, L., Cong, G., Jensen, C. S., & Wu, D. (2013). Spatial keyword query processing: An experimental evaluation. *Proceedings of the VLDB Endowment*, *6*(3), 217–228.

Chen, L., Shang, S., Yang, C., & Li, J. (2020). Spatial keyword search: A survey. *GeoInformatica*, *24*(1), 85–106.

Chen, X., Xu, J., Zhou, R., Zhao, P., Liu, C., Fang, J., & Zhao, L. (2020). S2R-tree: A pivot-based indexing structure for semantic-aware spatial keyword search. *GeoInformatica*, *24*(1), 3–25.

Chen, Z., Chen, L., Cong, G., & Jensen, C. S. (2021). Location- and keyword-based querying of geo-textual data: A survey. *The VLDB Journal*, *30*(4), 603–640.

Cong, G., Jensen, C. S., & Wu, D. (2009). Efficient retrieval of the top-k most relevant spatial web objects. *Proceedings of the VLDB Endowment*, *2*(1), 337–348.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [arXiv:1810.04805 [cs]].

Fagin, R. (2002). Combining fuzzy information: An overview. *ACM SIGMOD Record*, *31*(2), 109–118.

Finkel, R. A., & Bentley, J. L. (1974). Quad trees a data structure for retrieval on composite keys. *Acta Informatica*, *4*(1), 1–9.

Guttman, A. (1984). R-trees: A dynamic index structure for spatial searching. *ACM SIGMOD Record*, *14*(2), 47–57.

Jagadish, H. V., Ooi, B. C., Tan, K.-L., Yu, C., & Zhang, R. (2005). iDistance: An adaptive B+-tree based indexing method for nearest neighbor search. *ACM Transactions on Database Systems*, *30*(2), 364–397.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. University of California Press.

Mahmood, A., & Aref, W. (2019). Scalable Processing of Spatial-Keyword Queries. *Synthesis Lectures on Data Management*, *11*, 1–116.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space [arXiv:1301.3781 [cs]].

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., & Thirion, B. (2011). Scikit-learn: Machine Learning in Python. *The Journal of Machine Learning Research*, *12*, 2825–2830.

Pennington, J., Socher, R., & Manning, C. (2014). GloVe: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1532–1543.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep Contextualized Word Representations. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, *1*, 2227–2237.

Qian, Z., Xu, J., Zheng, K., Sun, W., Li, Z., & Guo, H. (2016). On Efficient Spatial Keyword Querying with Semantics. *Database Systems for Advanced Applications*, 149–164.

Qian, Z., Xu, J., Zheng, K., Zhao, P., & Zhou, X. (2018). Semantic-aware top-k spatial keyword queries. *World Wide Web*, *21*(3), 573–594.

Sun, J., Xu, J., Zheng, K., & Liu, C. (2017). Interactive Spatial Keyword Querying with Semantics. *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 1727–1736.

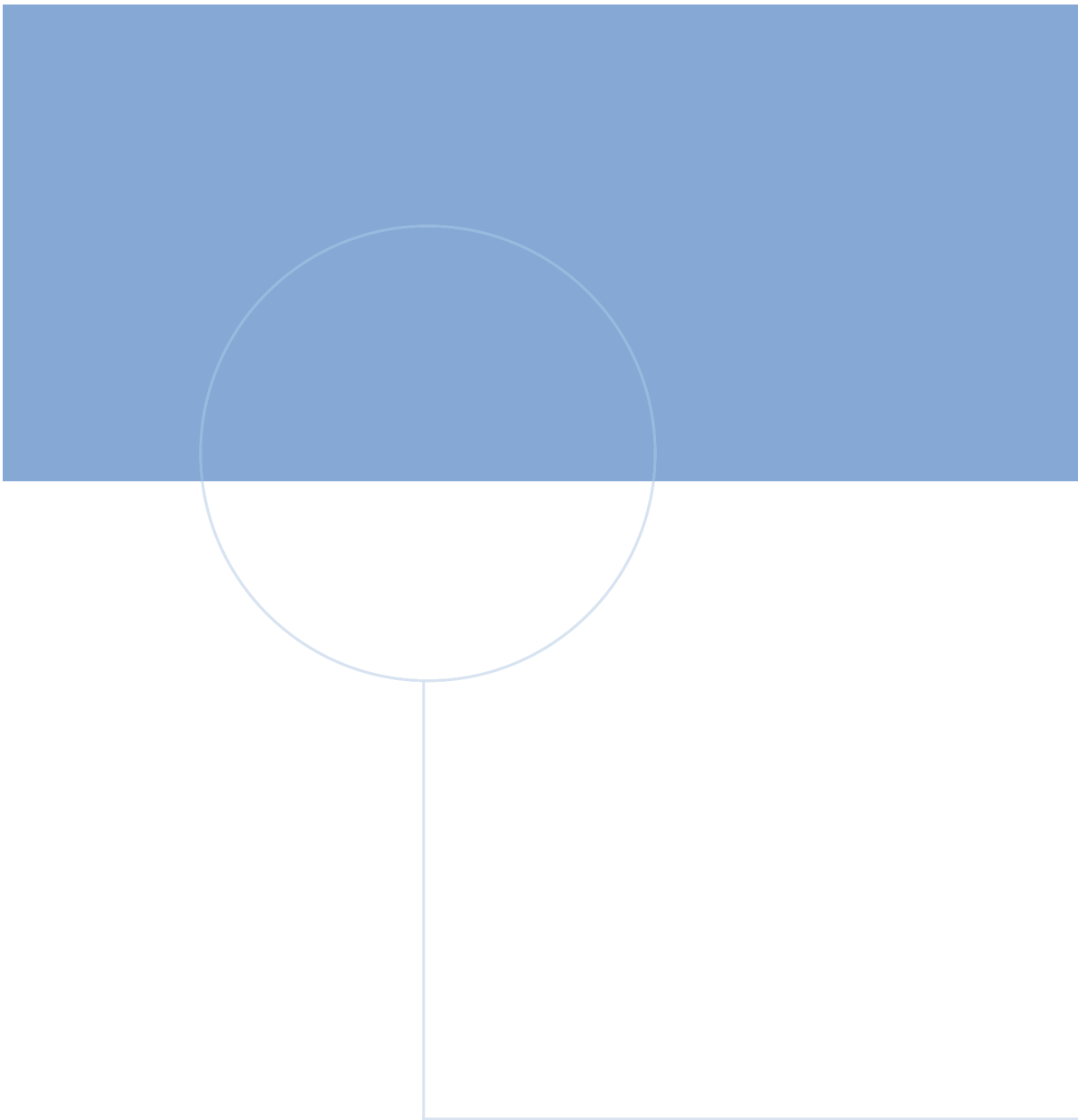
van der Maaten, L., & Hinton, G. (2008). Viualizing data using t-SNE. *Journal of Machine Learning Research*, *9*, 2579–2605.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 6000–6010.

Weiner, P. (1973). Linear pattern matching algorithms. *Proceedings of the 14th Annual Symposium on Switching and Automata Theory (swat 1973)*, 1–11.

Wu, D., Cong, G., & Jensen, C. (2012). A framework for efficient spatial web object retrieval. *The VLDB Journal*, *21*.

Yao, B., Li, F., Hadjieleftheriou, M., & Hou, K. (2010). Approximate string search in spatial databases. *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, 545–556.



NTNU

Norwegian University of
Science and Technology