Explanatory Search and Exploration of Spatial Entities

Kalliopi Basiakou

Master Thesis

— ♦ —

Ioannina, June 2024

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF IOANNINA

Explanatory Search and Exploration of Spatial Entities

A Thesis

submitted to the designated
by the Assembly
of the Department of Computer Science and Engineering
Examination Committee

by

Kalliopi Basiakou

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN DATA AND COMPUTER SYSTEMS ENGINEERING

WITH SPECIALIZATION
IN DATA SCIENCE AND ENGINEERING

University of Ioannina

School of Engineering

Ioannina 2024

Examining Committee:

- **Nikolaos Mamoulis,** Professor, Department of Computer Science and Engineering, University of Ioannina (Supervisor)
- **Panos Vassiliadis,** Professor, Department of Computer Science and Engineering, University of Ioannina
- **Apostolos Zarras,** Professor, Department of Computer Science and Engineering, University of Ioannina

Dedication

I would like to dedicate this thesis to my family.

ACKNOWLEDGMENTS

I would like to express my graditude to my advisor, Nikos Mamoulis, for giving me this opportunity and the support, he offered me through all the time, as my supervisor. I would also like to thank Dimitris Tsitsigkos for his support and encouragement throughout the course of my research. I appreciate the time you dedicated to providing thoughtful feedback and guidance, which has been incredibly beneficial to my work. Additionally, I would like to like to extend my sincere thanks to George Fakas, for his support and expertise. His strategic vision and organizational skills were crucial in deciding what we needed to do and ensuring that the research progressed smoothly. I am also deeply grateful to my family for always supporting me and having faith in me through all the years. Your unwavering belief in my abilities has been a constant source of strength and motivation. To my friends, thank you for always being there for me. Your companionship and encouragement have been invaluable throughout this journey. Thank you all for your mentorship and for believing in me. Your combined support has not only made this thesis possible but has also equipped me with the knowledge and skills necessary for my future endeavors.

Ioannina, June 2024

Kalliopi Basiakou

CONTENTS

LIST OF FIGURES

List of Tables

ABSTRACT

Kalliopi Basiakou, M.Sc. in Data Science and Engineering, Department of Computer Science and Engineering, School of Engineering, University of Ioannina, Greece, June 2024
Thesis Title: Explanatory Search and Exploration of Spatial Entities
Advisor: Nikolaos Mamoulis, Professor

When retrieving information based on geographic locations (location-based retrieval), it's important not just to consider where the objects are located but also to take into account additional descriptive information or context associated with those objects. This is especially important when the search results include a large number of objects, which can be overwhelming for the user.
This research focuses on developing methods to find and extract geographical objects within specific regions, using object summaries constructed from large data collections. These object summaries, except from ids and names of objects (contextual data), contain detailed information about their locations (geospatial information). Points of interest (POIs) are examples of such spatial entities and can include locations like restaurants, parks, landmarks, or any other significant places. Furthermore, this project examines the challenge of selecting a subset of query results that best represents the entire set. We propose that objects with similar context and close proximity should be proportionally represented in the selection. The project focuses on selecting a smaller, more manageable group of results from the larger set. These selected results should be both relevant and proportionally distributed in terms of spatial and contextual attributes, ensuring they are meaningful and provide a balanced mix of different locations and descriptions. The ultimate goal is to display a diverse subset of objects on the map, enhancing the user's ability to see a varied and pertinent range of results. To achieve this, a grid-based algorithm is employed, optimizing the process of spatial proportionality by dividing the spatial domain into a grid. Additionally, a random sampling algorithm is used to select a representative subset of spatial objects, maintaining spatial and contextual diversity by leveraging

randomness. The project also utilizes two algorithms for result selection post-grid or baseline algorithms: Greedy and Greedy-DisC. The first ensures diversity and relevance through a greedy heuristic and the second ensures coverage and dissimilarity among selected items. These algorithms power a web application where users input queries into a search box, and relevant points are dynamically shown on a map. The object summary created, with the query as the root, dynamically updates based on the current map bounds, ensuring users always see the most pertinent information for their area of interest. This functionality combines user-friendly search capabilities with the powerful visualization features of Google Maps, providing an intuitive and interactive experience. Overall, this project significantly enhances the retrieval and visualization of spatial and contextual data, making it easier for users to find and understand relevant information.

Εκτεταμενη περιληψη

Καλλιόπη Μπασιάκου, Μ.Δ.Ε. στην Επιστήμη Δεδομένων και Μηχανική, Τμήμα Μηχανικών Πληροφορικής και Μηχανικών Υπολογιστών, Πολυτεχνική Σχολή, Πανεπιστήμιο Ιωαννίνων, Ελλάδα, Ιούνιος 2024
Τίτλος Διπλωματικής Εργασίας: Επεξηγηματική Αναζήτηση και Εξερεύνηση Χωρικών Οντοτήτων.
Επιβλέπων: Νικόλαος Μαμουλής, Καθηγητής

Όταν ανακτούμε πληροφορίες με βάση γεωγραφικές τοποθεσίες είναι σημαντικό να λαμβάνουμε υπόψη όχι μόνο πού βρίσκονται τα αντικείμενα αλλά και τις επιπλέον περιγραφικές πληροφορίες ή το πλαίσιο που σχετίζεται με αυτά τα αντικείμενα. Αυτό είναι ιδιαίτερα σημαντικό όταν τα αποτελέσματα αναζήτησης περιλαμβάνουν μεγάλο αριθμό αντικειμένων, τα οποία μπορεί να είναι υπερβολικά για τον χρήστη. Με την ενσωμάτωση πληροφοριών σχετικά με το πλαίσιο στο οποίο ανήκει ένα ερώτημα, η διαδικασία ανάκτησης μπορεί να παρέχει πιο ουσιαστικά και διαχειρίσιμα αποτελέσματα, βελτιώνοντας την ικανότητα του χρήστη να βρει αυτό που ψάχνει αποδοτικά.

Η έρευνα αυτή εστιάζει στην ανάπτυξη μεθόδων για την εύρεση και εξαγωγή γεωγραφικών αντικειμένων εντός συγκεκριμένων περιοχών, χρησιμοποιώντας περιλήψεις αντικειμένων (Object Summaries) που κατασκευάζονται από μεγάλες συλλογές δεδομένων. Αυτές οι περιλήψεις αντικειμένων, εκτός από τα αναγνωριστικά και τα ονόματα των αντικειμένων (περιγραφικά δεδομένα), περιέχουν λεπτομερείς πληροφορίες σχετικά με τις τοποθεσίες τους (γεωχωρικές πληροφορίες). Τα σημεία ενδιαφέροντος (POIs) είναι παραδείγματα τέτοιων χωρικών οντοτήτων και μπορούν να περιλαμβάνουν τοποθεσίες όπως εστιατόρια, πάρκα, αξιοθέατα ή άλλες σημαντικές τοποθεσίες.

Επιπλέον, αυτό το έργο εξετάζει την πρόκληση της επιλογής ενός υποσυνόλου αποτελεσμάτων ερωτήματος, το οποίο αντιπροσωπεύει καλύτερα ολόκληρο το σύνολο. Προτείνουμε ότι τα αντικείμενα με παρόμοιο πλαίσιο και κοντινή απόσταση θα πρέπει να εκπροσωπούνται με ίδια αναλογία στην επιλογή. Η εργασία αυτή εστιάζει στην επιλογή μιας μικρότερης, πιο διαχειρίσιμης λίστας

αποτελεσμάτων από το μεγαλύτερο σύνολο. Αυτά τα επιλεγμένα αποτελέσματα πρέπει να είναι τόσο σχετικά όσο και αναλογικά κατανεμημένα, όσον αφορά τις χωρικές και περιγραφικές ιδιότητες, διασφαλίζοντας ότι έχουν κάποιο νόημα και παρέχουν ποικιλία διαφορετικών τοποθεσιών και περιγραφών. Με αυτό τον τρόπο μπορεί να αποφευχθεί η υπερβολική επιβάρυνση των χρηστών με πολλά παρόμοια αποτελέσματα, διευκολύνοντάς τους να βρουν ποικίλες και σχετικές με το ερωτημά τους πληροφορίες.

Για να επιτευχθεί αυτό, χρησιμοποιείται ο αλγόριθμος Grid, βελτιστοποιώντας τη διαδικασία της χωρικής αναλογικότητας διαιρώντας τον χωρικό τομέα σε πλέγμα. Επιπλέον, χρησιμοποιείται ο Random Sampling αλγόριθμος με σκοπό την επιλογή αντιπροσωπευτικού υποσυνόλου χωρικών αντικειμένων, διατηρώντας τη χωρική και περιγραφική ποικιλία μέσω της τυχαιότητας. Στην εργασία χρησιμοποιούνται επίσης δύο αλγόριθμοι για την τελική επιλογή αποτελεσμάτων που θα εμφανιστούν στο χάρτη, οι οποίοι εφαρμόζονται μετά τους αλγόριθμους Grid ή Baseline, και είναι ο Greedy και ο Greedy-DisC. Ο πρώτος διασφαλίζει ποικιλία και συνάφεια μέσω μιας άπληστης ευρετικής μεθόδου και ο δεύτερος διασφαλίζει κάλυψη και ανόμοιότητα μεταξύ των επιλεγμένων αντικειμένων.

Αυτοί οι αλγόριθμοι χρησιμοποιούνται σε μια διαδικτυακή εφαρμογή στην οποία οι χρήστες εισάγουν ερωτήματα σε ένα πλαίσιο αναζήτησης και τα σχετικά σημεία εμφανίζονται δυναμικά σε έναν χάρτη. Η περίληψη αντικειμένου (Object Summary) που δημιουργείται, με το ερώτημα ως ρίζα, ενημερώνεται δυναμικά με βάση τα τρέχοντα όρια του χάρτη, διασφαλίζοντας ότι οι χρήστες βλέπουν πάντα τις πιο σχετικές πληροφορίες για την περιοχή ενδιαφέροντός τους. Αυτή η λειτουργία συνδυάζει φιλικές προς τον χρήστη δυνατότητες αναζήτησης με τις ισχυρές δυνατότητες οπτικοποίησης των Χαρτών Google, παρέχοντας έτσι, μια διαδραστική εμπειρία. Εν κατακλείδι, αυτή η διπλωματική εργασία ενισχύει σημαντικά την ανάκτηση και οπτικοποίηση χωρικών και περιγραφικών δεδομένων, καθιστώντας ευκολότερο για τους χρήστες να βρίσκουν και να κατανοούν σχετικές πληροφορίες.

CHAPTER 1

INTRODUCTION

In the first section of this chapter, we present a brief description of our work and refer to the main directions and the main purpose of our research. In the second section of this chapter, we refer to the structure of this Thesis.

1.1 Goal

In the era of big data, the efficient retrieval and representation of information are critical for effective data management. With the increasing volume and complexity of data, particularly geospatial data, there is a growing need for methods that can provide users with clear and concise summaries of relevant information. This is especially pertinent in location-based retrieval systems where users search for information based on geographic locations. The challenge lies not only in considering the geographical locations of objects but also in accounting for the contextual information associated with these objects. When search results yield a large number of objects, it can overwhelm users, necessitating the development of methods that can distill and present the most pertinent information effectively.
To address this challenge, this thesis introduces the concept of Object Summary (OS), a concise representation of data about a particular Data Subject (DS). By presenting a clear and efficient overview of relevant data, OS enables users to interact with and understand the underlying information without needing to examine its full, detailed description. Additionally, the OS are pruned depending on the region input, ensuring

that only the most relevant and contextually appropriate data is included based on the user's query and location.

The goal of this project is to develop a novel exploration and explanatory paradigm for spatial data retrieval, specifically targeting the retrieval of places relevant to a queried entity within a specified region. The motivation behind this study stems from the vast availability of public and private datasets associated with locations, such as semantic knowledge graphs (e.g., YAGO, DBpedia), geosocial networks (e.g., Facebook, Foursquare), and points of interest tagged with textual descriptions (e.g., Google Places). The output will be the k most relevant places about the queried entity within the specified region, incorporating relevant nodes surrounding the entity node in the data graph. This approach will enhance usability by allowing users to explore important places that may not include the query keywords but are still highly relevant.

Additionally, it is important that the retrieval of places considers their spatial distribution to provide a fair and representative subset of places within a region. This approach will address the issue of relevance-only based retrieval, which can sometimes lead to a less informative or biased representation of places. By implementing proportionality techniques, the project seeks to facilitate regional fairness and prevent biases, ensuring a balanced representation of places from different areas. The baseline and grid algorithms discussed in this thesis are designed to achieve efficient and effective spatial proportionality. Moreover, selection algorithms play a critical role in enhancing user experience and maintaining clarity in data representation. By strategically displaying a subset of places on a map, we ensure that points are not clustered too closely together, reducing visual clutter and improving readability. The Greedy and Greedy-DisC algorithms discussed in this thesis are designed to ensure diversity and relevance, and construct a diverse subset, maximizing coverage and dissimilarity among the selected items.

To demonstrate the practical application of the concepts and algorithms discussed in this thesis, a web application was developed. This tool allows users to enter a keyword and view relevant nodes displayed on a map. The nodes form an object summary, with the keyword as the root, and dynamically update based on the map's current view. The application utilizes the algorithms discussed in this

thesis to ensure the displayed results are relevant, diverse, and proportionally representative of the spatial data. It features an intuitive interface that supports zooming, panning, and interactive markers, providing users with a comprehensive and engaging way to explore data. The responsive design ensures accessibility across various devices, making it a versatile tool for visualizing and interacting with spatial information.

1.2 Thesis Structure

This thesis is structured into eight chapters, each detailing different aspects of the research and development process:

In section 2, we review the related work and provide the background necessary for understanding the context of this thesis.

Section 3 delves into Object Summaries, explaining their concept, construction, and providing an algorithm for their creation. An example structure of a generated Object Summary is also presented.

Section 4 focuses on Spatial Proportionality, introducing its concept and discussing various algorithms designed to achieve it. The Baseline Algorithm and Grid Algorithm are explained in detail, including their implementation and advantages. We also compare the Baseline with the Grid Algorithm and explore a Random Sampling Algorithm for spatial proportionality.

Section 5 covers Selection Algorithms, emphasizing their importance and detailing the implementation of the Greedy Algorithm and Greedy-Disc Algorithm. An example use case is provided to illustrate the application of the Greedy-Disc Algorithm.

In section 6, we present a comprehensive analysis of experiments conducted to evaluate the performance of the algorithms and parameters discussed.

Chapter 7 describes the development and functionality of the web application that implements the work on spatial proportionality and diversity of results.

Finally, Chapter 8 concludes the thesis, summarizing the findings and discussing potential future work.

CHAPTER 2

RELATED WORK

There is a variety of other related work on object (entity) summarisations. For example [1] addresses the challenge of information overload in entity linking by proposing a method to create compact, structured summaries of entity descriptions. To avoid overloading human users with too much information, the authors aim to substitute entire entity descriptions with concise, effective summaries that maintain the quality of entity linking. The paper introduces three summarization approaches: characteristic summaries, which select features based on their ability to uniquely characterize each candidate entity; differential summaries, which prioritize features that differentiate candidate entities from each other; and contextual summaries, which select features relevant to the context of the entity mention using a class vector model. These perspectives are combined into a comprehensive summarization method that balances various aspects of entity description. Experimental results showed that the combined approach allowed users to link entities with accuracy comparable to full descriptions but with reduced time, highlighting its effectiveness in facilitating user decisions. Another work [2] discusses object summaries in the context of interactive entity resolution, where the goal is to select a subset of critical features from entity descriptions to be shown and judged by human users. The proposed method, C3D+P, aims to generate these compact summaries effectively. The features preferred for selection in the summaries are those that reflect the most commonalities shared by and the most conflicts between the two entities, as well as those that carry the largest amount of characteristic and diverse information about them. The paper emphasizes that these selected features are then grouped and ordered to improve readability and speed up the judgment process. The experimental

results demonstrate that summaries generated by this method help users judge more efficiently and accurately compared to entire entity descriptions. The method also outperforms existing summarization techniques by specifically focusing on the requirements of the entity resolution task, thus generating more useful and informative object summaries. Building on the concept of generating useful and informative object summaries, another study, [3] introduces Object Summaries (OS) as a novel result format for keyword searches in relational databases. An OS is designed to provide a comprehensive summary of data related to a specific Data Subject (DS) by creating a tree structure with the keyword-containing tuple at the root and related tuples as children. The paradigm liberates users from the need to know database schemata or query languages, instead relying on the concept of Affinity to determine the relevance of surrounding data. Affinity scores for relations and attributes help decide what to include in the OS, ensuring that only semantically meaningful data is presented. The paper highlights that this approach produces more complete and useful search results compared to traditional relational keyword search (R-KwS) methods, which often return disjointed tuples or require multiple keywords to form meaningful associations. Experimental evaluations on databases like TPC-H and Northwind showed high precision and recall, validating the effectiveness of the proposed method. The OS format was preferred by users for its self-contained and easily comprehensible presentation of information, making it a significant improvement over existing methods like précis queries, which can be harder to interpret due to their narrative presentation and lack of automated Affinity calculation. However, none of this work addresses spatial aspects of the data.

To further enhance the utility of object summaries, [4] presents methods for ranking object summaries (OSs) in response to keyword searches in relational databases. The authors propose a model that ranks OSs based on their relevance to thematic keywords, combining Information Retrieval (IR) properties, authoritative ranking using ObjectRank, and affinity, which measures the closeness of tuples to the data subject (DS) tuple. The thematic ranking is modeled as a top-k group-by join problem (kGBJ), which computes the join paths between identifying and thematic tuples without fully generating the OSs. Two main approaches are discussed: the Bi-Directional (BD) Approach, which computes complete OSs and ranks them,

and the Optimized kGBJ Approach, which focuses on relevant join paths and uses precomputed bounds to limit the search space. The methods were evaluated on DBLP and TPC-H datasets, demonstrating high precision and recall. The optimized kGBJ approach significantly outperformed the baseline BD method, showing up to 180 times faster performance in some cases. This thematic ranking model effectively addresses the challenges of ranking OSs in large datasets and ensures that users receive the most relevant OSs in response to their queries, however, the plain use of IR has limitations when applied to data graphs in general. Namely, they miss relevant nodes that are related to the keywords but they do not contain them [5]; e.g. the node Parthenon has relevance to Pericles although it does not include the word "Pericles". Our work, by selecting places from the object summary, addresses this problem.

Shifting the focus to the spatial dimension of data retrieval, various types of spatial-keyword queries have been proposed before. Spatial keyword search on datasets involves retrieving data objects based on both their geographical location and textual content. Such queries are Boolean kNN, top-k kNN, and Boolean range queries. A Boolean kNN query retrieves the k nearest objects to a user's current location that contain all specified keywords. The top-k kNN query, on the other hand, retrieves the k objects with the highest ranking scores, considering both their distance to the query location and the relevance of their text descriptions to the query keywords. Finally, the Boolean range query retrieves all objects within a specified spatial region whose text descriptions contain all the specified keywords. These indices typically use the R-tree or its variations, such as the R*-tree, to combine spatial and textual data efficiently for spatial keyword queries, where each minimum bounding rectangle keeps the textual information of all objects inside its bounds. However, these methods only search for individual objects that contain the specified keywords and do not retrieve relevant places that lack the keywords but are still pertinent to the query. Therefore, their direct application is not suitable in this context.

In addition to optimizing spatial keyword searches, numerous studies have explored different aspects of fairness. Spatial data fairness, as defined in the paper [6], addresses the unique challenges of ensuring equitable treatment in location-based applications where decisions are influenced by individuals' whereabouts. This

concept aims to prevent discrimination based on location data, which often correlates with sensitive attributes like race, income, and education. The paper introduces two main types of spatial fairness: distance-based fairness, relevant in scenarios like location-based advertising and ride-hailing, ensures individuals are not unfairly treated based on their proximity to a reference point, and zone-based fairness, which focuses on fairness in spatial coordinates, applicable in gerrymandering, loan analysis, and insurance pricing. To achieve these fairness goals, the paper proposes "fair polynomials," which adjust decision-making processes to ensure equitable treatment without significantly sacrificing data utility. Expanding the scope of fairness in data, [7] explores the concept of fair clustering under the disparate impact doctrine, emphasizing the need for approximately equal representation of each protected class within every cluster. This approach addresses the potential for machine learning algorithms to amplify existing biases present in training data. The authors introduce the idea of fairlets, minimal sets that ensure fair representation while maintaining clustering objectives, and show that fair clustering problems can be decomposed into finding good fairlets followed by traditional clustering algorithms. Although finding optimal fairlets is NP-hard, efficient approximation algorithms based on minimum cost flow are proposed. The empirical results on real-world datasets demonstrate that traditional clustering methods often yield unfair clusters, while fair clustering methods, though potentially more costly, maintain balanced solutions. The document also highlights the computational challenges associated with fair clustering, indicating that ensuring fairness introduces a significant computational bottleneck. Apparently, our work is different as we study the selection of a subset of objects instead of their clustering.

Concerning result diversification in information retrieval, the [8] discusses various methodologies aimed at enhancing the diversity and relevance of retrieved results. It highlights several algorithms, notably the Maximal Marginal Relevance (MMR) algorithm, which balances relevance and diversity by penalizing redundancy. Additionally, the Submodular Function Maximization method is examined for its efficient approach to diversification through submodular functions. The k-Nearest Neighbor (k-NN) approach is also mentioned, focusing on diversifying results by selecting items based on their dissimilarity to already chosen ones. These

comparisons provide a comprehensive understanding of the different strategies in result diversification. We used the greedy-disc algorithm because it effectively balances dissimilarity and coverage, offering a practical and robust solution for our specific diversification needs.

CHAPTER 3

OBJECT SUMMARIES

Chapter 3 provides an in-depth exploration of Object Summaries. It begins with Section 3.1, which introduces the concept of Object Summaries, detailing their purpose and significance. Section 3.2 discusses the process of constructing Object Summaries, including the specific algorithm used for their creation. Finally, Section 3.3 presents an example structure of a generated Object Summary, illustrating the practical application of the concepts discussed in the chapter.

3.1 About Object Summaries

An Object Summary (OS) is a concise representation of all data held in a database about a particular Data Subject (DS). The purpose of an object summary is to provide a clear and efficient overview, allowing users to understand and interact with the object or entity without needing to examine its full, detailed description. It is generated as a response to a query search and is structured as a tree, with the DS as the root node and its related nodes as children. Given the input node ("Pericles"), we start traversing the dataset and add on the object summary as child nodes the nodes surrounding the entity node (via edges/links) (e.g. Wife: Aspasia, Built: Parthenon, etc). The OS paradigm is particularly user-friendly for those accustomed to web keyword searches, providing a comprehensive summary that aids in data exploration and schema extraction.

3.2 Object Summaries Construction

To create an Object Summary, we first need to establish data structures to store and manage different types of data. Each map serves a specific purpose, facilitating the efficient creation of the OS:

- **nodeMap and nodeMapReverse:** Map node IDs to names and vice versa for easy lookup.
- **keywordsMap and keywordsMapReverse:** Map keyword IDs to keywords and vice versa for easy lookup.
- **edgesMap:** Store the relationships between nodes.
- **keywordsListMap:** Store lists of keywords associated with each node.
- **places:** Store geographic information for place nodes.

Using these mappings, we build the OS by traversing the relationships between nodes. We include nodes and their relationships up to three hops away from the root node. The OS is generated using a breadth-first traversal starting from the root node, adding nodes to the OS based on their relationships. The hierarchical structure is formed by enqueuing child nodes and adding them as children of the current node being processed. The tree structure ensures that the most important and representative nodes are included, maintaining the context and relationships between them. The importance and affinity of nodes are considered when constructing the tree, ensuring that nodes higher in the tree are more important and have a greater affinity to the root node.

A node in the OS can either be a place or a node without latitude and longitude values. Only nodes with latitude and longitude values, designated as places, are displayed on the map. After constructing the tree, we perform a pruning step to ensure relevance and accuracy. Nodes that have latitude and longitude values outside the specified map bounds are removed. This pruning step helps in maintaining the

geographical relevance of the data and ensures that the final OS only includes nodes within the desired map bounds.

This approach ensures that the OS provides a comprehensive and contextually rich summary of the data related to the DS, making it easier for users to explore and understand the underlying data structures.

Here, we can see the Data structures we need in tables.

Table 1: Node Mapping: Bidirectional Maps for Node IDs and Node Names

nodeMap	**nodeMapReverse**
map node IDs to their corresponding names	map node names to their corresponding IDs
nodeMap<Integer, String>:	nodeMapReverse<String, Integer>:
• Key: Node ID (Integer)	• Key: Node Name (String)
• Value: Node Name (String)	• Value: Node ID (Integer)
• Function: This map is used to retrieve the name of a node given its ID.	• Function: This map is used to retrieve the ID of a node given its name.

Table 2: Keyword Mapping: Bidirectional Maps for Keyword IDs and Keywords

keywordsMap	**keywordsMapReverse**
map keyword IDs to their corresponding keywords and vice versa.	map keyword IDs to their corresponding keywords and vice versa.
keywordsMap<Integer, String>:	keywordsMapReverse<String, Integer>:
• Key: Keyword ID (Integer)	• Key: Keyword (String)
• Value: Keyword (String)	• Value: Keyword ID (Integer)
• Function: This map is used to retrieve the keyword given its ID.	• Function: This map is used to retrieve the ID of a keyword given its text.

Table 3: Node Relationship Storage: Mapping Node IDs to Connected Node IDs

edgesMap
store the edges (relationships) between nodes.
edgesMap<Integer, int[]>:
• Key: Node ID (Integer)
• Value: Array of connected node IDs (int[])

• Function: This map is used to retrieve the IDs of nodes that are directly connected to a given node.

Table 4: Keyword Association Storage: Mapping Node IDs to Lists of Keyword IDs

keywordsListMap
store lists of keywords associated with each node.
keywordsListMap<Integer, int[]>:
• Key: Node ID (Integer)
• Value: Array of keyword IDs (int[])
• Function: This map is used to retrieve the list of keywords associated with a given node.

Table 5: Geographic Location Storage: Mapping Node IDs to Place Information

places
Purpose: To store place information for nodes that represent geographic locations.
places<Integer, PlaceObject>:
• Key: Node ID (Integer)
• Value: PlaceObject instance
• Function: This map is used to store and retrieve geographic information (like latitude and longitude) for nodes that represent places.

3.2.1 Algorithm for Object Summary Creation

In this section, we present Algorithm 3.2.1: Object Summary Creation, which outlines the process for creating an Object Summary (OS). The following pseudo details the steps required to traverse the dataset, identify relevant nodes, and construct the OS efficiently.

Algorithm 3.2.1 Object Summary Creation

```
1:  Function CreateObjectSummary(keyword)
2:  Initialize tempNeighbors, tempNeighbors2, tempNeighbors3 as null
3:  Initialize neighbors, neighbors2, neighbors3 as null
4:  Initialize neighborsArray as empty ArrayList
5:  Set initialNode = nodeMapReverse.get(keyword)
6:  If initialNode is null
```

```
7:      Set initialNode = keywordsMapReverse.get(keyword)
8:      If initialNode is null
9:          Return
10: If initialNode is a keyword
11:     ProcessKeyword(initialNode)
12: Else
13:     ProcessNode(initialNode)
14: Clear temporary maps: edgesMap, keywordsListMap, places
15: Function ProcessKeyword(initialNode)
16: Set nodeKeyword = keyword
17: Set nodeId = initialNode
18: Initialize neighborsArray
19: For each entry in keywordsListMap
20:     If entry contains nodeId
21:         Add entry key to neighborsArray
22: If neighborsArray is not empty
23:     Initialize tempNeighbors as ObjectInterface[neighborsArray.size()]
24:     For each parentNode in tempNeighbors
25:         Set neighbors2 = edgesMap.get(parentNode) // second hop
26:         ProcessSecondHopNeighbors(parentNode)
27: Function ProcessSecondHopNeighbors(parentNode)
28: Set nodeKeyword = keyword
29: Set nodeId = initialNode
30: Set neighbors = edgesMap.get(initialNode) // first hop
31: If neighbors is not empty
32:     Initialize tempNeighbors2 as ObjectInterface[neighbors.length]
33:     For each neighborNode in neighbors
34:         ProcessThirdHopNeighbors(neighborNode)
35: Function ProcessThirdHopNeighbors(parentNode)
36: Set nodeKeyword = keyword
37: Set nodeId = initialNode
38: Set neighbors = edgesMap.get(parentNode) // third hop
```

```
39: Initialize tempNeighbors3 as ObjectInterface[neighbors3.length]
40: If neighbors is not empty
41:     For each nodeObj in neighbors
42:         If nodeObj is a place
43:             PlaceObject placeObject = places.get(nodeId)
44:             PlaceObject newPlaceObject = new PlaceObject(placeObject.getId(),
    new Point(coordinates.getPoint().getLat(), coordinates.getPoint().getLon()),
    placeObject.getName(), placeObject.getType(), nodeMap.get(initialNode), 3)
45:             tempNeighbors3[i] = placeObject
46:         Else
47:             OtherObject otherObject = new OtherObject(parentNode2Id, temp-
    Neighbors3, nodeKeywords, nodeMap.get(parentNode2Id), 2)
48:             tempNeighbors3[i] = null
49:     ProcessThirdHopNeighbors(parentNode)
50: Function createRoot(initialNode, tempNeighbors)
51: If nodeKeywords = keywordsListMap.get(initialNode)
52:     PlaceObject coordinates = places.get(initialNode)
53:     PlaceObject placeObject = new PlaceObject(initialNode, new Point(coordi-
    nates.getPoint().getLat(), coordinates.getPoint().getLon()), placeObject.get-
    Name(), placeObject.getType(), nodeMap.get(initialNode), 0)
54:     objectSummary = placeObject
55: Else
56:     OtherObject otherObject = new OtherObject(initialNode, tempNeighbors,
    rootKeywords, nodeMap.get(initialNode), 0)
57:     objectSummary = otherObject
58: Return objectSummary
59: End Function
```

Algorithm 3.2.1 outlines the steps to create an Object Summary (OS) for a given keyword. The process begins by initializing necessary variables and retrieving the initial node corresponding to the keyword from the node and keyword maps. If the initial node is found, it is processed based on whether it is a keyword or a regular

node. The algorithm then traverses the dataset, processing neighbors through first, second, and third hops to construct the OS. Functions like ‘ProcessKeyword’, ‘ProcessSecondHopNeighbors’ and ‘ProcessThirdHopNeighbors’ handle the traversal and neighbor processing. Finally, the ‘createRoot’ function assembles the OS from the processed data and returns it. This structured approach ensures a comprehensive and efficient summary of the data related to the specified keyword.

3.3 Example Structure of a Generated Object Summary

Assume the following data:

- Node "Pericles" has neighbors "Athens", "Democracy", "Philosophy".
- "Athens" has neighbors "Greece", "Sparta".
- "Democracy" has neighbors "Government", "Elections".
- "Philosophy" has neighbors "Socrates", "Plato".

The resulting Object Summary (OS) would look like this:

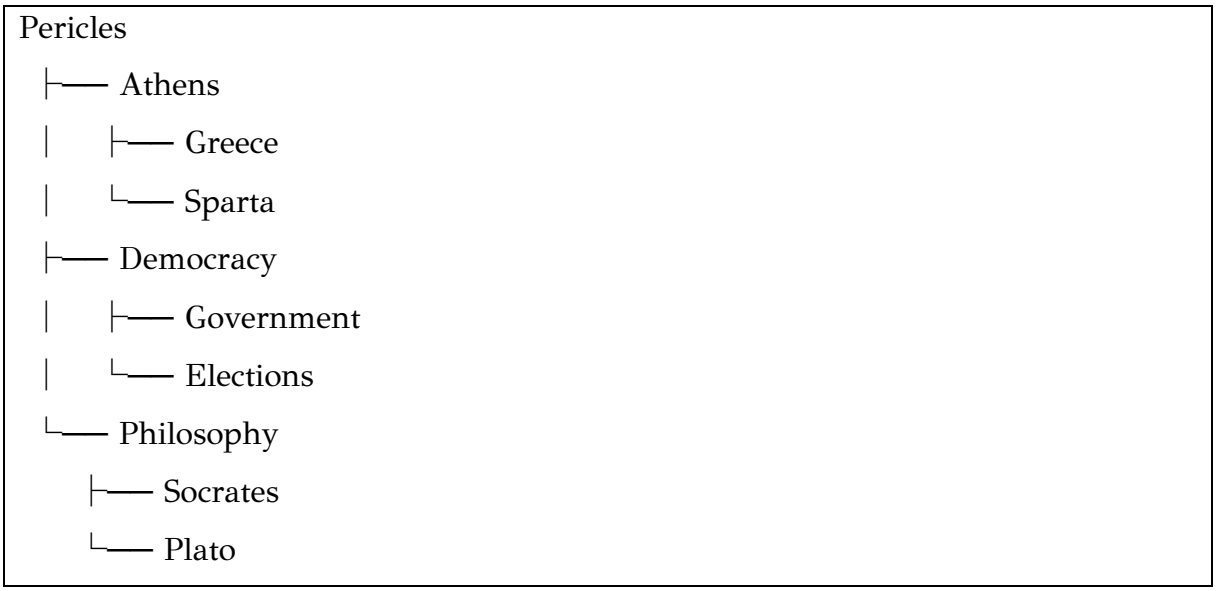


Figure 1: Object Summary for ‘Pericles’ demonstrating hierarchical relationships with key associated entities.

1. **Initialization**: initialNode is set to the ID corresponding to "Pericles".

2. **First Hop Neighbors**: Retrieve neighbors of "Pericles": "Athens", "Democracy", "Philosophy" and create PlaceObject or OtherObject instances for these neighbors.
3. **Second Hop Neighbors**:

For each first hop neighbor, retrieve their neighbors:

- "Athens" -> "Greece", "Sparta"
- "Democracy" -> "Government", "Elections"
- "Philosophy" -> "Socrates", "Plato"

Create PlaceObject or OtherObject instances for these second hop neighbors.

4. **Third Hop Neighbors**: Since there are no further neighbors listed, this step might be skipped for this example.
5. **Construct Root Node**: Create the root node "Pericles" with tempNeighbors as its children.
6. **Clean Up**: Clear temporary maps: edgesMap, keywordsListMap, places.

CHAPTER 4

SPATIAL PROPORTIONALITY

In this chapter, we delve into the realm of spatial proportionality and explore various algorithms designed to achieve this goal. We begin with an introduction to the fundamental concept of spatial proportionality, setting the stage for the algorithms that follow. Section 4.1 provides an overview of this concept, establishing a foundation for understanding the subsequent discussions. In Section 4.2, we present the Baseline Algorithm, detailing its implementation and functionality. This algorithm serves as a reference point for comparing more advanced methods. Next, in Section 4.3, we introduce the Grid Algorithm. We offer a comprehensive explanation of its workings, highlight its advantages, and describe its implementation in detail. This algorithm is examined for its efficacy and benefits over the Baseline Algorithm. Section 4.4 provides a comparative analysis of the Baseline and Grid

Algorithms, evaluating their respective strengths and weaknesses in achieving spatial proportionality. Finally, in Section 4.5, we explore the Random Sampling Algorithm for spatial proportionality. This section covers its implementation and discusses both the advantages and limitations of this approach, providing a balanced view of its practical applications.

4.1 Introduction to Spatial Proportionality

In the realm of spatial keyword search, where the goal is to retrieve and rank spatial objects based on their contextual and locational relevance, the concept of spatial proportionality emerges as a critical factor. Spatial proportionality aims to ensure that the retrieved subset of spatial objects represents the overall spatial distribution and contextual diversity of the original dataset. This concept not only enhances the quality of search results but also aids users in gaining a more comprehensive understanding of the spatial landscape. This report delves into the baseline and grid algorithms designed to achieve efficient and effective spatial proportionality.

4.1.1 The Concept of Spatial Proportionality

Spatial proportionality is a fundamental concept in spatial keyword search, aimed at ensuring that the subset of spatial objects retrieved is a representative sample of the entire dataset in terms of both spatial distribution and contextual relevance. This concept is crucial for providing users with search results that accurately reflect the geographic and thematic diversity of the dataset, thereby improving the usability and interpretability of the search outcomes. Spatial proportionality involves selecting a representative subset of spatial objects from a larger set, such that the chosen subset maintains the spatial and contextual distribution of the original dataset. This entails ensuring that the selected objects are proportionally spread out in the spatial domain and that they reflect the diversity of contexts present in the dataset. The challenge lies in balancing relevance, diversity, and proportional representation, which often requires sophisticated computational techniques. Proportionality with respect to context and location is essential to providing users with diverse and representative query results. Various proportionality scores are defined [9], such as spatial

proportionality $pS(p_i)$ and contextual proportionality $pC(p_i)$, which help measure how well the selected subset reflects the spatial and contextual distribution of the entire set. Spatial proportionality score $pS(p_i)$ of a place p_i is the following:

$$pS(p_i) = pSS(p_i) - pSR(p_i)$$

Where:

- $pSS(p_i)$ is the sum of spatial similarities between $sS(p_i, p_j)$ and all other places in the set S.
- $pSR(p_i)$ is the sum of spatial similarities between $S(p_i, p_j)$ and all other places in the subset R.

The goal is to select places that are not only relevant to the query but also ensure that the overall set is spatially diverse and contextually representative.

Note that here, $sS(p_i, p_j)$ stands for Ptolemy's spatial similarity, defined as $1 - dS(p_i, p_j)$, where $dS(p_i, p_j)$ computes the spatial diversity between p_i and p_j. The rationale behind $pSS(p_i)$ is to favor places surrounded by numerous neighbors within set S concerning the query point. Conversely, $pSR(p_i)$ favors places divergent from the remaining places in set R, thereby embracing spatial diversity. Importantly, both $pSS(p_i)$ and $pSR(p_i)$ pivot on the query location q. The score $pS(p_i)$ falls within the range $[0, K-k]$, akin to $pCS(p_i)$. Notably, computing $\mathrm{sS}(p_i, p_j)$for all pairs necessitates substantial computational effort.

In large spatial datasets, such as geographic information systems (GIS), social media geotags, and points of interest (POI) databases, the number of relevant objects returned by a query can be overwhelming. Without spatial proportionality, the top results may be clustered in a specific area or dominated by a particular context, which can lead to biased and less informative search results. Spatial proportionality addresses this issue by ensuring that the selected subset maintains Geographic Diversity. This ensures that the spatial objects are well-distributed across the geographic area of interest, preventing clustering in specific regions. Moreover, it reflects Contextual Diversity, ensuring that the objects represent a variety of contexts or themes, such as different types of POIs, activities, or events. Finally, it enhances User Comprehension, providing users with a holistic view of the spatial landscape, aiding in better decision-making and understanding of the area. Achieving spatial

proportionality involves balancing three key factors: relevance, spatial distribution, and contextual diversity. This requires sophisticated algorithms that can evaluate and integrate these factors effectively. Relevance refers to the closeness of the spatial objects to the query in terms of geographic proximity and thematic content, ensuring that the selected objects are pertinent to the user's query. Spatial Distribution ensures that the selected objects are spread out geographically, preventing the selection from being concentrated in a small area, which could skew the representation of the dataset. Finally, Contextual Diversity ensures that the objects represent different contexts or themes, avoiding redundancy and enhancing the richness of the information provided to the user. Relevance scores and importance scores are examples of how spatial proportionality can be quantified. Relevance scores are obtained by summing up the spatial similarity scores for each object, indicating its overall relevance in the context of the entire dataset. Importance scores, which are used to rank the objects, are computed by combining relevance scores with contextual information (such as hierarchical levels).

4.2 Baseline Algorithm

The baseline algorithm for spatial proportionality involves calculating pairwise spatial similarities between all objects, summing these similarities to get relevance scores, and then combining these scores with contextual information to get final importance scores. This approach, while accurate, can be computationally intensive, especially for large datasets. The Baseline Algorithm consists of these key steps: Initialization, Spatial Similarity Calculation, Proportionality Relevance Calculation, and Place Level Importance Calculation.

The algorithm begins by initializing essential parameters, including the collection of places within the target region and the maximum distance allowed for spatial calculations. The next step involves computing the spatial similarity scores between all pairs of spatial objects using the Euclidean distance. The similarity score between all pairs of places within the region is determined based on the Euclidean distance

between the geographical coordinates of each pair of places. This score is normalized by dividing it by the maximum distance using the formula:

$$sS(pi, pj) = \frac{\text{maxDist} - \text{Euclidean Distance}(pi, pj)}{\text{maxDist}}$$

Once the spatial similarities are computed, the algorithm calculates the proportionality relevance score for each object. The proportionality relevance score for a spatial object is the sum of its spatial similarity scores with all other objects in the dataset, indicating the overall relevance of the object in the spatial context using the formula:

$$pR(pi) = \sum_{\{j \neq i\}} sS(pi, pj)$$

The final step involves calculating the place level importance score for each object. This score is a combination of the proportionality relevance score and the hierarchical level of the object, using the formula:

$$\text{score}(pi) = \frac{\text{pr}[i]}{\text{totalPlaces} - 1} + \frac{1}{(\text{level}(pi) + a)}$$

where a is a smoothing factor to ensure proper weighting of objects at different levels. Finally, the algorithm executes the steps, that were previously described, in sequence and returns a list of places sorted by their calculated Place Level Importance scores.

The implementation details of the baseline algorithm for spatial proportionality include several key components. The algorithm utilizes a helper class to compute the Euclidean distance between pairs of geographical coordinates. It employs arrays and collections to manage and manipulate place objects and their associated scores efficiently. Additionally, places are sorted based on their Place Level Importance scores in descending order to prioritize the most relevant and important places within the region.

Concluding, the Baseline Algorithm provides a foundational approach for assessing the relevance and importance of places within a given geographical region. By

considering spatial similarity, pairwise relevance, and hierarchical importance, the algorithm offers valuable insights for various applications requiring spatial analysis and recommendation systems. The baseline algorithm, while effective in achieving spatial proportionality, can be computationally expensive due to the need for pair-wise comparisons and extensive similarity calculations.

4.2.1 Implementation of Baseline Algorithm

Algorithm 4.2.1, outlines a method to rank spatial objects by evaluating their contextual and locational relevance within a specified region. The algorithm processes a list of PlaceObjects and a maximum distance for normalization to produce a sorted list of these objects based on their computed scores. Below is the pseudocode for the algorithm.

Algorithm 4.2.1 Baseline

Input:

placesInRegion: List of PlaceObject

maxDist: Maximum distance for normalization

Output:

List of PlaceObject sorted by their scores

```
1:  Initialize:
2:      totalPlaces = size of placesInRegion
3:      ss = array of size (totalPlaces * totalPlaces)
4:      pr = array of size totalPlaces
5:      pli = array of PlaceObject of size totalPlaces
6:      a = 0.5
7:  // Step 1: Calculate Spatial Similarities
8:  Function calculateSS():
9:      for i from 0 to totalPlaces - 1:
10:         for j from 0 to totalPlaces - 1:
11:             distance = EuclideanDistance(placesInRegion[i].getPoint(),
    placesInRegion[j].getPoint())
```

```
            ss[i * totalPlaces + j] = (maxDist - distance) / maxDist
// Step 2: Calculate Proportionality Relevance
Function calculatePR():
    for i from 0 to totalPlaces - 1:
        sumOfSS = 0
        for j from 0 to totalPlaces - 1:
            if i != j:
                sumOfSS = sumOfSS + ss[i * totalPlaces + j]
        pr[i] = sumOfSS
// Step 3: Calculate Place Level Importance
Function calculatePLI():
    for i from 0 to totalPlaces - 1:
        node = placesInRegion[i]
        score = pr[i] / (totalPlaces - 1) + 1 / (node.getLevel() + a)
        node.setScore(score)
        pli[i] = node
// Main Execution
calculateSS()
calculatePR()
calculatePLI()
// Sort places by their scores in descending order
sortedPLI = sort pli in descending order based on scores
return sortedPLI
```

Here is the explanation of the pseudocode for Algorithm 4.2.1, Baseline. The process begins by initializing necessary data structures, including arrays for spatial similarities (ss), proportionality relevance (pr), and place-level importance (pli). The algorithm then proceeds through three main steps: First, the 'calculateSS' function computes the Euclidean distance between each pair of PlaceObjects. These distances are normalized using the maximum distance and stored in the spatial similarities array. Second, the 'calculatePR' function sums the spatial similarities for each PlaceObject, excluding itself, to determine its proportionality relevance. This sum represents how

each object is related to the others within the region. Third, the 'calculatePLI' function calculates a score for each PlaceObject by combining its proportionality relevance with a normalization factor based on its hierarchical level. This score is used to assess the overall importance of each PlaceObject. Finally, the PlaceObjects are sorted in descending order based on their scores, resulting in a ranked list that reflects both spatial distribution and contextual diversity. This method ensures that the most relevant spatial objects are highlighted, enhancing the effectiveness of spatial keyword searches.

4.3 Grid Algorithm

To address the computational challenges of the baseline algorithm, we used an innovative grid-based algorithm that offers an optimized approach by leveraging spatial partitioning techniques. Specifically, we delve into the optimization of Ptolemy's similarity computation by using an algorithm capable of accelerating the calculation of $s_S(p_i, p_j)$ for any given pair of places p_i and p_j. Ptolemy's similarity measure is a metric used to evaluate the similarity between pairs of spatial entities in the context of spatial keyword searches. This measure integrates both spatial and textual relevance to provide a comprehensive similarity assessment. It leverages Ptolemy's theorem, which involves the relationships between distances in a cyclic quadrilateral. Given two places p_i and p_j and a query point q, Ptolemy's similarity measure $s_S(p_i, p_j)$ is defined as:

$$s_S(p_i, p_j) = \frac{d(p_i, q) \cdot d(p_j, q) + d(p_i, p_j) \cdot d(q, q)}{d(p_i, q) + d(p_j, q)}$$

where:

- $d(p_i, q)$ is the spatial distance between place p_i and the query point q.
- $d(p_j, q)$ is the spatial distance between place p_j and the query point q.
- $d(p_i, p_j)$ is the spatial distance between place p_i and place p_j.

This measure balances the spatial proximity of the places to the query point with their direct spatial relationship. By considering these distances, the measure effectively captures both the geographical closeness and the contextual relevance of the places concerning the query.

The grid-based algorithm optimizes the process of achieving spatial proportionality by dividing the spatial domain into a grid. Each spatial object is assigned to a cell within this grid, and the algorithm approximates spatial similarities based on the cells rather than directly between all individual objects. This approach significantly reduces the number of pairwise comparisons needed, making it suitable for large-scale applications. This algorithm is designed to operate on two distinct grid structures: a squared grid and a radial grid structure but we selected to use the squared grid.

4.3.1 Detailed Explanation of Algorithm

Now let's see more details on how grid partitioning and proportionality techniques enhance the efficiency and scalability of spatial keyword searches. First, we have the Grid Partitioning (step1). The spatial domain is divided into a grid of cells. Each spatial object is assigned to a cell based on its coordinates. This partitioning helps in reducing the number of pairwise comparisons by considering only the objects within the same cell or neighboring cells. More precisely, the algorithm is initiated by generating a structured grid, denoted as G, consisting of square cells. This grid is centered around a specified query location q and effectively covers the spatial distribution of all places within the set S. The dimensions of the grid, including the length of its sides and the number of cells it encompasses, are strategically determined to optimize computational efficiency. The center of the grid G_c aligns with the query location q, while the length of each side (G_z) is set to twice the distance (f_p) between q and the farthest point in S. The grid size determines how finely the spatial domain is partitioned, directly impacting the number of comparisons and the level of detail captured.

Here, we have an example of grid:

Consider a grid where cells are denoted as $c_{x,y}$ with x and y being the coordinates relative to the query point q. For example, $c_{1,2}$ represents the cell at coordinates (1, 2) from the center. Below is a representation of such a grid, to help visualize how the grid partitions the spatial domain around the query point.

$c_{-3,3}$	$c_{-2,3}$	$c_{-1,3}$	$c_{1,3}$	$c_{2,3}$	$c_{3,3}$
$c_{-3,2}$	$c_{-2,2}$	$c_{-1,2}$	$c_{1,2}$	$c_{2,2}$	$c_{3,2}$
$c_{-3,1}$	$c_{-2,1}$	$c_{-1,1}$	$c_{1,1}$	$c_{2,1}$	$c_{3,1}$
$c_{-3,-1}$	$c_{-2,-1}$	$c_{-1,-1}$	$c_{1,-1}$	$c_{2,-1}$	$c_{3,-1}$
$c_{-3,-2}$	$c_{-2,-2}$	$c_{-1,-2}$	$c_{1,-2}$	$c_{2,-2}$	$c_{3,-2}$
$c_{-3,-3}$	$c_{-2,-3}$	$c_{-1,-3}$	$c_{1,-3}$	$c_{2,-3}$	$c_{3,-3}$

This approach simplifies and speeds up the process of spatial keyword search by reducing the computational load while maintaining accurate approximations of similarity scores. The pre-computed cell center scores can be reused for various queries, making this method both efficient and scalable.

The next step is the cell allocation (step 2), where we assign each place p from the set S to its corresponding grid cell. For each cell c_i, we maintain a count ($|c_i|$) representing the number of places it contains. Additionally, we approximate the location of each cell's center (c_{ci}), which serves as a proxy for the collective positions of all places within that cell. Next, we have the similarity score calculation (step 3). In this step, we calculate the Ptolemy's similarity score (p_s) for each cell c_i. Leveraging precomputed similarity scores $s_S(c_{ci}, c_{cj})$ between the centers of every pair of cells (c_i, c_j), stored in a matrix (s_{SM}), we employ a computation scheme that efficiently considers the cardinality of each cell ($|c_i|$) and the precomputed similarity scores.

This computation, adapted from Equation: $p_S(p_i) = \sum_{p_j \in S, p_i \neq p_j} s_S(p_i, p_j)$, involves summing the product of the cardinalities of c_i and c_j with their corresponding precomputed similarity scores, and then subtracting 1 to eliminate self-comparisons. So, instead of calculating the exact Euclidean distances between all pairs of objects, the grid-based algorithm approximates these distances by considering the distances between the centers of the grid cells.

Similar to the baseline algorithm, the grid-based algorithm calculates the proportionality relevance scores for each spatial object. The spatial similarity scores between objects within the same cell and neighboring cells are summed up to compute the proportionality relevance score for each object. The final place level importance score for each object is computed by combining its proportionality relevance score and its hierarchical level. The formula used is:

$$score(p_i) = \frac{pr[i]}{(totalPlaces - 1)} + \frac{1}{(level(p_i) + a)}$$

This step ensures that the scores reflect both the spatial and contextual importance of each object.

4.3.2 Advantages of Grid Algorithm

The grid-based algorithm offers several advantages. One key advantage is computational efficiency. By reducing the number of pairwise comparisons through grid partitioning, the grid-based algorithm significantly lowers the computational cost. This efficiency makes it feasible to apply the algorithm to large-scale datasets, where the baseline algorithm would be too slow. Another advantage is scalability. The grid-based algorithm scales well with the size of the dataset. As the dataset grows, the grid can be adjusted to maintain a balance between accuracy and computational efficiency. This allows the algorithm to handle a large number of spatial objects without a significant increase in computational complexity. Finally, the algorithm provides approximate similarity. While the grid-based algorithm uses approximations, it still maintains a reasonable level of accuracy in representing spatial proportionality. The use of grid cells allows for a balance between exact calculations and computational feasibility, providing a practical solution for large datasets.

4.3.3 Implementation of Grid Algorithm

The following pseudocode outlines Algorithm 4.3.2 Grid Algorithm, which is designed to rank Points of Interest (POIs) by leveraging grid partitioning and spatial similarity approximation. The algorithm operates through four main steps: grid partitioning, spatial similarity approximation, proportionality relevance calculation, and

place-level importance calculation. It follows with a detailed explanation of each step to provide a comprehensive understanding of the process.

Algorithm 4.3.2 Grid Algorithm

```
Input: POIs, maxDist, gridSize
Parameters: gridSize
// Step 1: Grid Partitioning
for each POI in POIs:
    cell_x = floor((POI.Latitude - minLatitude) / (maxLatitude - minLatitude) * gridSize)
    cell_y = floor((POI.Longitude - minLongitude) / (maxLongitude - minLongitude) * gridSize)
    grid[cell_x][cell_y].add(POI)
// Step 2: Spatial Similarity Approximation
for each cell_i in grid:
    for each cell_j in grid:
        distance = EuclideanDistance(center(cell_i), center(cell_j))
        similarity[cell_i][cell_j] = (maxDist - distance) / maxDist
for each cell in grid:
    for each POI in cell:
        for each neighbor_cell in get_neighboring_cells(cell):
            for each neighbor_POI in neighbor_cell:
                similarity_score = similarity[cell][neighbor_cell]
                ss[POI_i][POI_j] = similarity_score
// Step 3: Proportionality Relevance Calculation
for each POI in POIs:
    sum_of_similarities = 0
    for each neighbor_POI in get_neighboring_POIs(POI):
        POI_i = neighbor_POI;
        sum_of_similarities += ss[POI][neighbor_POI]
    pr[POI] = sum_of_similarities
// Step 4: Place Level Importance Calculation
```

```
25:  a = 0.5 // Smoothing factor
26:  for each POI in POIs:
27:      final_score = pr[POI] / (total_POIs - 1) + 1 / (POI.Level + a)
28:      POI.set_score(final_score)
29:  // Sorting and Output
30:  sorted_POIs = sort(POIs, by=final_score, order=descending)
31:  return sorted_POIs
```

The algorithm begins with grid partitioning, where the spatial domain is divided into a grid of cells. Each POI is assigned to a specific cell based on its latitude and longitude coordinates. This is done by calculating the cell's x-coordinate ('cell_x') by normalizing the POI's latitude within the grid size, and similarly, calculating the cell's y-coordinate ('cell_y') by normalizing the POI's longitude within the grid size. The POI is then added to the corresponding cell in the grid ('grid[cell_x][cell_y]'). The second step approximates the spatial similarity between POIs by calculating the distances between the centers of the grid cells they belong to. For each pair of cells ('cell_i', 'cell_j') in the grid, the Euclidean distance between their centers is computed, normalized, and stored in a similarity matrix ('similarity[cell_i][cell_j]'). For each POI in each cell, the algorithm retrieves the precomputed similarity score between the current cell and its neighboring cells, and stores the similarity score for the POI pair in the similarity score matrix ('ss[POI_i][POI_j]'). The third step involves calculating the proportionality relevance (PR) score for each POI by summing its spatial similarities with neighboring POIs. For each POI, the algorithm initializes 'sum_of_similarities' to 0, then iterates through its neighboring POIs, adding the similarity score between the POI and each neighboring POI to 'sum_of_similarities'. The PR score for the POI ('pr[POI]') is then set to 'sum_of_similarities'. In the fourth step, the final importance score for each POI is calculated by combining its PR score with a factor based on its hierarchical level. This is done using the formula: 'final_score = pr[POI] / (total_POIs - 1) + 1 / (POI.Level + a)', where 'a' is a smoothing factor set to 0.5. The computed final score is then assigned to the POI. The final step involves sorting the POIs based on their computed scores in descending order and returning the sorted list. By following

these steps, the Grid Algorithm efficiently partitions the spatial domain, approximates spatial similarities, calculates relevance scores, and ranks POIs, ensuring both computational efficiency and effective spatial keyword search results.

Concluding, the grid-based algorithm offers a practical and efficient approach to achieving spatial proportionality in large-scale spatial keyword search applications. By leveraging spatial partitioning techniques and approximate similarity calculations, it significantly reduces computational overhead while maintaining a reasonable level of accuracy. This makes the grid-based algorithm an essential tool for enhancing the relevance and usability of spatial search results in large datasets.

4.4 Comparison of Baseline with Grid Algorithm

The table below compares the Baseline Algorithm and the Grid-Based Algorithm across various aspects of their operation. It highlights differences in initialization, spatial similarity calculation, proportionality relevance calculation, place level importance calculation, computational complexity, accuracy, and scalability. The Baseline Algorithm uses direct pairwise comparisons, resulting in high accuracy but also high computational complexity. In contrast, the Grid-Based Algorithm partitions the spatial domain into a grid, approximating similarities using cell centers, which reduces computational complexity and enhances scalability, albeit with slightly lower accuracy.

Table 6: Comparison of Baseline and Grid-Based Algorithms for Spatial Keyword Search

Aspect	Baseline Algorithm	Grid Algorithm
Initialization	Directly uses all POIs	Partitions spatial domain into a grid
Spatial Similarity Calculation	Exact pairwise similarity using Euclidean distance	Approximate similarity using grid cell centers

Proportionality Relevance Calculation	Sums exact similarities with all other POIs	Sums approximate similarities within same/neighboring cells
Place Level Importance Calculation	Combines relevance score with hierarchical level	Combines relevance score with hierarchical level
Computational Complexity	High, due to pairwise comparisons	Lower, due to reduced number of comparisons
Accuracy	High, due to exact calculations	Slightly lower, due to approximations
Scalability	Limited by high computational cost	Scales well with large datasets

4.5 Random Sampling Algorithm for Spatial Proportionality

The Random Sampling Algorithm is a straightforward method used to select a representative subset of spatial objects from a larger dataset. This approach leverages randomness to ensure that the selected subset maintains the spatial and contextual diversity of the entire dataset. More precisely, Random Sampling Algorithm is designed to select a subset of places from a given region based on specific criteria. It initializes with parameters including the list of places in the region, maximum distance, desired number of results, and a distance percentage. The algorithm calculates a minimum distance threshold based on the provided parameters. It then randomly selects places from the region while ensuring they meet the distance criteria and have not been previously selected. The process continues until the desired number of results is obtained or there are no more places left to consider. Finally, it outputs the selected places sorted by score in descending order, along with their IDs and coordinates. This algorithm provides a systematic way to sample diverse locations from a region while maintaining spatial separation and potentially prioritizing places based on certain attributes.

4.5.1 Implementation of Random Sampling

The following pseudocode details a method for selecting a specified number of Points of Interest (POIs) from a given region, based on distance constraints and scoring.

Algorithm 4.5.2 Random sampling

```
Input: POIs, maxDist, numberOfRes, distPercentage
1:      Initialize results array with size numberOfRes
2:      Calculate minDist based on maxDist and distPercentage
3:      while counterInResults < numberOfRes:
4:          Randomly select a place objTemp from placesInRegion
5:          Add objTemp to results if it satisfies distance constraints and is not
        already included
6:          Increment counterInResults
7:          Remove objTemp from placesInRegion
8:      Sort results array in descending order by score
9:      return results
```

The algorithm begins by initializing an array named 'results' with a size equal to the desired number of results ('numberOfRes') to store the selected POIs. Next, the minimum distance ('minDist') is calculated based on the maximum distance ('maxDist') and a given distance percentage ('distPercentage'). This step likely determines the lower bound for distance constraints. The main part of the algorithm is a loop that continues until the 'results' array contains the specified number of POIs. Within this loop, a place ('objTemp') is randomly selected from the list of POIs ('placesInRegion'). The selected place is added to the 'results' array if it meets the distance constraints and is not already included. The counter tracking the number of results ('counterInResults') is incremented, and the selected place ('objTemp') is removed from 'placesInRegion' to avoid duplicate selections. Once the desired number of POIs has been selected, the 'results' array is sorted in descending order based on the score of each POI. Finally, the sorted 'results' array is returned as the output of the algorithm. This method ensures that a specified number of POIs are

selected randomly, subject to distance constraints, and then sorted by their scores for final output.

4.5.2 Advantages and Limitations of Random Sampling

The algorithm has several advantages and limitations. Among the advantages are its simplicity, as it is easy to implement and understand, and its efficiency, since it is computationally efficient due to requiring only random selection. However, the algorithm also has limitations. It lacks control, as it does not guarantee proportional representation in terms of spatial distribution and contextual diversity. Additionally, there is significant variance in results between different runs due to the random nature of the selection process. Lastly, it may not always provide a representative subset, especially for datasets with clustered distributions.

While the random sampling algorithm is useful for its simplicity and efficiency, it often serves as a starting point for more complex methods designed to ensure spatial proportionality. More sophisticated algorithms, such as the baseline and grid-based algorithms, provide better guarantees for maintaining spatial and contextual diversity in the selected subsets. The random sampling algorithm, however, remains a valuable tool for quick approximations and baseline comparisons in the context of spatial keyword search.

CHAPTER 5

SELECTION ALGORITHMS

In this chapter, we focus on the critical role of selection algorithms and their practical implementations. We start with Section 5.1, which discusses the importance of selection algorithms, highlighting their significance in various computational and real-world contexts. Section 5.2 delves into the Greedy Algorithm for selection, explaining its implementation in detail. This section serves as a foundation for understanding how simple, yet effective algorithms can solve selection problems efficiently. Following this, Section 5.3 introduces the Greedy-Disc Algorithm. We provide a detailed explanation of its implementation and demonstrate its application through an example use case. Specifically, Section 5.3.2 explores how the Greedy-Disc Algorithm can be used to diversify historical places in Athens related to Pericles, showcasing its practical utility.

5.1 Importance of Selection Algorithms

To enhance user experience and maintain clarity, we considered displaying a subset of the places of the object summary we got as result on the map. This strategic

selection aims to cover the map as evenly as possible, ensuring that the displayed points are not clustered too closely together. By implementing a greedy algorithm, we ensure that each chosen point maintains a minimum distance from the others. This approach reduces visual clutter and improves readability, allowing users to focus on key locations without being overwhelmed by too much information. Additionally, this method improves performance by reducing loading times, making the map more user-friendly and efficient. The result is a clean, aesthetically pleasing map that highlights significant points of interest, making it easier for users to interact with and analyse the displayed data.

5.2 Greedy Algorithm for Selection

Generally, a greedy algorithm is a problem-solving approach that makes the locally optimal choice at each stage with the hope of finding the global optimum. Greedy algorithms are typically used for optimization problems. The key characteristic of a greedy algorithm is that it builds up a solution piece by piece, always choosing the next piece that offers the most immediate benefit. We chose Greedy Algorithm to display K distinct objects from the list of all objects retrieved from the query after the algorithm's application (grid/baseline). The goal of this algorithm is to select a subset of spatial objects based on proportionality and relevance to a query context.
Greedy Algorithm is designed to select up to K places from a list such that each selected place is sufficiently far from the others, ensuring diversity and relevance. It is a greedy heuristic algorithm that iteratively selects the next best object based on its contribution to the proportionality of the current result set. The algorithm starts with an empty result set and computes initial scores or distances. Then, it uses distance metrics to ensure selected objects are diverse. This involves pairwise comparisons and optimization to reduce computational complexity. We use the Euclidean distance between places to ensure that each selected place is at least a minimum distance away from others. The algorithm selects objects that maximize the proportionality score, ensuring they contribute positively to the overall diversity and relevance, and iteratively adds them to the result set based on their contributions until

the desired number of objects (K) is reached. Finally, the algorithm returns a subset of objects that balance proportionality and relevance, ensuring diversity.

5.2.1 Implementation of Greedy Algorithm

Algorithm 5.2.1, the Greedy Algorithm, is designed to select a specified number of Points of Interest (POIs) based on distance constraints and scoring criteria. The algorithm ensures that the selected POIs are randomly chosen, meet the defined distance criteria, and are subsequently sorted by their scores. The following pseudocode illustrates the detailed steps of this algorithm:

Algorithm 5.2.1 Greedy Algorithm

Input: A list of PlaceObjectDistinct results, integer K, double maxDist, double distPercentage.
Output: A list of PlaceObjectDistinct resultsK that are selected.

```
1:  Initialize resultsK as an empty list.
2:  Calculate minDist as distPercentage * maxDist.
3:  Add the first place in results to resultsK and indexSet.
4:  for each place in results starting from the second place do
5:      if place is not in indexSet then
6:          if checkDistanceFromOtherPlacesInResults(place, counter) then
7:              Add place to resultsK.
8:              Add place to indexSet.
9:              Increment counter.
10:         end if
11:     end if
12:     if counter == K then
13:         break
14:     end if
15: end for
16: return resultsK
```

The algorithm starts by initializing the input parameters: the list of objects (results), the number of objects to select (K), the maximum distance (maxDist), and the distance percentage (distPercentage). It also initializes the result list (resultsK), which will store the selected objects, and an instance of the EuclideanDistance class for distance calculations. The method calculateMinDistBasedOnRegion computes the minimum allowable distance (minDist) between any two selected objects based on the given distPercentage of maxDist. The main selection happens in the execute method. It initializes a set to keep track of already selected object IDs to avoid duplicates. The first object from the list is always selected and added to resultsK. For each subsequent object, it checks if the object has not already been selected (using indexSet) and whether it maintains the minimum distance requirement from all previously selected objects (checkDistanceFromTheOtherPlacesInResults). If both conditions are satisfied, the object is added to resultsK and the ID is added to the index set. This process continues until K objects are selected or all objects are considered. The method checkDistanceFromTheOtherPlacesInResults iterates over the selected objects and calculates the distance between the current object and each of the already selected objects using the Euclidean distance formula. If any distance is found to be less than minDist, the object is rejected; otherwise, it is accepted.

5.3 Greedy-Disc Algorithm

The Greedy-DisC algorithm aims to construct a DisC diverse subset of a given set of query results, ensuring that this subset represents the entire set (coverage) while maintaining dissimilarity among the selected items. It is considered as a heuristic method designed to approximate a solution to an NP-hard problem. The goal of Greedy-DisC algorithm is to select a diverse subset of objects from a larger set such that the selected subset maximizes coverage and dissimilarity among its members. It does this by iteratively selecting the object with the largest "white neighborhood," which refers to the number of neighboring objects that have not yet been included in the diverse subset or marked as "covered."

We start with an empty subset (S) and we color all objects in the set (P) as white. Then there is the selection process. While there are still white objects we select the white object p_i that has the largest white neighborhood $N_r^W(p_i)$. This is the set of white neighbors within a radius r of p_i. Then we add p_i to the subset S, we color p_i black, indicating it has been added to S and we color all white neighbors of p_i (i.e., objects in $N_r^W(p_i)$) grey, indicating they are now covered by p_i. To efficiently implement the algorithm, we maintain a sorted list L' of all white objects based on the size of their white neighborhood. The object with the largest white neighborhood is always at the top of this list.

When initializing L', compute the size of the white neighborhoods for all objects. This is done by performing a range query $Q(p_i, r)$ for each object p_i and updating the neighborhood sizes accordingly. To reduce computational overhead, the algorithm uses a pruning rule: A leaf node (in the tree structure used for range queries) that contains no white objects is colored grey. When all children of an internal node are grey, the internal node is also colored grey. During range queries, subtrees rooted at grey nodes are not searched, thus reducing the number of node accesses, and speeding up the algorithm.

A key parameter in the Greedy-DisC algorithm is the radius r. The radius r significantly influences both the performance of the algorithm and the number of objects returned in the result subset. When r is small, each selected object covers a smaller area and fewer neighbors, resulting in the need for more objects to achieve full coverage. This increases the computational cost as the algorithm performs more iterations and range queries. Conversely, a larger radius allows each selected object to cover a larger area and more neighbors, reducing the number of objects needed and potentially lowering the overall computational effort. For tightly clustered datasets with a smaller maximum diagonal distance, a moderate increase in r can quickly reduce the number of objects required for coverage, enhancing performance and efficiency. In contrast, for widely spread datasets with a larger maximum diagonal distance, a significantly larger radius might be necessary to achieve similar reductions, though this could increase computational complexity due to larger range queries. To optimize results, it's advisable to start with a moderate r value and

incrementally adjust it, balancing the radius to effectively reduce the number of objects while managing computational costs. By fine-tuning r according to the dataset's distribution and the desired subset size, you can maximize both the efficiency of the algorithm and the quality of the results. To determine the optimal radius for specific datasets, we will perform a series of experiments, varying r and observing the resulting subset size and computational performance. These experiments will help identify the most effective radius values, ensuring that the algorithm performs optimally for different dataset characteristics.

5.3.1 Implementation of Greedy-Disc Algorithm

The following pseudo-code and its subsequent description detail the steps of the algorithm.

Algorithm 5.3.1 Greedy-DisC

Input: A set of objects P and a radius r.
Output: An r-DisC diverse subset S of P.

```
1:  S = ∅
2:  for all pi ∈ P do
3:      Color pi white
4:  end for
5:  while there exist white objects do
6:      Select the white object pi with the largest |NM_r(pi)|
7:      S = S ∪ {pi}
8:      Color pi black
9:      for all pj ∈ NM_r(pi) do
10:         Color pj grey
11:     end for
12: end while
13: return S
```

The Greedy-DisC algorithm begins by initializing an empty subset SS. All objects in the set PP are initially colored white, indicating that they have not been processed. The algorithm then enters a while loop that continues as long as there are white objects remaining. Within the loop, the algorithm selects the white object p_i that has the largest number of white neighbors within a given radius r (denoted as $|NrW(pi)|$). This object p_i is added to the subset S, and its color is changed to black to indicate that it has been included in the subset. Next, all white neighbors of p_i (objects within radius r) are colored grey, indicating that they are now covered by p_i and should not be selected again. This process repeats until no white objects remain. Finally, the algorithm returns the subset S, which represents the rr-DisC diverse subset of P.

5.3.2 Example Use Case: Diversifying Historical Places in Athens Related to Pericles

Consider a scenario where a user wants to find a diverse set of historical places in Athens that are related to Pericles, such as museums, archaeological sites, and galleries. The goal is to provide a subset of attractions that cover different types of locations and are spatially distributed across the city, ensuring that the selected attractions are both representative of Pericles' era and diverse in nature.

Scenario: A user queries, "Pericles" The objective is to present a diverse set of places that cover different aspects related to Pericles, including archaeological sites, museums, and galleries, ensuring they are spread out across the city.

Initial Setup: Historical places related to Pericles include the Acropolis, Parthenon, Ancient Agora, National Archaeological Museum, Acropolis Museum, Stoa of Attalos, Theatre of Dionysus, Odeon of Herodes Atticus, Kerameikos, Pnyx Hill, Museum of Cycladic Art, and Benaki Museum. The radius r is set to a value that ensures places within a 1 km radius are considered neighbors.

Iteration Details:

In the first selection, Acropolis is chosen. The neighborhood covered includes the Parthenon, Theatre of Dionysus, and Odeon of Herodes Atticus. The subset S includes {Acropolis}. The status is: Acropolis (black), Parthenon, Theatre of Dionysus, and Odeon of Herodes Atticus (grey), others (white). In the second selection, Ancient Agora is chosen, covering central Athens with multiple historical elements. The neighborhood covered includes the Stoa of Attalos. The subset S includes {Acropolis, Ancient Agora}. The status is: Acropolis, Ancient Agora (black), Parthenon, Theatre of Dionysus, Odeon of Herodes Atticus, Stoa of Attalos (grey), others (white). In the third selection, National Archaeological Museum is chosen for its comprehensive coverage of artifacts from Pericles' era. The neighborhood covered includes adjacent museums and galleries. The subset SS includes {Acropolis, Ancient Agora, National Archaeological Museum}. The status is: Acropolis, Ancient Agora, National Archaeological Museum (black), nearby locations grey, others white. In the fourth selection, Acropolis Museum is chosen for its focus specifically on artifacts from the Acropolis. The neighborhood covered includes nearby attractions within the Acropolis vicinity. The subset S includes {Acropolis, Ancient Agora, National Archaeological Museum, Acropolis Museum}. The status is: Acropolis, Ancient Agora, National Archaeological Museum, Acropolis Museum (black), covered places grey, others white. In the fifth selection, Kerameikos is chosen as an important archaeological site and ancient cemetery. The neighborhood covered includes nearby ancient ruins and sites. The subset S includes {Acropolis, Ancient Agora, National Archaeological Museum, Acropolis Museum, Kerameikos}. The status is: all selected places (black), rest (grey). In the sixth selection, Benaki Museum is chosen for its coverage of a range of historical periods, including that of Pericles. The neighborhood covered includes surrounding historical and cultural sites. The subset SS includes {Acropolis, Ancient Agora, National Archaeological Museum, Acropolis Museum, Kerameikos, Benaki Museum}. The status is: all selected places (black), rest (grey). Concluding, after running the Greedy DisC algorithm, the final subset SS includes a diverse range of historical places related to Pericles, spread across Athens and representing various types of attractions such as archaeological sites, museums, and galleries. This approach ensures that users are presented with a varied set of options,

each providing unique insights into the era of Pericles and reducing redundancy. By using the Greedy DisC algorithm, the selection of historical places is efficiently diversified, balancing both the spatial distribution and the diversity of historical contexts offered, thus enhancing the user's exploration of Pericles' legacy in Athens.

CHAPTER 6

EXPERIMENTS

In this chapter, we present a comprehensive analysis of various experiments conducted to evaluate the performance of different algorithms and parameters. We begin with an introduction to the experiments in Section 6.1, outlining the objectives and significance of the experimental evaluations. Section 6.2 provides a detailed description of the dataset used in the experiments, including a focus on popular subregions. Subsections 6.2.1 and 6.2.2 cover the identification of these subregions and the implementation of the algorithm for their creation. We then delve into specific experiments: Section 6.3 describes Experiment A, which focuses on the creation of object summaries. Section 6.4 details Experiment B, involving the tuning of

parameter dd in the Random Sampling Algorithm. Section 6.5 covers Experiment C, where the grid size parameter is tuned in the Grid Algorithm. Section 6.6 discusses Experiment D, which involves tuning the radius parameter rr in the Greedy-Disc Algorithm. Section 6.7 presents Experiment E, comparing the Grid Algorithm with the Baseline Algorithm, with a specific performance comparison on smaller regions with numerous nodes in Subsection 6.7.1. Section 6.8 explores Experiment F, which compares different selection algorithms. Finally, Section 6.9 concludes the chapter by summarizing the findings from the experiments and identifying the optimal model based on the results.

6.1 Experiments Introduction

In our research, we conducted six distinct experiments to comprehensively evaluate and optimize various algorithms related to spatial data analysis. In our research, we conducted a series of six distinct experiments to comprehensively evaluate and optimize various algorithms related to spatial data analysis. The primary objective of these experiments was to systematically investigate and improve the performance of algorithms used for data retrieval, quickly displaying objects on maps, and ensuring diversity in the presented data. By experimenting with different parameters and methods, we aimed to identify the most effective strategies and configurations, ensuring that our findings contribute to more efficient and accurate spatial data analysis. In the first experiment, we measured the time required to create object summaries by calculating the total time taken. The second experiment involved fine-tuning a sampling algorithm by testing different distance values to determine the optimal configuration. In the third experiment, we ran Grid algorithm to identify the most suitable grid size that yields better results. The fourth experiment aimed at tuning the Greedy-disc algorithm by adjusting the radius value to enhance performance. The fifth experiment compared the grid algorithm against a baseline algorithm to assess relative effectiveness. Finally, our sixth experiment involved a comparative analysis of selection algorithms, specifically evaluating the performance of random sampling, greedy, and greedy-disc algorithms. These experiments

collectively aimed to optimize algorithmic performance and provide insights into their practical applications in spatial data analysis.

6.2 Description of Dataset

In our experiments, we utilized a comprehensive dataset derived from DBpedia, which encompasses various types of data relevant to spatial analysis. The dataset is composed of several files, each serving a specific purpose. The 'node.txt' file contains all the nodes with their respective IDs and names, totaling 8,099,956 nodes. A subset of these nodes, specifically representing places, is detailed in the 'places.txt' file, which includes 883,664 place nodes. The 'pid.txt' file provides the latitude and longitude coordinates for each place node, facilitating spatial mapping. For each node, the 'keywordlist.txt' file lists the related keyword IDs, while the 'keyword.txt' file maps each keyword ID to its corresponding keyword name. Additionally, the 'edges.txt' file outlines the connections between nodes, with each line indicating a pair of connected nodes, encompassing a total of 6,799,279 connections. This rich dataset enabled us to perform detailed and diverse spatial data analyses, crucial for optimizing the algorithms under investigation.

For the experiments that test the algorithms, we created the object summaries of the nodes. In total, our dataset comprises 1,059,011 object summaries (OS). For our experiments, we focused on a subset of these object summaries, specifically selecting those with a large number of distinct places. This subset allowed us to evaluate the algorithms' performance in handling complex and diverse spatial data, ensuring that our findings are robust and applicable to scenarios involving high variability and density of spatial information. Here are some statistics about the object summaries used as our dataset for the experiments.

Table 7: Datset Statistics

Avg Total Places of Subset's Os	Avg Distinct Places of Subset's OS	Max value of Total Places of an OS (Overall)	Max value of Distinct Places of an OS (Overall)
10939	2672	50188	7449

6.2.1 Popular Subregions

Additionally, for each node within the small subset of summaries used in our experiments, we identified smaller areas that contain a high concentration of places. This allowed us to test the efficiency of the algorithms in these densely populated regions. By focusing on areas where the maximum distance between objects is smaller compared to the initial dataset, we aimed to assess the algorithms' performance in scenarios that simulate real-world conditions of high spatial density and local popularity. This approach ensured that our evaluation covered both broad and localized spatial contexts, providing a comprehensive analysis of the algorithms' effectiveness.

6.2.2 Implementation of Algorithm for Popular Subregions Creation

The following pseudocode outlines the steps taken to evaluate the algorithms in these concentrated regions. It encompasses the identification of densely populated nodes, the selection of smaller areas with maximum inter-object distances significantly smaller than those in the initial dataset, and the subsequent performance testing of the algorithms.

Algorithm 6.2.2 Popular Subregions Creation

```
    Function createSubregion(objectsSummaryNode, nodId, placesInRegion, totalPlaces)
1:  Declare variables:
2:      xmin, ymin, xmax, ymax as double, initialized to 0.0
3:      subregionSize as double, initialized to 0
4:      placesInSubregion as empty list
5:      sizeOfPlacesInRegion as double, set to totalPlaces * 0.2
6:  while subregionSize < estimatedPlacesInRegion
7:      If subregionSize == 0
8:          Set ymin to latitude of the first place in placesInRegion
```

```
9:          Set xmin to longitude of the first place in placesInRegion
10:         Increment subregionSize by 1
11:         Add the first place from placesInRegion to placesInSubregion
12:         Remove the first place from placesInRegion
13:     Else if subregionSize == 1
14:         Update ymin and xmin with latitude of the first place in placesInRegion
15:         Update ymin and xmin with longitude of the first place in placesInRe-
        gion
16:         Increment subregionSize by 1
17:         Add the first place from placesInRegion to placesInSubregion
18:         Remove the first place from placesInRegion
19:         Call insideBboxIntersectsCheck with current MBR and update
        placesInSubregion and placesInRegion
20:         Update subregionSize with the size of placesInSubregion
21:     Else if the first place in placesInRegion is within the current MBR
22:         If the place is not already in placesInSubregion
23:             Increment subregionSize by 1
24:             Add the place to placesInSubregion
25:             Remove the place from placesInRegion
26:     Else
27:         Update xmin, xmax, ymin, ymax with the coordinates of the first place
        in placesInRegion
28:         If the place is not already in placesInSubregion
29:             Increment subregionSize by 1
30:             Add the place to placesInSubregion
31:             Remove the place from placesInRegion
32:         Call insideBboxIntersectsCheck with updated MBR and update
        placesInSubregion and placesInRegion
33:         Update subregionSize with the size of placesInMBR
34: End while
35: End Function
```

```
36: Function insideBboxIntersectsCheck(xmin, xmax, ymin, ymax, placesInRegion, placesInSubregion)
37: Initialize placesInMBR as empty list
38: For each place in placesInRegion
39:     If place is not in placesInSubregion
40:         If latitude of place is between xmin and xmax
41:             If longitude of place is between ymin and ymax
42:                 Add place to placesInMBR
43: Return placesInMBR
44: End Function
```

The provided pseudocode describes a function named 'createSubRegion' which partitions a set of places into smaller subregions based on their geographic coordinates. The function initializes variables to track the minimum and maximum latitude and longitude (xmin, xmax, ymin, ymax), as well as lists to hold places within the current subregion. It iteratively processes places from 'placesInRegion', updating the bounding box coordinates (MBR) as it adds places to the current subregion. The function checks if the subregion size reaches an estimated size (20% of the total places) and calls another function, 'insideMBRObjectsCheck', to verify and update places within the current bounding box. The 'insideMBRObjectsCheck' function identifies and returns places within the specified bounding box, adding them to a list if their coordinates fall within the bounds. This process continues until the subregion size meets the estimated number of places, ensuring each subregion contains places geographically close to each other.

6.3 Experiment A – Object Summary Creation

In this experiment, we aim to measure the average time required to construct an object summary using data from DBpedia. The process involves reading input experiment parameters, creating necessary data structures, and generating object summaries for each experiment. The experiment is conducted by measuring the time taken for the OS creation. The steps involved in the experiment are as follows:

1. Setup,Initialization and Reading Experimental Data: The experiment starts by initializing variables to record the start and end times for each object summary creation. It also sets up paths to the required DBpedia data files and reads the input experimental configurations from a specified file. Then, using the 'ReadInputData' class, the experiment reads the input data which includes various experiment configurations such as object summary node, node ID, and query region.

2. Object Summary Creation: For each experiment configuration, the script checks if the source data is from DBpedia. It initializes a 'DBpediaSummary' object with the required data file paths. The process involves creating structures necessary for the object summary, generating the object summary for the specified node, and retrieving the summary.

3. Finding Relevant Places: After creating the object summary, the experiment uses the 'FindRelevantPlaces' class to find relevant places within the specified query region based on the generated object summary.

4. Time Measurement and Calculation: The time taken for each object summary creation is measured using 'System.nanoTime()'. This time is recorded and accumulated to calculate the total time taken for all experiments. The average time per experiment is computed and displayed after each iteration and at the end of the experiment.

5. Output: The experiment prints the time taken for each object summary creation and the current average time in seconds. Finally, it prints the total average time for all experiments.

Result:

Average Time for OS creation: 74.653683055 sec

6.4 Experiment B – Tuning parameter d in Random Sampling Algorithm

In this experiment, we aim to evaluate the performance of a sampling algorithm using different maximum distances between results on a map. To achieve this, we compare the performance of the sampling algorithm against a baseline algorithm by tuning the parameter 'd'. The parameter d represents a minimum distance threshold that dictates how close two selected objects can be to each other when displayed on the map. By adjusting d, we ensure that the objects shown in the results are sufficiently spaced apart, thereby enhancing the clarity and usefulness of the spatial representation on the map. This allows us to test and refine the algorithm's ability to maintain an optimal balance between object density and spatial distribution. After executing the experiments, we obtain the average execution time for both algorithms and the relative approximation error of the sampling algorithm compared to the baseline. The experiment involves the following steps:

Setup, Initialization and Reading Experimental Data: The experiment initializes variables to record execution times and scores for both the baseline and sampling algorithms. It also sets up paths to the required DBpedia data files and reads the input experimental configurations from a specified file. Then, using the 'ReadInputData' class, the experiment reads the input data which includes various experiment configurations such as object summary node, and query region.

1. Object Summary Creation: For each experiment configuration, the script checks if the source data is from DBpedia. It initializes a 'DBpediaSummary' object with the required data file paths and generates the object summary for the specified node.

2. Finding Relevant Places: After creating the object summary, the experiment uses the 'FindRelevantPlaces' class to find relevant places within the specified query region based on the generated object summary.

3. Maximum Distance Calculation: The experiment calculates the maximum distance within the query region using the 'CalculateMaxDistRegion' class.

4. Executing Baseline Algorithm: The experiment runs the 'Algorithm_sS_baseline_DBpedia' baseline algorithm and records its execution time.

5. Executing Sampling Algorithm: The experiment iterates over different values of the parameter 'd' (distance percentage options) and runs the 'Algorithm_sampling_DBpedia' sampling algorithm for each value, recording its execution time.

6. Calculating Approximation Error: The experiment calculates the relative approximation error of the sampling algorithm compared to the baseline algorithm using the 'FindRelativeApproximationError' class. This step involves comparing the scores of the top results from both algorithms.

7. Output: The experiment records the execution times and approximation errors for each value of 'd' and calculates the average times and errors.

Results:

Table 8: Performance Metrics for various DistPercentage Values

DistPercentage	Average Relative Approximate Error (%)	Average Time Sampling (sec)	Average Time Baseline (sec)
0,0005	17,60469078	0,006767489	11,20882603
0,001	18,40003295	0,016682745	11,20882603
0,002	19,02796759	0,020663199	11,20882603
0,005	21,35887775	0,027228646	11,20882603
0,01	23,30192525	0,02993709	11,20882603
0,02	27,50947446	0,034960268	11,20882603
0,05	35,47916029	0,040393139	11,20882603

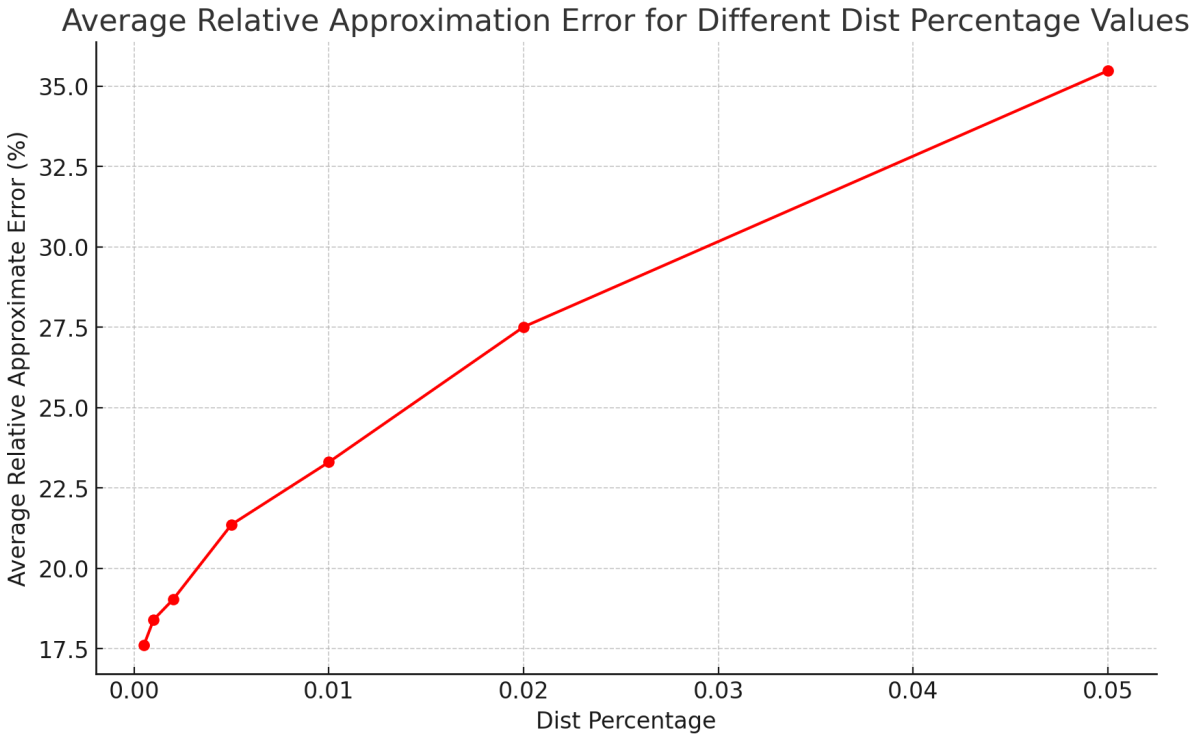


Figure 2: Average Relative Approximation Error for Different DistPercentage values

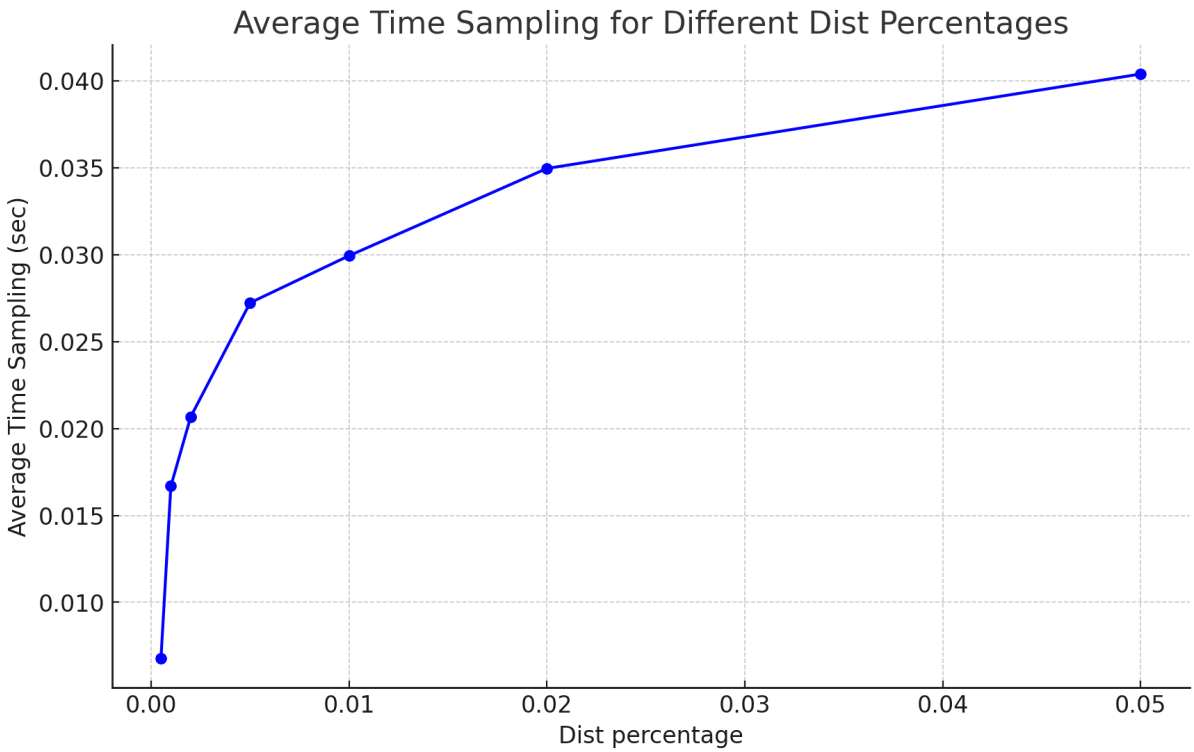


Figure 3:Average Time of Sampling for Different DistPercentage values

The performance evaluation of the sampling algorithm reveals that as the distance threshold parameter d increases, the average relative approximate error also

increases. Starting from a distance percentage of 0.0005 with an error of approximately 17.6%, the error gradually rises to about 35.5% at a distance percentage of 0.05. This trend indicates that as the distance between selected objects increases, the sampling algorithm's accuracy relative to the baseline decreases, suggesting that the objects become more spaced apart, leading to less accurate representations compared to the baseline. In terms of average execution time, the sampling algorithm consistently demonstrates significantly lower execution times compared to the baseline algorithm across all distance thresholds. While the baseline algorithm takes around 11.21 seconds consistently, the sampling algorithm's execution time starts at approximately 0.0068 seconds for the smallest distance percentage (0.0005) and increases slightly to around 0.0404 seconds for the largest distance percentage (0.05). Despite this increase, the sampling algorithm remains substantially faster than the baseline.

6.5 Experiment C – Tuning parameter grid size in Grid Algorithm

In this experiment, we aim to evaluate the performance of a grid-based algorithm by tuning the grid size parameter and comparing its performance against a baseline algorithm. In this experiment, we aim to evaluate the performance of a grid-based algorithm by tuning the grid size parameter and comparing its performance against a baseline algorithm. We calculate the average execution time and the relative approximation error of the grid algorithm compared to the baseline. By adjusting the grid size, we seek to identify the optimal configuration that balances computational efficiency with accuracy, ensuring that the algorithm performs well under various conditions. The experiment involves the following steps:

1. **Setup, Initialization and Reading Experimental Data**: The experiment initializes variables to record execution times and scores for both the baseline and grid algorithms. It also sets up paths to the required DBpedia data files and reads the input experimental configurations from a specified file. Then, using the ReadInputData class, the experiment reads the input data which

includes various experiment configurations such as diagonal distance flag, object summary node, and query region.

2. **Object Summary Creation, Finding Relevant Places and Maximum Distance Calculation**: For each experiment configuration, the script checks if the source data is from DBpedia. It initializes a DBpediaSummary object with the required data file paths and generates the object summary for the specified node. Then, the experiment uses the FindRelevantPlaces class to find relevant places within the specified query region based on the generated object summary. Afterwards, the experiment calculates the maximum distance within the query region using the CalculateMaxDistRegion class, which is necessary for both algorithms.
3. **Executing Baseline Algorithm**: The experiment runs the Algorithm_sS_baseline_DBpedia baseline algorithm and records its execution time.
4. **Executing Grid Algorithm**: The experiment iterates over different grid sizes (e.g., 6x6, 8x8, 10x10, etc.) and runs the Algorithm_grid_DBpedia grid algorithm for each grid size, recording its execution time.
5. **Calculating Approximation Error**: The experiment calculates the relative approximation error of the grid algorithm compared to the baseline algorithm using the FindRelativeApproximationError class. This step involves comparing the scores of the top results from both algorithms.
6. **Output**: The experiment records the execution times and approximation errors for each grid size and calculates the average times and errors. The results are printed for each grid size, showing the average time for the grid algorithm, the average time for the baseline algorithm, and the average relative approximation error.

Table 9: Performance Metrics for different grid size values

Grid Size	Average Relative Approximate Error (%)	Average Time Grid (sec)	Average Time Baseline (sec)
6x6	37,8912	0,0016	11,5633
8x8	30,939	0,0011	11,5633
10x10	30,7669	0,0006	11,5633
12x12	23,6697	0,0007	11,5633
14x14	22,0168	0,0007	11,5633
20x20	19,6488	0,0029	11,5633
30x30	12,0469	0,0036	11,5633
40x40	9,1839	0,009	11,5633
50x50	7,9788	0,0269	11,5633
60x60	6,5882	0,0569	11,5633
70x70	5,7652	0,0989	11,5633

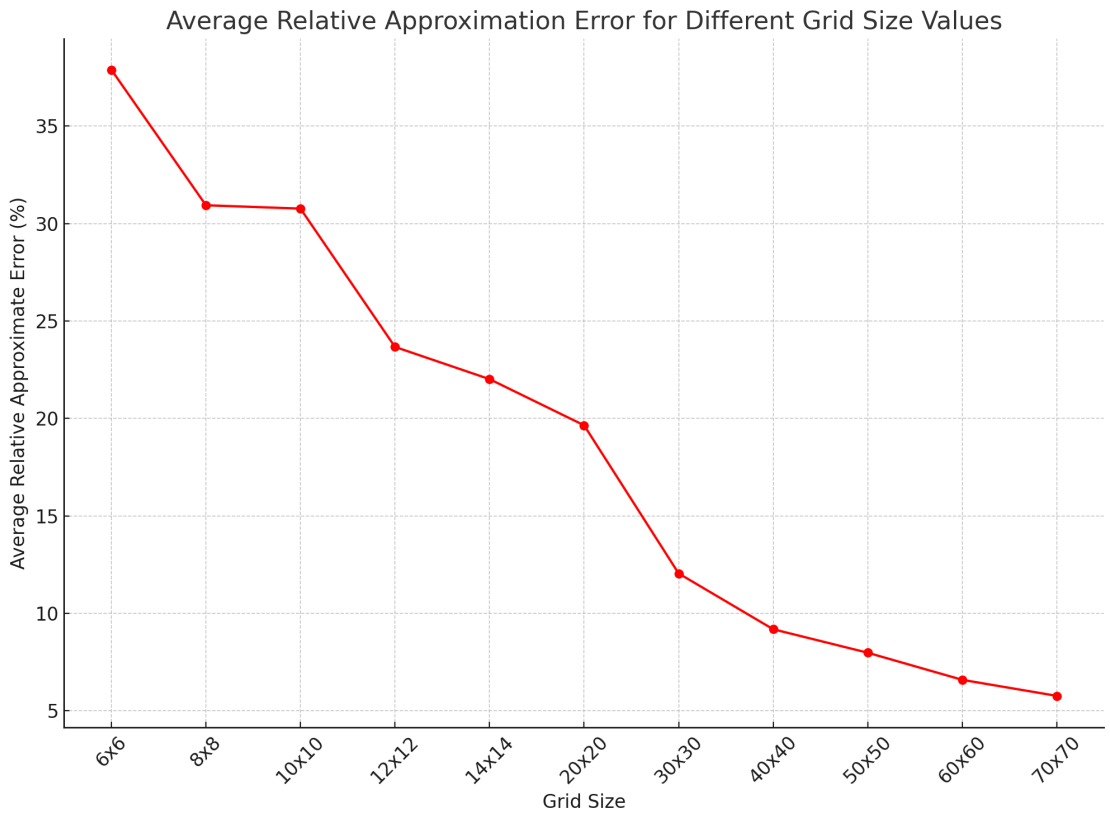


Figure 4: Average Relative Approximation Error for Different grid size values

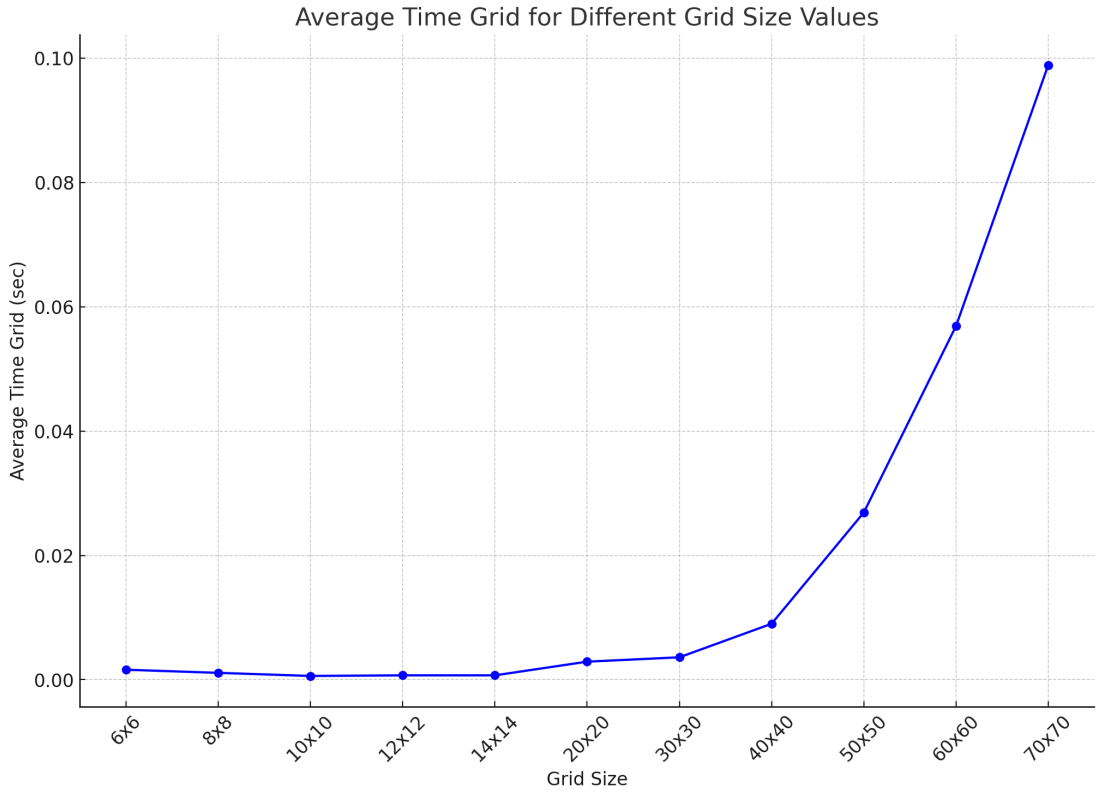


Figure 5: Average Time Grid for Different grid size values

Based on the results of the experiment, several conclusions can be drawn regarding the performance of the grid-based algorithm. The evaluation demonstrates that as the grid size increases, the average relative approximation error decreases significantly. Starting from a 6x6 grid with an error of approximately 37.9%, the error steadily decreases to about 8% with a 50x50 grid. This trend indicates that larger grid sizes enhance the accuracy of the grid-based algorithm compared to the baseline. In terms of computational efficiency, the grid-based algorithm consistently outperforms the baseline algorithm across all grid sizes. The average execution time for the grid algorithm ranges from 0.0006 seconds for a 10x10 grid to 0.0269 seconds for a 50x50 grid, which is significantly lower than the baseline algorithm's consistent execution time of around 11.56 seconds. These results highlight a clear trade-off between grid size and computational efficiency. Smaller grids are faster but less accurate, while larger grids provide higher accuracy at a marginally increased computational cost.

6.6 Experiment D – Tuning radius (r) in Greedy-Disc Algorithm

In this experiment, we aim to determine the optimal radius value for the greedy-disc algorithm by evaluating its performance across various radius settings. The greedy-disc algorithm is tested with the following radius values: 0.001, 0.0005, 0.0001, and 0.00005. These values are multiplied by the maximum distance of objects in each object summary to derive the specific radius used in the experiment. For each radius value, we measure the time taken to execute the algorithm, providing insights into the computational cost associated with different radius sizes. Additionally, we assess the approximate error by comparing the algorithm's outputs to those of a baseline algorithm known for its accuracy. By analyzing these two metrics—execution time and approximate error—we aim to identify the radius value that offers the best trade-off between speed and precision, ultimately enhancing the practical utility of the greedy-disc algorithm.

1. **Read Experiment Data**: Read experiment parameters from the input file and initialize necessary data structures and variables.

2. **Create Object Summaries and Calculate Maximum Distance**: Create object summaries, identify places in the query region and for the identified places, calculate the maximum distance (maxDist).
3. **Execute Baseline Algorithm**: Run the baseline algorithm to get baseline results. Measure the execution time and store the results.
4. **Execute Grid Algorithm**: Set up grid parameters, create a grid and execute the Grid algorithm to get grid-based results. Measure the execution time and store the results.
5. **Run Greedy-Disc Algorithm with Various Radius Values**: Iterate over the defined radius array: {0.001, 0.0005, 0.0001, 0.00005}. For each radius value: Calculate the specific radius as a fraction of maxDist. Execute the greedy-disc algorithm to get the results. Measure and store the execution time and calculate the relative approximation error compared to the baseline
6. **Output Results**: For each radius value, aggregate the execution times and relative approximation errors and then calculate the average execution time and average relative approximation error for each radius.

Results:

Table 10: Performance Metrics for different r percentage values

r_percentage	radius	greedyDisc +grid time	Relative approximation error
0.001	19,9657	209,3226	13,2511
0.0005	9,9829	345,3922	10,8128
0.0001	1,9966	994,0339	6,7853
0.00005	0,9983	1288,931	5,7729
0.00001	0.1996	1551.8521	4.93
0.000005	0.0998	1572.2599	4.8279

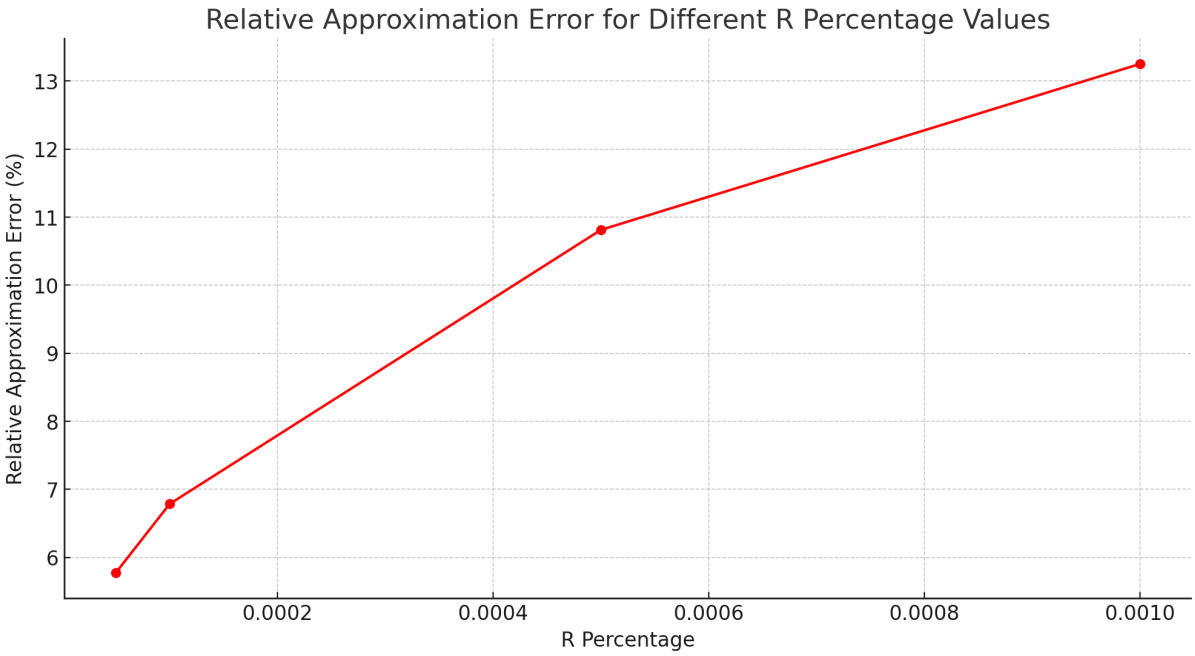


Figure 6:Average Relative approximation Error for Different r_percentage values

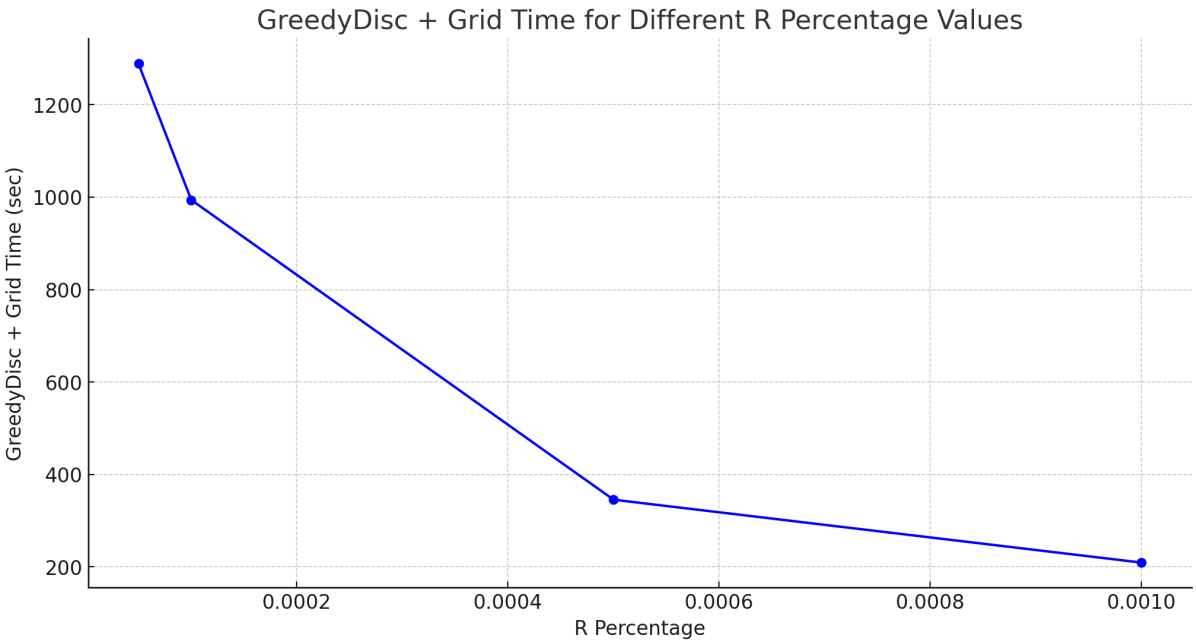


Figure 7: Average Greedy-Disc+Grid Time for Different r_percentage values

The experiment results reveal a clear trade-off between the execution time of the greedy-disc algorithm combined with the grid approach and the relative approximation error. As the radius percentage decreases, the execution time for the greedy-disc

+ grid algorithm increases significantly, from 209.32 seconds at a 0.001 radius percentage to 1288.93 seconds at a 0.00005 radius percentage, while the baseline algorithm's execution time remains constant at approximately 16.71 seconds. Concurrently, the relative approximation error decreases from 13.25% to 5.77% as the radius percentage decreases, indicating that smaller radius values yield more accurate results closer to the baseline algorithm. These findings highlight a trade-off between computational efficiency and accuracy: larger radius values (e.g., 0.001) offer faster results with higher approximation errors, whereas smaller radius values (e.g., 0.00005) provide more accurate results at the cost of higher execution times. A balanced approach suggests the 0.0005 radius value as a good compromise, offering a relative approximation error of 10.81% with a moderate execution time of 345.39 seconds. Ultimately, the choice of radius value should align with whether the priority is on faster computation or higher accuracy.

6.7 Experiment E – Comparison of Grid with Baseline Algorithm

In this experiment, we compare the performance of a grid-based algorithm with a baseline algorithm in terms of execution time and approximation error, using a grid size of 50x50 determined from previous experiments. The process involves reading input data that includes various experiment configurations, creating an object summary for each configuration, and finding relevant places within a specified query region. We then calculate the maximum distance within the query region, which is necessary for both algorithms. The baseline algorithm is executed first, and its execution time is recorded. Subsequently, the grid algorithm is executed with the specified grid size, and its execution time is also recorded. We calculate the relative approximation error of the grid algorithm compared to the baseline algorithm using the scores of the top results from both algorithms. Finally, we record and calculate the average execution times and approximation errors, presenting the results to evaluate the efficiency and accuracy of the grid-based approach. This comprehensive comparison enables us to evaluate the effectiveness of the grid algorithm in enhancing the efficiency and accuracy of spatial data analysis tasks.

Results:

Table 11: Grid-Baseline Performance Comparison

Avg nodes/OS=2672, grid size=50x50	Grid	Baseline	Sampling
Average Time (sec)	0,0269	12,4592	0,0236
Relative Approximate Error (%)	7,9788		15,7349

6.7.1 Performance Comparison of Grid and Baseline Algorithm on Smaller Regions with Numerous Nodes

Building on our previous experiment with a large subset of data, we have extended our investigation to a different dataset, focusing on popular subsets of each node where the region of each object summary is smaller. For this experiment, we utilized a smaller grid size of 20x20, reflecting the reduced complexity and size of the data regions. The objective remains the same: to measure and compare the execution time and approximation error of the grid-based algorithm against the baseline algorithm. By conducting this experiment with a different dataset and a smaller grid size, we aim to provide a more nuanced understanding of the grid-based algorithm's performance across varying data scales and complexities.

This study's outcomes will help determine the adaptability and efficiency of the grid-based algorithm in different spatial data scenarios, thereby offering insights into its practical applications for spatial data analysis tasks across diverse datasets.

All map

Avg nodes/OS=2672, grid size=20x20	Grid	Baseline
Average Time (sec)	0,0029	11,5633
Relative Approximate Error (%)	19,6488	

Subregion

Avg nodes/OS=1597, grid size=20x20	Grid	Baseline
Average Time (sec)	0,002	5,3338
Relative Approximate Error (%)	8,0141	

Figure 8: Grid-Baseline Performance Comparison in All map and Popular Subregion

The results indicate that for the entire map, the grid-based algorithm exhibited a significantly lower average execution time (0.0029 seconds) compared to the baseline algorithm (11.5633 seconds). Similarly, for subregions, the grid-based algorithm demonstrated a reduced execution time (0.002 seconds) compared to the baseline (5.3338 seconds). In terms of relative approximate error, the grid-based algorithm had a higher error (19.6488%) for the entire map compared to the baseline. However, this error reduced considerably when focusing on subregions (8.0141%), though no baseline error was provided for direct comparison. These findings suggest that the grid-based algorithm is significantly more efficient in terms of execution time across both the entire map and subregions. While it introduces a higher approximation error in larger, more complex data regions, its accuracy improves with smaller, less complex data regions. This efficiency and adaptability highlight the algorithm's potential for handling large datasets and complex spatial data scenarios quickly. Given its performance, the grid-based algorithm is particularly suited for

real-time spatial data analysis tasks where rapid processing is essential, and small approximation errors can be tolerated.

6.8 Experiment F – Selection Algorithms Comparison

In this experiment, we compare the performance of three selection algorithms: Greedy 1, Greedy Disc, and Random Sampling. We evaluate these algorithms by running them for the top 20, 50, 100, 200, and 300 results ('k'). We then compare their results with those obtained from the baseline algorithm. First, the baseline algorithm is executed, and its execution time is recorded. Next, the grid algorithm is executed with the specified grid size equal to 50x50, and its execution time is also recorded. We use the grid algorithm because it demonstrated better performance in previous experiments. The results from the grid algorithm are then used as input for the Greedy 1 and Greedy Disc algorithms. The Random Sampling algorithm uses the list of relevant places within the specified query region based on the generated object summary as input (the grid algorithm is not run in this case). We add the execution time of these selection algorithms to the time of grid execution. Therefore, we have the time for grid+Greedy 1, the time for grid+Greedy Disc, and the time for Random Sampling. Finally, we calculate the approximation error by comparing the combined results of the grid and each selection algorithm and the ones from random sampling algorithm with the results of the baseline algorithm. This comparison allows us to assess the efficiency and effectiveness of each algorithm in selecting the most relevant data points from the dataset, providing insights into their relative strengths and weaknesses across different result set sizes.

Results:

Topk 20	Sampling	Grid+Greedy	Grid+greedyDisc
Average Time	0,0075	0,0217	183,3711
Relative Approximate Error	18,6618	4,0038	12,7394

Topk 50	Sampling	Grid+Greedy	Grid+greedyDisc
Average Time	0,0174	0.0177	179,9156
Relative Approximate Error	17,8747	6,7433	14,2236

Topk 100	Sampling	Grid+Greedy	Grid+greedyDisc
Average Time	0,0236	0,0178	177,7771
Relative Approximate Error	15,7349	6,1199	12,8638

Topk 200	Sampling	Grid+Greedy	Grid+greedyDisc
Average Time	0,0341	0,0194	176,7889
Relative Approximate Error	14,5861	6,592	11,5350

Topk 300	Sampling	Grid+Greedy	Grid+greedyDisc
Average Time	0,0461	0,022	177,3524
Relative Approximate Error	14,001	6,972	10,7188

Figure 9: Selection Algorithms Comparison for different top k results

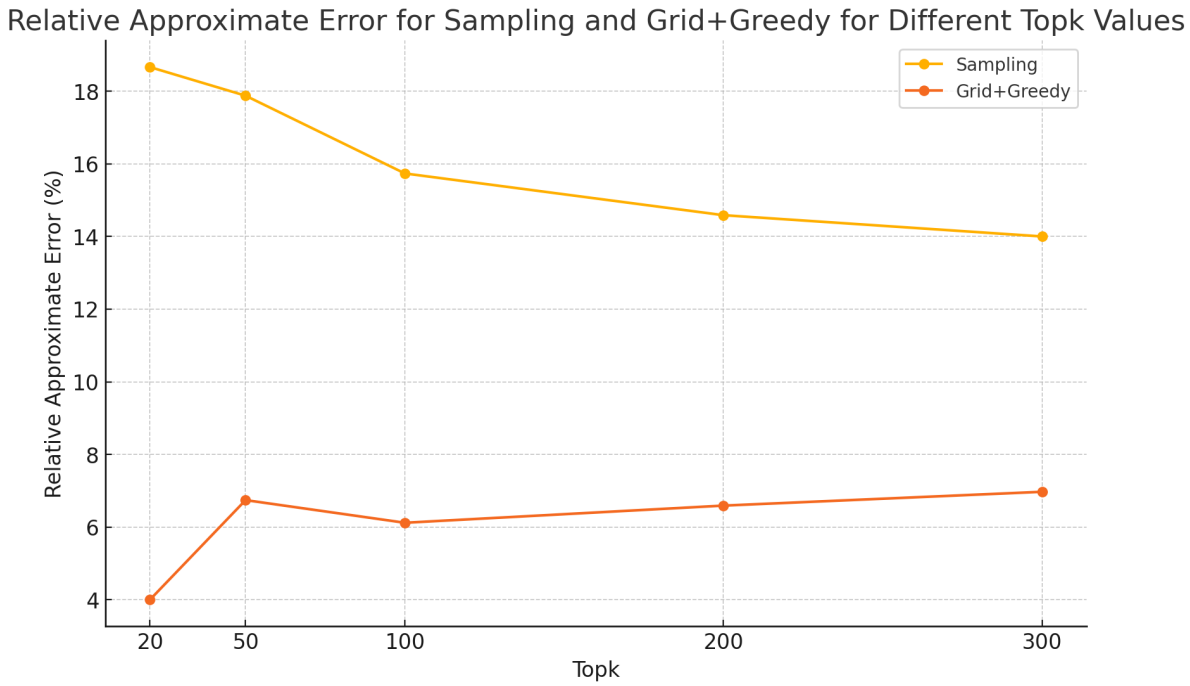


Figure 10:Average Approximate Error for Greedy+Grid and Random Sampling for different top k values

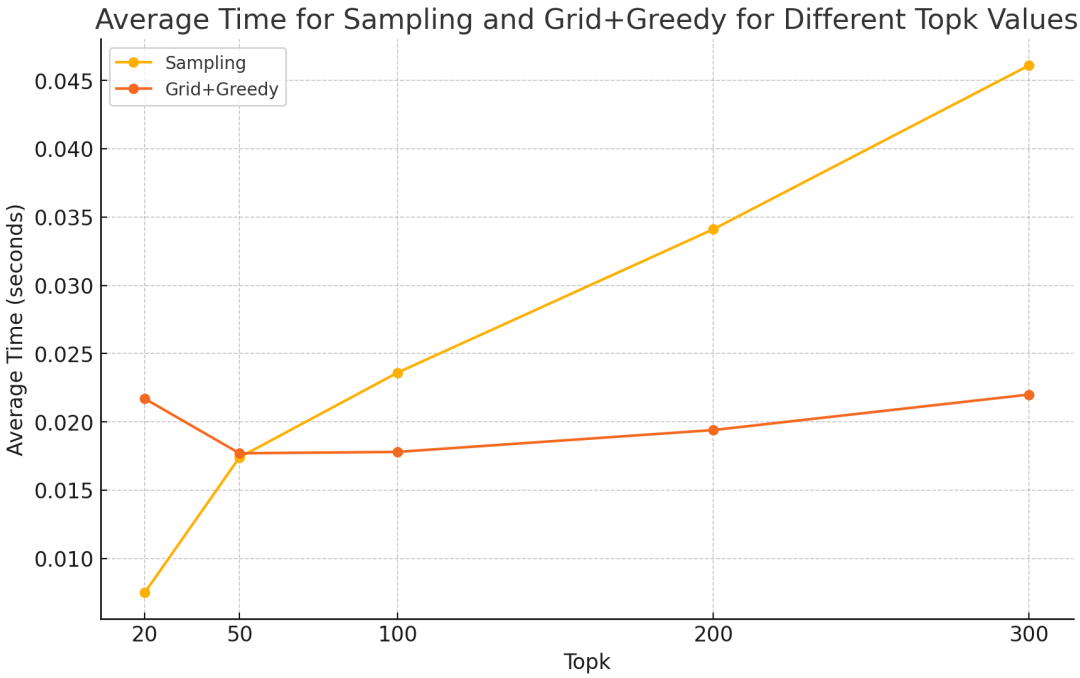


Figure 11: Average Time for Greedy+Grid and random Sampling for different top k values

The experiment comparing the Sampling, Grid+Greedy, and Grid+GreedyDisc algorithms across various ‘k’ values (20, 50, 100, 200, and 300) has provided valuable insights into their efficiency and accuracy. Each algorithm exhibits unique strengths and weaknesses, influencing their suitability for different applications. Following this, we see a detailed analysis of their efficiency, accuracy, and the overall implications of these results.

The efficiency of the three algorithms was evaluated based on their average execution times across different ‘k’ values (20, 50, 100, 200, and 300). The Random Sampling algorithm consistently demonstrated superior efficiency, with the lowest execution times ranging from 0.0075 seconds for top k 20 to 0.0461 seconds for top k 300. In comparison, Greedy had slightly higher, but still relatively low, execution times, ranging from 0.0217 seconds for top k 20 to 0.022 seconds for top k 300. On the other hand, Greedy-Disc showed significantly higher execution times, consistently around 177 to 183 seconds, indicating a substantial computational cost. These results highlight Sampling as the most time-efficient algorithm, followed closely by Greedy, with Greedy-Disc being the least efficient.

In terms of accuracy, measured by relative approximate error, the Greedy algorithm generally outperformed the other algorithms across all ‘k’ values. It achieved the lowest relative approximate errors, ranging from 4.0038 for top k 20 to 6.972 for top k 300, indicating high precision in selection. The Greedy-Disc algorithm also performed well, with errors decreasing as ‘k’ increased, from 12.7394 for top k 20 to 10.7188 for top k 300, demonstrating better accuracy than Sampling but not as high as Greedy. Conversely, the Random Sampling algorithm had the highest relative approximate errors across all ‘k’ values, ranging from 14.001 for top k 300 to 18.6618 for top k 20, reflecting its lower precision due to its random nature.

Concluding, the experiment reveals distinct trade-offs between the three algorithms. Random Sampling is the most efficient in terms of execution time but sacrifices accuracy. Greedy provides a balanced approach, offering good accuracy with moderate

efficiency, making it a strong candidate for tasks requiring both speed and precision. Greedy-Disc, while offering relatively good accuracy, particularly for higher 'k' values, is the least efficient due to its high computational cost. Therefore, the choice of algorithm should be guided by the specific requirements of the application: Sampling for speed, Greedy for a balance of speed and accuracy, and Greedy-Disc for accuracy when computational resources are not a constraint.

Due to the vastly different ranges of execution times, particularly for the Greedy Disc algorithm, plotting the results would not provide a visually effective comparison. The large discrepancy in values would distort the visual representation, making it challenging to interpret the results accurately. Therefore, we opted not to plot the results of the Greedy Disc algorithm.

6.8.1 Performance Evaluation of Greedy and Greedy Disc Algorithm on Smaller Regions with Numerous Nodes

In this experiment, we run selection algorithms: Greedy and Greedy Disc using a subset with smaller regions containing numerous nodes. We evaluate these algorithms by running them for the top 20, 50, 100, 200, and 300 results ('k'). We then compare their results with those obtained from the baseline algorithm. The experiment was conducted similarly to the one in the previous section. We have recorded the total average time taken for each algorithm and the approximation error by comparing the combined results of the grid and each selection algorithm with the results of the random sampling algorithm and the baseline algorithm.

Table 12: Greedy-Disc Results for subset with Smaller Regions

Top k Results	Average Time Greedy-Disc	Average Relative Approximate Error
20	61,2937	6,2907
50	59,5522	5,7779
100	60,8516	5,3338

200	61,332	4,472
300	62,1185	3,7846

Table 13: Greedy Results for subset with Smaller Regions

Top k Results	Average Time Greedy	Average Relative Approximate Error
20	0,0268	2,0626
50	0,0225	2,9924
100	0,0227	3,1142
200	0,0244	3,6563
300	0,0274	4,0104

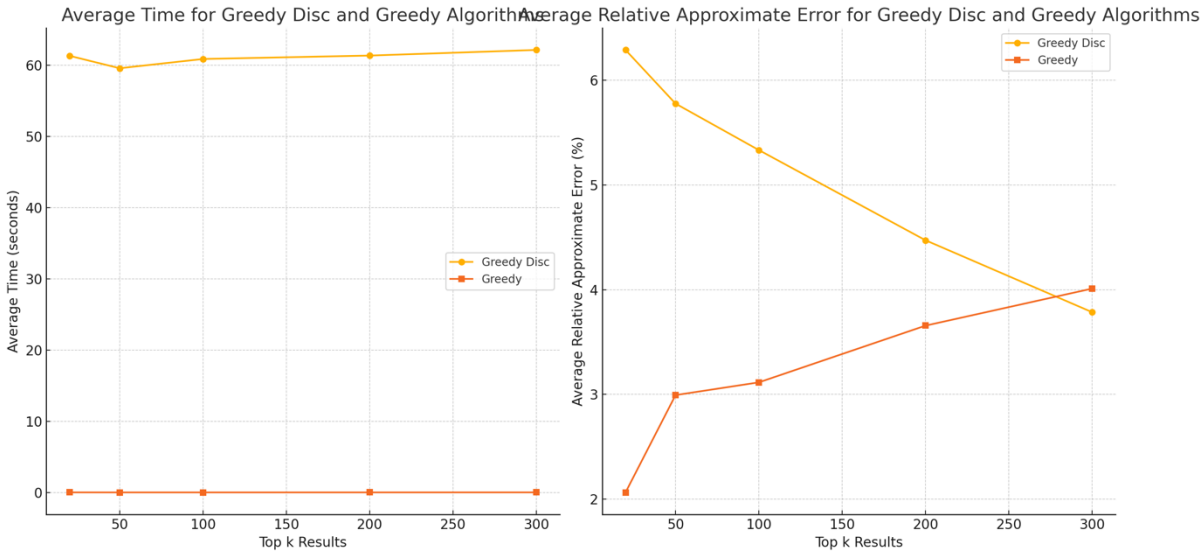


Figure 12: Comparison of Greedy and Greedy-Disc for subset with Smaller Regions

After executing this experiment, we evaluated the performance of the Greedy and Greedy Disc algorithms using subsets with smaller regions containing numerous nodes and compared them to results from subsets with larger regions. The Greedy algorithm demonstrated significantly lower execution times across all tested scenarios but exhibited a higher approximation error as the size of the result set increased. In contrast, the Greedy Disc algorithm excelled in maintaining more consistent

execution times and showed a decreasing approximation error with larger result sets, making it particularly suitable for tasks requiring higher accuracy. Although its execution times were considerably higher in larger regions, the Greedy Disc algorithm's accuracy improvement with increasing result set sizes highlights its advantage in scenarios where precision is paramount. Overall, the Greedy 1 algorithm is ideal for applications where speed is critical, while the Greedy Disc algorithm is better suited for tasks requiring higher accuracy, especially in smaller data regions. The selection of the appropriate algorithm should thus balance the trade-offs between execution time and accuracy based on the specific requirements of the application and the characteristics of the data regions.

6.9 Experiments Conclusion and Optimal model

After comparing the Grid, Baseline, and Random Sampling algorithms for spatial proportionality, and the Greedy and Greedy-Disc algorithms for selection, we have identified the optimal model for our purposes. Using the Grid algorithm for spatial proportionality is ideal because it is faster and maintains a relatively low approximation error. For displaying a subset of results on the map, the Greedy algorithm is the best choice due to its efficiency, lower approximation error, and acceptable execution time.

Although the Sampling algorithm performs very quickly, it has a higher approximation error, making it suitable for scenarios where speed is prioritized over accuracy. The Greedy-Disc algorithm, while ensuring diversity in the results displayed on the map, has a significantly higher execution time. However, its approximation error is not excessively high, suggesting it could be efficient in smaller regions, such as when zooming in on a specific area on the map. This balance between execution time and error makes the Grid and Greedy combination the most effective overall, with the option to use Sampling for speed and Greedy-Disc for diversity in smaller regions.

CHAPTER 7

DEVELOPMENT AND FUNCTIONALITY OF THE WEB APPLICATION

In this chapter, we provide a detailed overview of our web application, starting with a short description in Section 7.1. This section sets the context for understanding the application's design and capabilities. Section 7.2 delves into the functionality of the web application. Subsection 7.2.1 explores the use of @react-google-maps/api in our project, highlighting its integration and benefits. Subsection 7.2.2 discusses the implementation of interactive markers, customization options, and user interaction features, showcasing how users can engage with the map. In Subsection 7.2.3, we explain the process of dynamic marker fetching based on the map view and search keywords, ensuring relevant and updated data display. Subsection 7.2.4 outlines the MVC (Model-View-Controller) model employed in the application, providing a clear understanding of its architecture. Finally, Subsection 7.2.5 presents the overall workflow of the application, detailing the processes from user input to data rendering.

7.1 Web Application Description

This thesis involves creating a web application that allows users to enter a keyword in an input form, which then displays relevant nodes on a map. These nodes represent the object summary with the input keyword as the root. The nodes shown on the map dynamically update based on the current map bounds, ensuring that users always see the most pertinent information for their area of interest. This functionality combines user-friendly search capabilities with the powerful visualization features of Google Maps, providing an intuitive and interactive experience.

The main goal of this project is to visualize an object summary (OS) on a map, where the summary is generated dynamically based on a user-provided keyword. The application leverages advanced algorithms to determine the most relevant nodes to display, ensuring the results are both comprehensive and focused. The map interface supports zooming and panning, allowing users to explore different geographical areas and refine their search results in real-time. Additionally, the application includes interactive markers that users can click on to obtain more detailed information about each node. These markers are customizable, enabling the display of various types of data such as text and links. This enriches the user experience by providing multiple layers of information immediately. To enhance usability, the application features a responsive design that works seamlessly across different devices, including desktops, tablets, and smartphones. This ensures accessibility and convenience for users, regardless of their preferred platform. Overall, the web application integrates sophisticated data processing with a sleek and user-friendly interface, making it a valuable tool for visualizing and exploring object summaries based on user-defined keywords.

7.2 Web Application Functionality

The application consists of several key components and features that work together to deliver this functionality. To begin with, in the front page we have a search input form that allows users to input a keyword that serves as the root of the object summary. The SearchBar component captures the user's input and triggers the

creation of the object summary based on this keyword. On form submission, the application sends a request to the backend to fetch nodes related to the keyword, within the current map bounds. Next, we have the Map Component which displays the nodes of the object summary on a Google Map, providing a visual representation of the data. For the implementation we utilize @react-google-maps/api to render the map and manage map interactions. This app, also, provides dynamic Marker fetching. As the user zooms or pans the map, the application fetches and displays markers that fall within the new map bounds. Additionally, users can click on markers to view additional information in an InfoWindow. The purpose of Dynamic Marker Fetching is to ensures that only relevant nodes are displayed based on the current map view and search keyword. In this app Loading Indicators are also used to enhance user experience by indicating data loading processes. We use ClipLoader from react-spinners to show a spinner while fetching data. The spinner is displayed whenever a new fetch request is initiated and hides once the data is successfully loaded. Next, we have Marker Customization. The purpose is to differentiate nodes based on their hierarchical level within the object summary.Custom marker icons are used to represent different levels (yellow for level 0, red for level 1, blue for level 2, and green for level 3).The getMarkerIcon function assigns appropriate icons to markers based on their level. This comprehensive approach ensures that users can efficiently explore and interact with data related to their chosen keyword, making the application both powerful and user-friendly. In the following sections, we will describe the key features in more detail to provide a deeper understanding of the application's functionality and implementation.

7.2.1 Use of @react-google-maps/api in our Project

In this thesis, we integrated the @react-google-maps/api library to leverage the robust mapping capabilities of Google Maps within a React application. This library is a well-maintained wrapper around the Google Maps JavaScript API, designed to work seamlessly with React. Our goal was to provide users with an interactive and intuitive map interface, and @react-google-maps/api proved to be an excellent tool for this purpose. The decision to use @react-google-maps/api was driven by several key

factors. First of all, the library is specifically designed for React applications, providing a straightforward and familiar API for React developers. It also, offers efficient and performant rendering of Google Maps, with optimizations to handle large data sets and complex map features. Moreover, the library supports a wide range of customization options, allowing us to tailor the map's appearance and functionality to our specific needs. Finally, being widely used and actively maintained, @react-google-maps/api benefits from extensive documentation, community support, and regular updates. As far as the implementation is concerned, the initial setup of @react-google-maps/api was simple and well-documented. We installed the library via npm and configured it with our Google Maps API key. The library's modular structure allowed us to import only the components we needed, optimizing our application's performance.

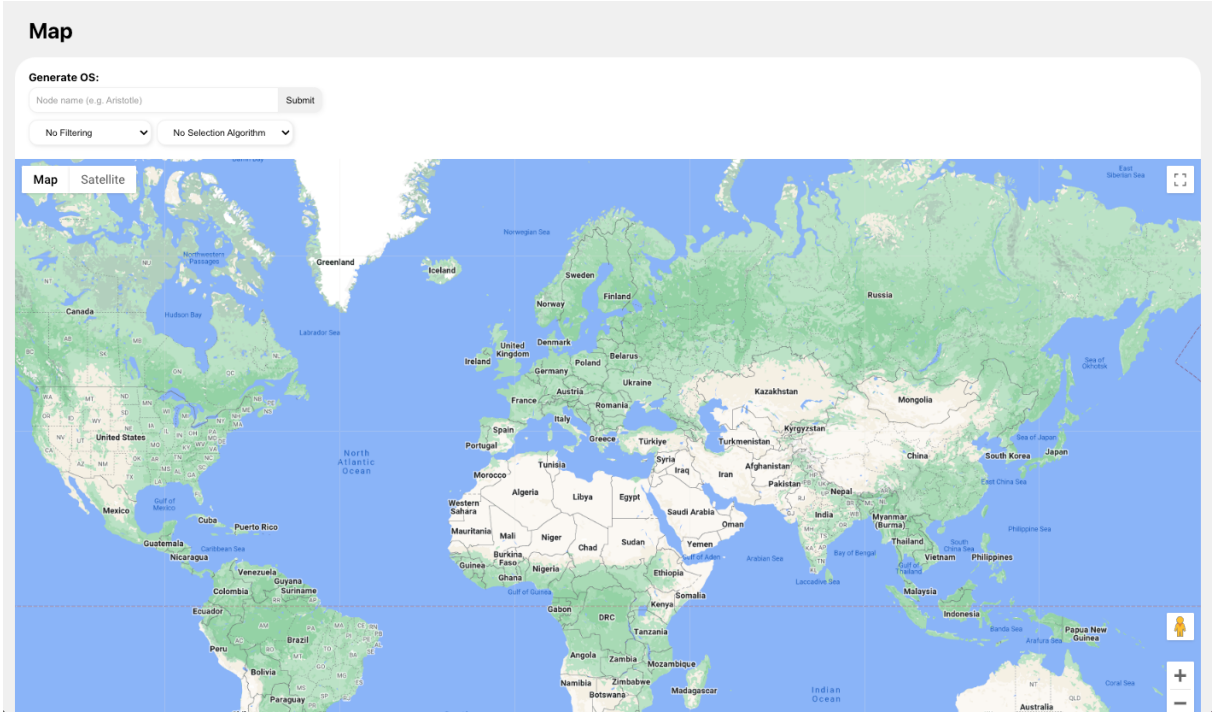


Figure 13: Web Application Home Page

7.2.2 Interactive Markers, Customization and User Interaction

Interactive markers were a crucial feature in our project, providing users with dynamic and engaging ways to interact with the map. These markers served as points of interest on the map, each capable of displaying additional information when

interacted with. We implemented interactive markers using the Marker and InfoWindow components from @react-google-maps/api. Each marker was placed at a specific geographic location, and when clicked, it triggered an infowindow that displayed relevant of the representing node. To enhance user interaction, we customized the markers and infowindows in several ways. To begin with, we used custom marker icons to make different types of locations easily distinguishable. More specifically we used different colors depending on the level of marker in the object summary. So, we have yellow for level 0(the root of os) , red for level 1, blue for level 2 and green for level 3. This involved specifying an icon property for the Marker component, allowing for a more intuitive and visually appealing map interface. Moreover, to enhance the dynamic content on the map, the content of the infowindows was generated dynamically based on the marker clicked. This allowed us to provide rich, location-specific information.

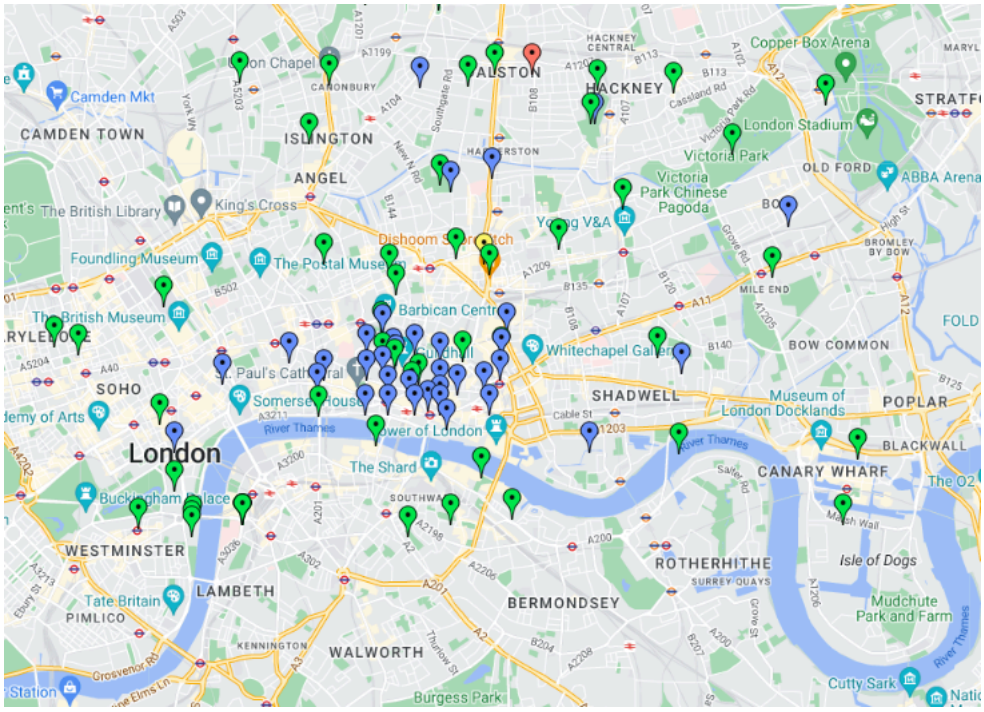


Figure 14: Different color of Markers on map

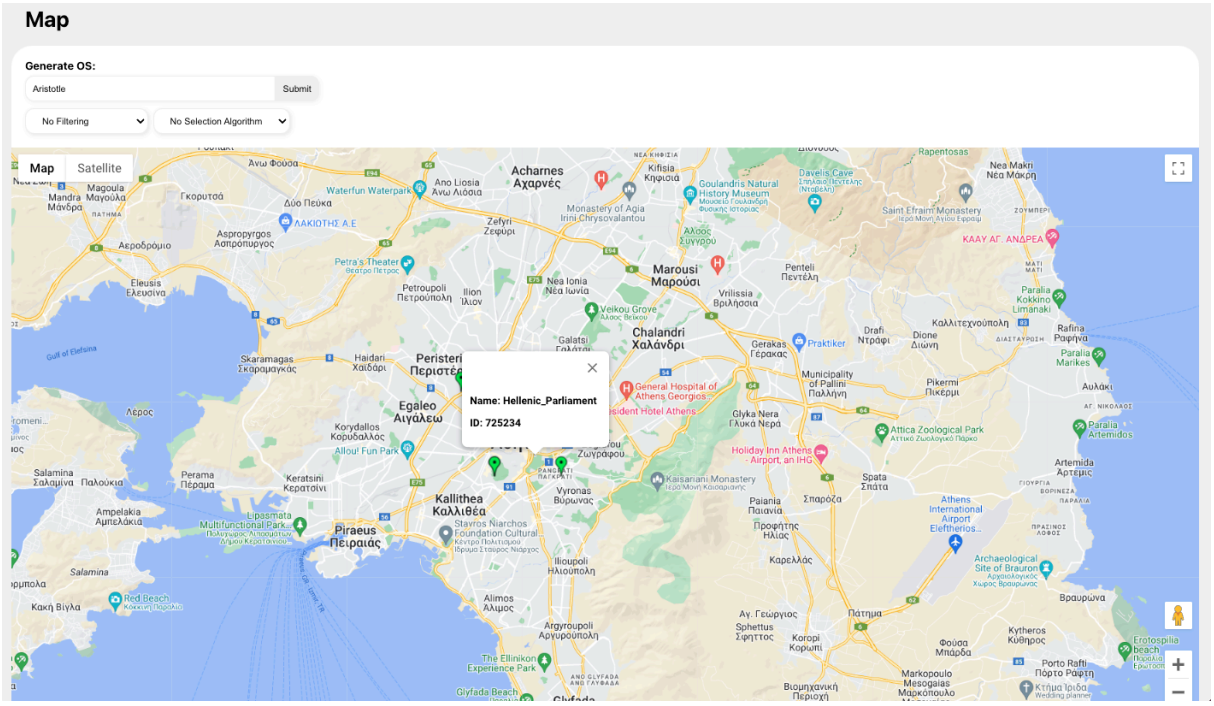
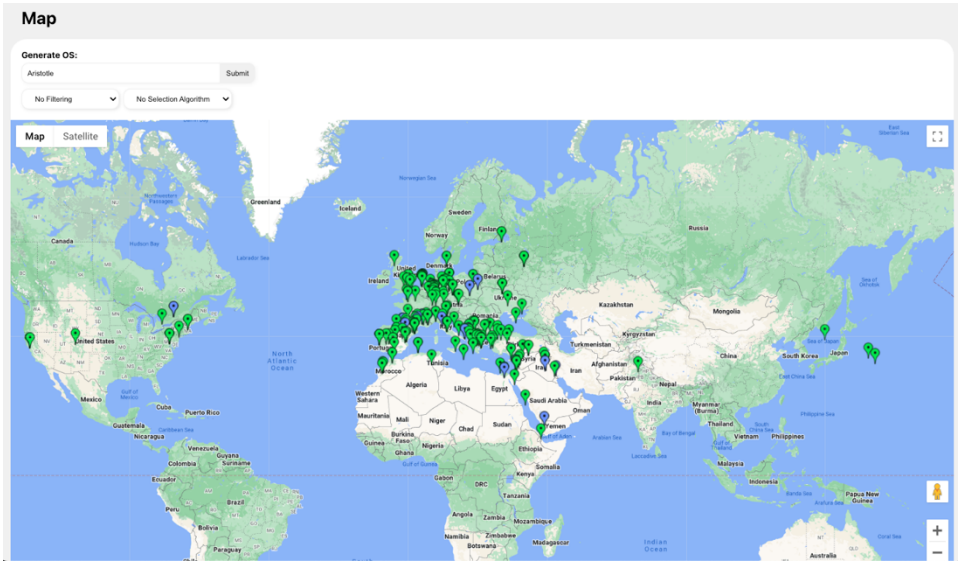


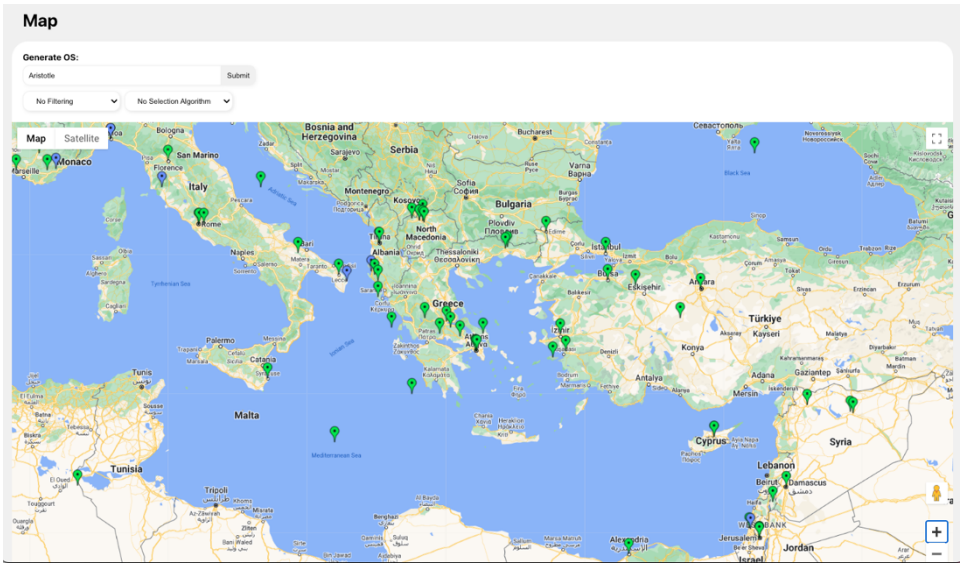
Figure 15: Example of InfoWindow

7.2.3 Dynamic Marker Fetching Based on Map View and Search Keywords

In our project, we implemented a dynamic marker fetching mechanism that updates the displayed markers based on the current viewport of the map and a keyword entered in the search bar. This ensures that users are always presented with the most relevant markers corresponding to their current area of interest and search criteria, enhancing both performance and user experience. The core idea is to fetch and display markers that fall within the bounds of the current map window and match the search keyword. As the user zooms or pans the map, or changes the search keyword, the latitude and longitude of the map's bounds change, triggering a fetch request to update the markers accordingly. Event handlers for onBoundsChanged, onZoomChanged, and onDragEnd capture map movements and update the markers accordingly. This dynamic fetching approach ensures that only the markers relevant to the current view and search criteria are loaded, significantly improving performance. It avoids the need to load all markers at once, which can be resource-intensive and slow down the application. Additionally, by updating

markers in real-time as the user navigates the map and enters search keywords, we provide a more responsive and interactive user experience. From a user perspective, this method ensures that they always see the most pertinent information based on their current map view and search criteria. As users zoom in to focus on a smaller area, more detailed markers become visible. Conversely, zooming out provides a broader overview with fewer markers, preventing the map from becoming cluttered. The ability to filter markers by search keywords further refines the displayed results to match user interests.





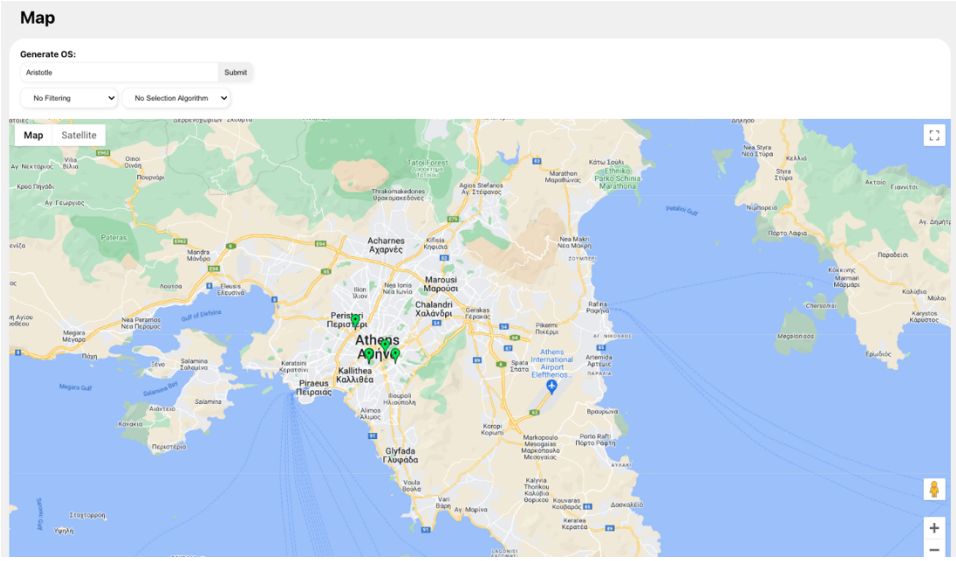


Figure 16 : Example of Zooming-In result

7.2.4 MVC Model

The Model-View-Controller (MVC) architecture is a widely-used software design pattern that separates an application into three interconnected components: the Model, the View, and the Controller. This separation helps manage complex applications by isolating the internal representations of information from the ways that information is presented to and accepted from the user. Here's how we implemented the MVC model in our project. The Model component represents the data and the business logic of the application. It is responsible for retrieving data, processing it, and storing it. In our project, the Model was responsible for retrieving marker data from the backend API based on the current map bounds and search keywords and storing and updating marker data in response to changes in the map view or search input. Also, the Model is responsible for the business Logic that means, filtering the markers based on the current search keyword and managing state related to the markers, such as which markers are selected or visible. The View component represents the UI of the application. It is responsible for rendering the data provided by the Model to the user and capturing user input. In our project, the View was responsible for rendering the Google Map component, displaying markers on the map based on data from the Model, capturing user interactions such as panning, zooming,

and clicking on markers and finally, displaying infowindows with additional marker information when a marker is clicked. The Controller acts as an intermediary between the Model and the View. It processes user inputs from the View, updates the Model, and then updates the View based on the new state of the Model. In our project, the Controller was responsible for handling User Input, for example, managing state changes triggered by user interactions with the map and search bar, updating the Model with new map bounds or search keywords and updating the View. That means, the controller ensures that the View reflects the current state of the Model and coordinates the fetching and rendering of markers based on the current map bounds and search keyword.

7.2.5 Workflow

1. **User Input:**

The user enters a keyword in the search input form and submits it.
The handleForm2Submit function captures the keyword and initiates a fetch request to retrieve the object summary nodes related to the keyword.

2. **Data Fetching:**

The application constructs an API request with parameters including the keyword, current map bounds (northEast and southWest coordinates), and selected algorithms for filtering and selection. The backend processes this request, generates the object summary, and returns the relevant nodes.

3. **Map Rendering:**

The fetched nodes are stored in the application's state (markers).
The map component renders these nodes as markers, updating them dynamically based on user interactions with the map.

4. **User Interaction:**

Users can zoom and pan the map, which triggers updates to the map bounds.
The application fetches new nodes that fall within the updated bounds, ensuring that the map always displays relevant information.

Users can click on markers to open InfoWindows that provide additional information about each node. The application handles marker clicks by updating the selected-Marker state and rendering the corresponding InfoWindow.

CHAPTER 8

CONCLUSION

This thesis set out to enhance the retrieval and visualization of spatial and contextual data, with a particular focus on achieving spatial proportionality and diversity in the displayed results. Through the development and testing of various algorithms, we aimed to identify the most efficient and effective methods for these tasks. After conducting comprehensive experiments comparing the Grid, Baseline, and Random Sampling algorithms for spatial proportionality, as well as the Greedy and Greedy-Disc algorithms for result selection, we have identified the optimal model for our purposes.

- **Grid Algorithm for Spatial Proportionality**: The Grid algorithm emerged as the ideal choice for achieving spatial proportionality. It provides a faster execution time compared to the Baseline algorithm and maintains a relatively low approximation error. This balance of speed and accuracy makes it highly suitable for large-scale applications where efficiency is crucial.
- **Greedy Algorithm for Result Selection**: For displaying a subset of results on the map, the Greedy algorithm is the best choice. It is efficient, has a lower approximation error, and offers an acceptable execution time. This makes it ideal for providing users with a clear and concise set of results that are both relevant and proportionally representative of the overall data set.

While the optimal model combines the Grid algorithm for spatial proportionality and the Greedy algorithm for result selection, alternative algorithms may be preferable in specific scenarios:

- **Random Sampling Algorithm**: Although the Random Sampling algorithm operates very quickly, it has a higher approximation error. This makes it suitable for scenarios where speed is prioritized over accuracy, such as initial exploratory searches or real-time applications where rapid response is critical.

- **Greedy-Disc Algorithm**: The Greedy-Disc algorithm ensures diversity in the results displayed on the map but comes with a significantly higher execution time. Despite this, its approximation error is not excessively high, suggesting that it could be effective in smaller regions, such as when users zoom in on a specific area of the map. This makes it a valuable tool for applications requiring high diversity and detailed local analysis.

The practical implementation of these findings is demonstrated through the development of a web application. This application integrates the Grid and Greedy algorithms to dynamically generate and display object summaries based on user queries. The application supports interactive map features, providing an intuitive and engaging user experience.

Future work could explore further optimization of the algorithms to reduce execution time and approximation error. Additionally, expanding the application to support more complex queries and integrating additional data sources could enhance its functionality.

Overall, this thesis has demonstrated the effectiveness of combining the Grid and Greedy algorithms for spatial proportionality and result selection, respectively. By addressing the challenges of efficiently retrieving and visualizing spatial data, this work contributes to the field of data management and provides a robust foundation for future advancements. The development of the web application showcases the practical applicability of these findings, offering users a powerful tool for exploring and interacting with spatial data.

REFERENCES

[1] Gong Cheng, Danyun Xu, and Yuzhong Qu.: *Summarizing Entity Descriptions for Effective and Efficient Human-centered Entity Linking* - the State Key Laboratory for Novel Software Technology, Nanjing University, China

[2] G. Cheng, D. Xu, Y. Qu, *C3d+ p: A summarization method for interactive entity resolution*, Journal of Web Semantics, pages 203-213, 2015.

[3] G. J. Fakas, *A Novel Keyword Search Paradigm in Relational Databases: Object Summaries,* Data and Knowledge Engineering (DKE) Journal, pages 208–229, 2011.

.

[4] G. Fakas, Y. Cai, Z. Cai, N. Mamoulis, *Thematic ranking of object summaries for keyword search*, Data and Knowledge Engineering Journal (DKE), pages 1–17, 2018.

[5] A. Balmin, V. Hristidis, Y. Papakonstantinou, *Objectrank: Authority-Based Keyword Search in Databases,* VLDB, pages 564-575, 2004

[6] G. J. Fakas, G. Kalamatianos, *Proportionality on Spatial Data with Context,* ACM Transactions on Database Systems (TODS), 2023

[7] S. Shaham, G. Ghinita, C. Shahabi, *Models and mechanisms for spatial data fairness*, Proceedings of the VLDB Endowment, 16(2), pp.167-179, 2022.

[8] Marina Drosou, E. Pitoura, *DisC diversity: result diversification based on dissimilarity and coverage.* – Proceedings of the Endowment, 2012

[9] G. Kalamatianos, G. J. Fakas, N. Mamoulis, *Proportionality in Spatial Keyword Search,* Proceedings of the *ACM Conference on the Management of Data (SIGMOD)*, 2021.

Short Biography

Kalliopi Basiakou is an M.Sc. graduate student with specialization in Data Science and Engineering, at the Department of Computer Science and Engineering (CSE) of the University of Ioannina, Greece. She received her M.Eng. degree from the Department of Computer Science and Engineering of the University of Ioannina, in 2022. Her research interests include data analysis, data visualization (especially spatial data and network data), applications development, data management etc.