Mestrado em Engenharia Informática
Dissertação
Relatório Final

Algorithms and Data Structures for Large Scale Geographic Information Systems

Bernardo Marques
basimoes@student.dei.uc.pt

Orientador:
Prof. Dr. Luís Paquete
Eng. Pedro Reino
Data: 29 de Janeiro de 2016

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

University of Coimbra

Department of Informatics Engeneering

M.Sc Thesis

Algorithms and Data Structures for Large Scale Geographic Information Systems

Author:
Bernardo Marques

Supervisors:
Prof. Dr. Luís Paquete [1]
Eng. Pedro Reino [2]

Jury:
Prof. Dr. Carlos Fonseca [1]
Prof. Dr. Joel Arrais [1]

[1] *CISUC, Department of Informatics Engineering, University of Coimbra*
[2] *Smartgeo Solutions*

January 29st, 2016

Abstract

With the increasing availability of geographically referenced data, the efficiency of Geographic Information Systems (GIS) is becoming increasingly relevant. In this project, we address the problem of efficiently retrieving geographic data based on spatial relationships. We describe an implementation of a GIS solution based on Polygonal Map Quadtrees. This data structure is kept in memory while persistent storage is handled by an underlying database management system. To validate the implementation, we describe an experimental analysis in a wide range of scenarios. The experimental results suggest that this approach can have better performance than a state-of-the-art GIS framework in terms of running time.

keywords: Multi-Dimensional Indexing, Topological Relationships, Geographic Information Systems

Sumário

A tendência de crescimento da disponibilidade de informação, contendo esta um grande número de referências geográficas, torna premente a necessidade de tornar os Sistemas de Informação Geográfica (SIG) mais eficientes. A presente tese aborda o problema de, dado um conjunto de informação espacial, encontrar objetos nesse conjunto que verifiquem determinadas relações topológicas entre si. Neste documento, é descrita a implementação de um SIG baseado em *Polygonal Map Quadtrees*. Esta estrutura de dados é mantida em memória e o armazenamento em disco é gerido por um Sistema de Gestão de Base de Dados. De forma a validar a implementação, é descrita uma análise experimental considerando diferentes cenários. Os resultados obtidos indicam que a abordagem descrita neste documento poderá ter melhor desempenho que um dos SIG mais usados na atualidade em termos de tempo de execução.

palavras-chave: Indexação Multi-Dimensional, Relações Topológicas, Sistemas de Informação Geográfica

Contents

Chapter 1

Introduction

Considering the increasing volumes of spatially referenced data continuously generated by networks of people and devices, efficiency in Geographic Information Systems (GIS) is becoming more and more challenging. This efficiency challenge also impacts deeply the usability of many client applications. Therefore, it is necessary for the existing systems to adapt in order to improve the processing of such amounts of data.

This work is developed within the project "SGP - GIMS" funded by QREN whose goal is to develop a Geographic Information System that allows management and manipulation of georeferenced data. The project is a partnership between the Department of Mathematics and Department of Informatics Engineering from the University of Coimbra (UC) and Smartgeo Solutions, a company that develops GIS-based applications. The tasks developed at UC are concerned with the development of methods to solve particular problems that arise in GIS, such as the multi-objective shortest path problem, the traveling salesman problem with multiple salesmen and other constraints, finding representative sets of points in a finite point set and solving topological queries in large geographic datasets.

The goal of this work is to develop a GIS solution capable of managing and manipulating geographic data, while providing easy integration with different Database Management Systems (DBMS) and other platforms. Special emphasis is given to the efficiency of solving topological queries. These are queries that filter geographic data based on the topological relationships, i.e. queries that find all geographic data that verify a certain relationship with a reference geographic element. The relationships considered here are based on topological predicates such as *intersects* or *covers* as illustrated in Figure 1.1. This allows the construction of queries such as "find all points contained in polygon x", which produces the effect shown in Figure 1.2. Given that the amount of data can be very large, the practical efficiency of the algorithms and data structures is a central concern in this thesis.

Topological queries play a very important role in most GIS. They not only provide the end user with powerful searching features, as they also provide a way of verifying the consistency of the

FIGURE 1.1: Examples of relationships. In both images the geometric objects intersect, but on the right the blue circle covers the red one.

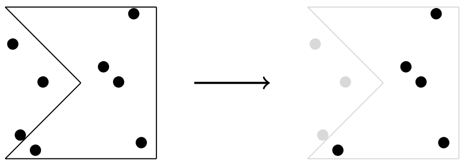


FIGURE 1.2: Example of a query: ”find all points covered by the polygon”.

spatial data. For example, if two polygons represent lakes, they should not intersect each other, otherwise the two would be a single lake. This type of inconsistency can be detected using topological queries. Along with this example, situations like countries overlapping each other or a restaurant in the middle of the ocean can also be detected with this type of queries.

This document describes the design and implementation of a system compliant with the described objectives. The document begins by introducing concepts and definitions. Most of the introduced concepts are based on standards published by the Open Geospatial Consortium (OGC), an organization that develops interface standards in the field of geospatial information [1]. Chapter 2 presents the requirement elicitation for this project in order to provide an objective definition of the problem addressed. From this list of requirements an architecture is developed and presented in Chapter 3. An experimental comparison between the developed system and postGIS (a database management system with spatial capabilities) is described in Chapter 4. Finally, in Chapter 5, conclusions and further research directions are discussed.

1.1 Geometric Objects

Given that the developed system has to communicate with other systems, such as Database Management Systems (DBMS) and other GIS, it is important to follow the applicable standards when defining the types of geometric objects supported. Therefore, the system supports the types of geometric object *point*, *linestring* and *polygon*, as defined in [14]. These objects are categorized in accordance to their Hausdorff dimension, thus the dimension of a point is 0, a linestring 1 and a polygon 2. Next, the properties of theses geometric objects are explained in detail.

A point is defined by its x and y coordinates, corresponding to latitude and longitude. A linestring is defined by a set of connected line segments, except for the first and last segment in the sequence. Finally, a polygon is defined by a ring, which is a set of connected line segments with no self-intersections, that defines its external border. Optionally, a polygon may contain internal rings that define holes in the polygon. There can be no intersection between any pair of rings that form a polygon.

Considering this description, the notions of interior, border and exterior of a geometric object are introduced as follows:

- The interior of a point is the point itself.
- A point contains no border.
- The exterior of a point is the entire domain except the point.
- The interior of a linestring is the union of all line segments that compose it except for their endpoints.
- The border of a linestring is the set of endpoints of the line segments that compose the linestring.
- The exterior of a linestring is the set of points that are not in its interior nor in its border.
- The interior of a polygon is the set of points contained by the external ring that are not contained by the internal rings. Moreover, any point lying in the line segments that compose the rings is not a part of the polygon's interior.
- The border of a polygon is the union of the line segments that compose the polygon's rings.
- The exterior of a polygon is the set of points of the domain that are neither in its interior nor in its border.

Throughout this document, whenever not explicitly indicated, it is assumed that an object is described by the union of its interior and its border.

1.2 Topological Relationships

Topological relationships can be defined as relationships between geometric objects that are invariant to transformations such as rotating, scaling or translating. Therefore, predicates such as includes, touches, contains or crosses are examples of possible topological relationships between two geometric objects. These relationships have been standardized by the Open Geospatial Consortium using the 9-intersection model[6]. In the following, an explanation of this model is provided.

The 9-intersection model is based on a 3×3 matrix in which each cell has some information about the intersection between the two geometric objects under consideration. Consider two sets of points A and B that represent two distinct geometric objects. Let A° denote the interior of A, ∂A the border of A and A^c the exterior of A. Moreover, the function $hdim(x)$ calculates the Hausdorff dimension of the argument point set x, whenever x is not the empty set. If x is the empty set, then $hdim(x)$ becomes $FALSE$. Therefore the range of the function hdim is $\{FALSE, 0, 1, 2\}$, since 3-dimensional object are outside of the function's domain. Considering this, the 9-intersection model is then based on the following matrix.

$$\begin{bmatrix} hdim(A^\circ \cap B^\circ) & hdim(A^\circ \cap \partial B) & hdim(A^\circ \cap B^c) \\ hdim(\partial A \cap B^\circ) & hdim(\partial A \cap \partial B) & hdim(\partial A \cap B^c) \\ hdim(A^c \cap B^\circ) & hdim(A^c \cap \partial B) & hdim(A^c \cap B^c) \end{bmatrix}$$

This matrix allows for the verification of a wide set of topological relationships between geometric objects using mask matrices that verify certain conditions. For example, in order to verify whether a geometric object contains another, it is possible to take the above matrix and compare it with the mask matrix shown below.

$$\begin{bmatrix} T & * & * \\ * & * & * \\ F & F & * \end{bmatrix}$$

Symbol $*$ means that the cell may assume any value, T means that the value of the cell has to be different from $FALSE$ (i.e. ≥ 0) and F means it has to be $FALSE$. Thus, the conditions represented by the matrix can be written as:

$$(hdim(A^\circ \cap B^\circ) \geq 0) \ \wedge \ (hdim(A^c \cap B^\circ) = FALSE) \ \wedge \ (hdim(A^c \cap \partial B) = FALSE).$$

This means that given two geometric objects A and B, if their interiors intersect and there is no intersection between the exterior of A and the interior or the border of B, the geometric object A contains B.

The verification of certain topological relationships requires knowledge about the dimension of the intersections and the objects under consideration. For example, if the goal is to find if two geometric objects overlap, it must be first guaranteed that the two geometric objects have the same Hausdorff dimension. After this, the mask matrices shown below may be used, matrix A for 0 and 2-dimensional objects and matrix B for the 1-dimensional cases.

$$\begin{bmatrix} T & * & T \\ * & * & * \\ T & * & * \end{bmatrix} \qquad \begin{bmatrix} 1 & * & T \\ * & * & * \\ T & * & * \end{bmatrix}$$
$$(A) \qquad\qquad\qquad (B)$$

In this case, in order to verify an overlapping relationship between two 1-dimensional objects (line segments), the intersection of their interiors must also be a 1-dimensional object (a line segment).

Finally, a topological query is defined by a set S of geometric objects to be filtered, a reference geometric object q, which is referred to as "query object" or "reference object" throughout the document, and a topological predicate p. From there, a topological query can be described as finding all objects of S that verify the topological relationship p with the reference object q.

1.3 Geographic Information Systems

Geographic Information Systems (GIS) are systems that deal with the capture, storage, manipulation and analysis of geographic data. Since the scope of this thesis is concerned with storing and searching spatial data, this section gives an overview on other GIS that provide similar features.

Some of the most used systems for storing and searching spatial data are PostGIS, Oracle Spatial and MySQL. These are relational databases with support for spatial indexing, allowing the creation of 2-dimensional indexes. There are several spatial indexing techniques available, which will be discussed in detail further on in this document. However, these systems rely mostly on indexes of the R-tree family, being Oracle Spatial the only that provides support for Quadtree based indexes [2, 3, 4]. Since these are different indexing structures, they have different properties and expected performances. Therefore, providing the end user with the possibility of choosing between different indexing schemes can have a significant impact on the performance, considering that only the user knows what operations will be performed on the data.

Considering that we are interested in large volumes of data, another important factor to take into account is the parallelization of work among several nodes in a cluster. An interesting implementation of such a solution is Hadoop-GIS. This software uses an R-tree index which is used as a basis for the Map Reduce algorithm, which distributes the work among the cluster nodes [5].

The system developed within the scope of this thesis differentiates itself from the ones described above because it is designed with the single purpose of fast topological querying in mind. The resulting system can therefore be described as an in-memory database with an indexing scheme optimized for querying.

Chapter 2

Requirement Elicitation

In this chapter, more precise definition of the problem is presented. In order to do so, the requirements elicited from Smartgeo are presented and discussed, and a general overview of the planned functionality is provided in this text.

The system is intended to work as a middleware between GeoServer and a collection of underlying Database Management Systems (DBMS). This means that the system should be able to connect and to fetch data from a DBMS and then, using this data, to answer topological queries from GeoServer. Therefore, GeoServer is a client of the system, using it as a source of geographic data. The main objective is to provide more efficient topological queries.

The requirements of this project are split into three groups: integration with GeoServer, integration with a DBMS and basic functionalities for geometry manipulation. In order to create an easy referencing scheme, each requirement has an associated ID. These IDs have prefixes depending on the nature of the requirement. Namely, core functionality requirements have prefixed "FE", DBMS integration requirements have prefixed "DB" and GeoServer integration requirements have prefixed "GS". After this prefix follows a sequential number with two digits.

2.1 Core Functionality

2.1.1 Functional Requirements

ID: FE01
Description: The system should recognize points, linestrings and polygons (as described in [14]) as geometric objects.

ID: FE02
Description: The system should provide a mechanism for solving queries relating a list of

$$Contains \quad \begin{bmatrix} T & * & * \\ * & * & * \\ F & F & * \end{bmatrix}$$

$$Covers \quad \begin{bmatrix} T & * & * \\ * & * & * \\ F & F & * \end{bmatrix} \quad \begin{bmatrix} * & T & * \\ * & * & * \\ F & F & * \end{bmatrix} \quad \begin{bmatrix} * & * & * \\ T & * & * \\ F & F & * \end{bmatrix} \quad \begin{bmatrix} * & * & * \\ * & T & * \\ F & F & * \end{bmatrix}$$

$$Intersects \quad \begin{bmatrix} T & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \quad \begin{bmatrix} * & T & * \\ * & * & * \\ * & * & * \end{bmatrix} \quad \begin{bmatrix} * & * & * \\ T & * & * \\ * & * & * \end{bmatrix} \quad \begin{bmatrix} * & * & * \\ * & T & * \\ * & * & * \end{bmatrix}$$

$$isContained \quad \begin{bmatrix} T & * & F \\ * & * & F \\ * & * & * \end{bmatrix}$$

$$isCoveredby \quad \begin{bmatrix} T & * & F \\ * & * & F \\ * & * & * \end{bmatrix} \quad \begin{bmatrix} * & T & F \\ * & * & F \\ * & * & * \end{bmatrix} \quad \begin{bmatrix} * & * & F \\ T & * & F \\ * & * & * \end{bmatrix} \quad \begin{bmatrix} * & * & F \\ * & T & F \\ * & * & * \end{bmatrix}$$

$$Meets \quad \begin{bmatrix} F & T & * \\ * & * & * \\ * & * & * \end{bmatrix} \quad \begin{bmatrix} F & * & * \\ T & * & * \\ * & * & * \end{bmatrix} \quad \begin{bmatrix} F & * & * \\ * & T & * \\ * & * & * \end{bmatrix}$$

$$Overlaps \quad \underset{\text{if } hdim(A) = 0 \vee hdim(A) = 2}{\begin{bmatrix} T & * & T \\ * & * & * \\ T & * & * \end{bmatrix}} \qquad \underset{\text{if } hdim(A) = 1}{\begin{bmatrix} 1 & * & T \\ * & * & * \\ T & * & * \end{bmatrix}}$$

FIGURE 2.1: 9 intersection matrix masks representing different topological relationships

geometric objects with a given reference geometric object through one of the topological relationships: *Contains*, *Covers*, *Intersects*, *is Contained*, *is Covered by*, *Meets* and *Overlaps.* These relationships are defined by the 9 intersection matrix masks in Figure 2.1
Dependencies: FE01

ID: FE03
Description: It should be possible to insert, update and remove geometric objects from the system.
Dependencies: FE01

2.1.2 Non-Functional Requirements

ID: FE04
Description: The time taken by the system to answer topological queries should be as minimal as possible. Specifically, the system should be significantly faster than postgres with the postGIS extension to answer topological queries.
Dependencies: FE02

2.2 Integration with Database Management Systems

2.2.1 Functional Requirements

ID: DB01
Description: The system should be able to connect to a postgres database with the postGIS extension, provided that the database is accessible to the system.

ID: DB02
Description: The system must be capable of fetching geographic data from a postgres database with the postGIS extension.
Dependencies: DB01

ID: DB03
Description: Changes in the data that occur in the system should be propagated to the underlying database.
Dependencies: DB01

ID: DB04
Description: Changes in the data stored in the database should be propagated to the system.
Dependencies: DB01

ID: DB05
Description: The communication between the system and the DBMS should be based in open standards in order to ease the implementation effort of extending the support to alternative DBMSs.

2.3 Integration with Geoserver

2.3.1 Functional Requirements

ID: GS01
Description: It is possible to connect the system and Geoserver. This connection must allow Geoserver to attempt topological queries and data editions.

ID: GS02
Description: When Geoserver attempts to use the system to solve a topological query, the system answers the request with the correct set of geometric objects. Whenever the requested query is malformed or generates an error within the system, a suitable error message must be sent back.
Dependencies: GS01, FE02

ID: GS03
Description: GeoServer should be able of inserting, updating and removing data from the system. When such operations result in errors, the system should provide GeoServer with a suitable error message.
Dependencies: GS01, FE03

2.4 Requirement Fulfillment

The project cannot succeed without the fulfillment of requirements FE01, FE02 and FE03, given that these represent the very basic operations of geographic data manipulation that must be supported by the system. Considering requirement FE04, the fulfillment of this requirement is essential to the success of this project, given that it stands as the main goal of the system. Therefore, the final implementation of the system must fulfill all these four requirements.

Regarding requirements GS01, GS02 and GS03, full integration with GeoServer can be a very time consuming and difficult task since it implies either the implementation of known communication standards or changing GeoServer's code in order to facilitate communication. Since these requirements are not essential for a working system usable by a third party, the final implementation of the system does not try to fulfill them. However, the architectural plan defines how the system can be expanded to fulfill these requirements.

Finally, the fulfillment of requirements DB01 and DB02 can be considered as essential since it is important to have the capacity of easily inserting existing data in the system. However, requirements DB03 and DB04 do not represent essential features to have in the final system, since a dynamic environment is not the main focus of the current project. Regarding requirement DB05, since the system will have to provide a way of communicating with PostGIS, it makes sense to fulfill this requirement.

Given this, the following statement defines the requirement fulfillment threshold above which the project is considered successful.

> Develop a system capable of managing and manipulating geographic data, in which topological queries are performed with significant better time performance than postGIS. Moreover, provide a standardized interface capable of fetching geographic data from PostGIS-based databases.

Therefore, the requirements fulfilled by the final system implementation are FE01, FE02, FE03, FE04, DB01, DB02 and DB05. Regarding the remaining requirements, the architecture of the system defines ways of extending the system in order to provide the functionality described in the requirements. In order to reach the stated threshold an architecture is developed and presented in the next chapter.

Chapter 3

System Design

The current chapter proposes an architecture to comply with the requirements discussed in the previous chapter. The source code corresponding to this architecture can be found in [17].

In the previous chapter, three sets of requirements are discussed, namely, core functionality requirements, integration with an underlying DBMS requirements and integration with Geoserver requirements. Each of these sets of requirements is translated to layers in the top-level architecture diagram presented in Figure 3.1.

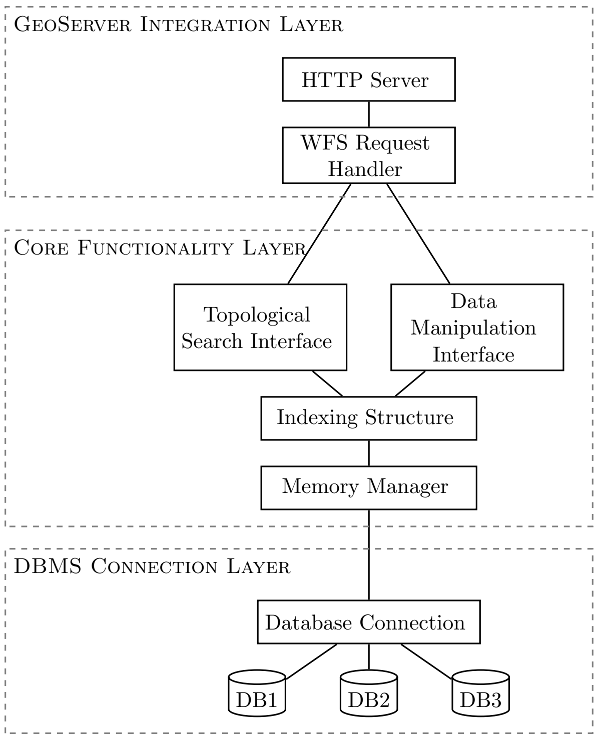


Figure 3.1: Top-level system architecture design.

In the following sections, the approaches taken to solve the particular problems addressed by each layer are presented.

3.1 Core Functionality – Data Model

This section defines how geographic data is represented within the system. This is an important subject that requires special care since the system is integrated with other GIS applications and the data model needs to be compatible. For this reason, the data model used by the system is based on the models proposed in the applicable standards [14]. This data model is presented in Figure 3.2 as a class hierarchy.

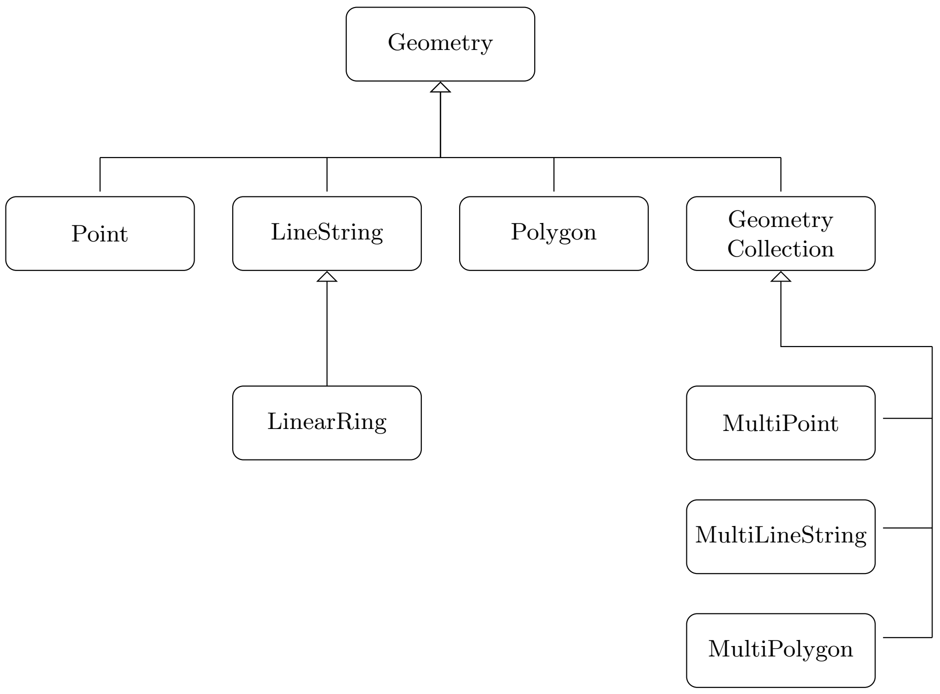


FIGURE 3.2: Geometry types and their inheritance relationships.

As shown, the type *Geometry* is used as a basis for all others. Since it is necessary to uniquely identify the different geometric objects both internally and in the underlying DBMS, the class geometry has the properties *internal_id* and *external_id*. Where *internal_id* represents the id used within the system to identify the geometric object and *external_id* is the id used to identify the geometric object within the source DBMS.

The type *Point* is simply composed by its two coordinates (x,y). The type *LineString* is composed by a dynamic list of endpoints. The type *Polygon* is composed by a *LineString* that defines its external border and a dynamic list of *LineStrings* that represent its internal rings. Finally the type GeometryCollection is composed by a dynamic list of geometric objects. The types *LinearRing*, *MultiPoint*, *MultiLineString* and *MultiPolygon* do not have properties of their own. Instead they pose semantic constraints on the properties of the parent classes.

At this point, the data model for geometric data is defined. However, geographic data is also composed by non-geometric data and a map projection or geodetic datum. The indexing of non-geometric data is outside the scope of this project and is therefore not handled by the system (apart from the ids). Finally, since the only operations provided by the system are based on topological predicates, the map projection used is not relevant for the final results as long as all objects use the same one. Therefore, the system defines a global map projection and all indexed objects are converted from their original projection.

3.2 Core Functionality – Storing Spatial Data

This section discusses the data structure used by the system to index the geographic data. On simplest end of the spectrum, the data structure used would be a list. However, such solution would lead to an iteration over the entire dataset in order to answer a topological query. Therefore, it is necessary to investigate other data structures that provide better performance, especially for topological querying.

Spatial indexing is the process of organizing a collection of geometric data according to its positioning in the space. When searching for a particular set of geometric objects, this index allows an efficient scan of the data, without considering geometric objects that are unrelated and far away from the objects of interest. This indexing is usually achieved by partitioning the space into cells and associating geometric objects with the cells they intersect, or by creating hierarchically organized groups of objects.

This topic has been widely studied and several data structures have been designed for this purpose[11, 19, 20]. It is possible to categorize the approaches according to their properties[13]. Namely, whether it is possible for the cells that result from space partitioning to overlap or not (overlapping or disjoint cells), and whether the indexing structure partitions the space in regular pattern or in a data-dependent dynamic pattern.

For example, data structures such as the R-tree and R*-tree, group geometric objects and index the bounding box of the resulting collection in a hierarchical structure. This results in the possibility of the groups of objects having overlapping bounding boxes [11]. Therefore these data structures belong to the overlapping cells category. On the other hand, data structures such as the R+-tree, Polygonal Map Quadtree and Cell tree, divide the space in disjoint cells and associate with each one of them the geometric objects that intersect it in some way.

This two categories have inherent advantages and drawbacks. For example, when querying a data structure with overlapping cells, if the query intersects an area with overlapping cells, all of those have to be examined, which might ultimately result in the scanning of the entire database. Data structures based on disjoint cells, despite not having this problem, will most likely have higher memory requirements. This is because, in order to achieve disjointness, this data structures clip geometric objects against the cell. This results in a higher overhead, due to the storage of the clipped objects and due to possible repetition of edges that intersect multiple cells.

Within the group of data structures based on disjoint cells it is still possible to have data-dependent data structures and regular data structures. The first group includes data structures that create hierarchically organized groups of objects with disjoint bounding boxes at each hierarchical level, such as the R$^+$-tree and the Cell tree. These sort of decomposition is data-dependent since the shape of the cells depends on the dispersion of the data throughout the space. In this way, operations that require the combination of two indexing structures, force the recalculation of a whole new indexing structure, such as the map overlay operation in which two maps are placed over each other. The second group includes data structures that partition the space in regular patterns, such as the Uniform Grid and the Quadtree. The Uniform Grid divides the space into similar rectangles, while the Quadtree adapts the decomposition to the distribution of the data and divides the space in powers of two. Given the uniformity of these approaches, it is fairly easy to perform operations such as the map overlay with them.

According to an empirical study performed by Hoel and Samet in [13], data structures based on disjoint cell have better performance for query operations. Moreover, since the map overlay operation can be relevant to combine two or more layers of geographic data when solving a query, it is also relevant to choose a data structure with regular partitioning. Therefore, the implementation described in this document uses a Polygonal Map Quadtree to store the spatial data. The Uniform Grid is discarded because it can be seen as a particular case of a polygonal map quadtree but has a less powerful subdivision criterion.

Finally, the remainder of this section provides an brief overview over the functionality of the Quadtree.

3.2.1 Polygonal Map Quadtree

The Polygonal Map Quadtree is a spatial indexing structure due to Hanan Samet [19]. It begins with a bounding box that encloses the entire set of geometric objects to be indexed, which is the domain of the root node of the tree. Whenever the geometric objects that intersect a given node (and therefore are indexed in that node), violate a given criterion, the node is split in four equal quadrants and the geometric objects distributed among them accordingly. Each of the new quadrants corresponds to a leaf node whose parent is the node that has just been split.

Since the nodes are split in four disjoint quadrants it is possible to say that this data structure is based on a regular partition of the space in disjoint cells. That is, all nodes have size proportional to the tree domain scaled down by some power of two and no two cells at the same hierarchical level overlap each other.

Construction

The Quadtree begins with an empty root node whose bounding box covers all geometric objects. The objects are then inserted one by one in this node.Whenever a certain criterion is violated, the node is split in four equal quadrants. The geometric objects stored at the split node are then inserted in its child nodes. A geometric object is indexed in a node if and only if it intersects the box that defines the node domain.

The insertion algorithm is therefore recursive. The process begins at the root node and then recursively proceeds to the child nodes that intersect the geometric object being inserted. The geometric object is then added to each leaf node it intersects, resulting in node splits whenever a node is considered invalid. The validity of a node may be defined in several ways. In the approach proposed by Samet, three possible criteria were defined, resulting in different time complexities and properties. The criterion considered in this section states that at most one point may lie within the region defined by the bounding box of a leaf node.

Given this criterion and considering that the nodes are split in regular patters, in order to separate two infinitely close points the tree would have infinite length. Thus, the depth of the tree is inversely proportional to the distance between the closest pair of points within the set of geometric objects. According to [19], the maximum depth of the tree can be calculated with Equation 3.1, where d denotes the distance between the closest pair of vertexes, if the domain of the tree is a unit square.

$$Depth = 1 + \log_2\left(\frac{\sqrt{2}}{d}\right) \tag{3.1}$$

Geometry Searching

Searching the Quadtree implies finding all leaf nodes that intersect the given query geometry. Therefore, if the query geometry is a point, the number of steps leading to the respective nodes are proportional to the tree depth, since the maximum number of leaf nodes intersected by the point is constant. As for lines and polygons, the number of intersected leaf nodes is not constant. Therefore the complexity of searching for geometric objects of these types is proportional to the number of intersected leaf nodes multiplied by the tree depth. This is illustrated in Figure 3.3, where the query geometric is a line segment and the intersected leaf nodes are highlighted in blue.

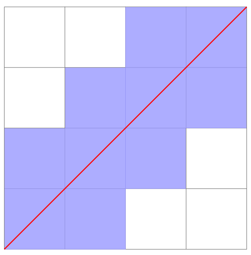


FIGURE 3.3: A line segment query with the retrieved nodes highlighted in blue.

Finally, in order to perform range query in a Polygonal Map Quadtree, all nodes that intersect the border of the query rectangle and all nodes that are strictly contained in it must be examined, as illustrated in Figure 3.4 where the blue cells are intersected by the border of the query window and the green cells are strictly contained in it. It is possible to verify that the number of nodes whose geometric objects have to be reported is proportional to the area of the query window. Therefore, for very large query windows it becomes less computationally demanding to iterate over a list of the indexed geometric objects than to use the Quadtree index.

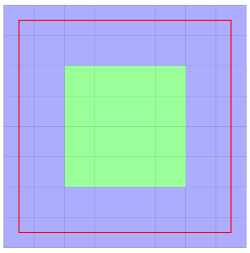


FIGURE 3.4: Example of a range query in a Polygonal Map Quadtree.

3.3 Core Functionality – Solving Topological Queries

Topological queries have been a core problem in spatial databases for a long time. In order to address the problem, a pipeline is proposed by Kriegel et al.[15]. This pipeline divides the process in three steps:

1. use an indexing data structure to discard some of the objects that are disjoint with the reference object.
2. use simpler versions of the geometric objects (i.e. approximations) to quickly discard some of the objects that are disjoint with the reference object.
3. calculate the topological relationships between the remaining objects and the reference object.

This pipeline is depicted in Figure 3.5. The remainder of this section discusses the details of each the three steps shown in the pipeline: index, geometric filter, exact shape processor.

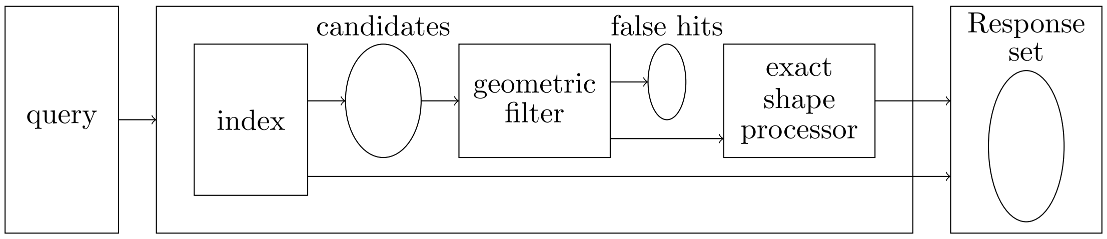


FIGURE 3.5: Query processing pipeline

3.3.1 Indexing Structure

Taking advantage of the Quadtree indexing scheme, it is possible to take into consideration only the geometric objects stored in certain nodes of the tree. Moreover, since the geometric objects are clipped to the nodes, it is possible to consider only portions of the original geometric objects in order to derive conclusions regarding the 9 intersection matrix between the two objects.

In Figures 3.6, 3.7 and 3.8 the nodes relevant to the construction of a 9-Intersection Matrix (9-IM) between two objects are depicted. The nodes intersected by the border of the reference object (Figure 3.6) have references to all geometric objects whose border intersects the reference object. Nodes contained in the reference object (Figure 3.7) have references to the objects that are contained in the reference object. Finally, all objects with Hausdorff dimension 2 that contain the reference object are referenced in at least one of the nodes intersected by a straight line coming from any point of the reference object and going in any direction (Figure 3.8). This use of the Quadtree makes it possible to avoid scanning the entire dataset of geometric objects.

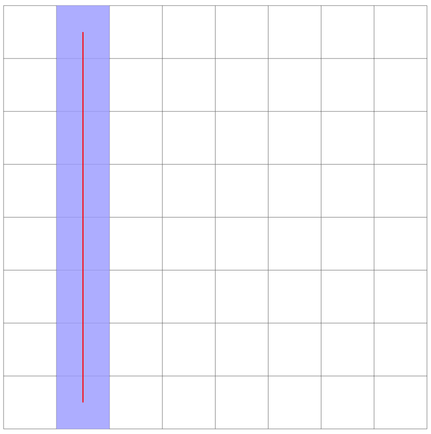


FIGURE 3.6: Nodes intersected by the border of the query object (highlighted in blue)

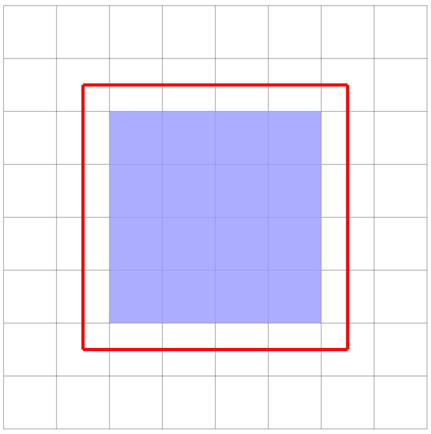


FIGURE 3.7: Nodes strictly contained by the query geomerty (highlighted in blue)

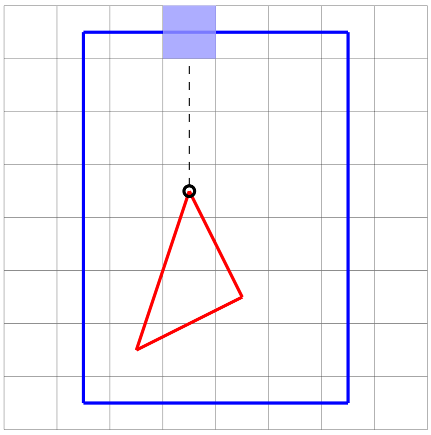


FIGURE 3.8: Nodes intersected by polygons that contain the query object (highlighted in blue). These nodes can be found by iterating over all nodes that are intersected by an half line that starts at one of the points of the query geometry and going in any direction.

3.3.2 Geometric Filter

The geometric filter step consists of using geometric approximations of the geometric objects to reach conclusions about their intersection. It is intended as a shortcut to avoid using more complex algorithms to evaluate the intersection between the two original objects.

Within the scope of the present project, geometric approximations are simplifications of some original geometric object which can either be a linestring or a polygon. These approximations can be classified according to certain topological relationships with the original object in two relevant categories [10], discussed below.

Conservative Approximations

Conservative approximations are simplifications that cover the original object, i.e. no point of the interior or border of the original object is also a point of the exterior of the approximation [10]. Given this, if two approximations do not intersect, the matching original objects do not intersect. Moreover it is possible to derive other conclusions from the intersection of two approximations taking into consideration their "false area". The "false area" of an approximation is the area of the intersection between the approximation and the exterior of the original object, i.e. the area of the approximation that does not intersect the original object as illustrated in Figure 3.9.

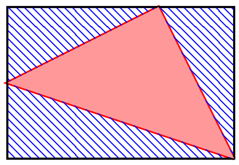


Figure 3.9: conservative approximation of a red triangle with "false area" dashed in blue

Consider two objects A and B with respective approximations A' and B'. Moreover, consider a function $a(x)$ that calculates the area of object x and a function $f(x, y)$ that calculates the "false area" of approximation x regarding the original object y. If $a(A' \cap B') \geq \min(f(A', A), f(B', B))$ then objects A and B must intersect. It is also possible to derive the possible intersection matrices depending on whether the left side is strictly greater than the right side of the inequality.

If $a(A' \cap B') > \min(f(A', A), f(B', B))$ and no approximation covers the other, the following matrix is obtained:

$$\begin{bmatrix} 2 & 1 & 2 \\ 1 & \leq 1 & 1 \\ 2 & 1 & 2 \end{bmatrix}$$

If $a(A' \cap B') > \min(f(A', A), f(B', B))$ and one of the approximations covers the other, the following matrix is obtained:

$$\begin{bmatrix} 2 & 1 & * \\ 1 & \leq 1 & * \\ * & * & 2 \end{bmatrix}$$

The quality of a conservative approximation is inversely proportional to its area. This is due to the fact that as the area of the approximations is minimized, the probability of two approximations being disjoint increases.

Progressive Approximations

Progressive approximations are simplifications which are covered by the original object, i.e. no point of the interior or border of the approximation is also a point of the exterior of the original object [10]. Thus, this type of approximation applies only to polygons. Since such an approximation is covered by the original polygon, if two progressive approximations intersect each other then the original objects must also intersect. It is then possible to derive the following part of the intersection matrix:

$$\begin{bmatrix} 2 & * & * \\ * & * & * \\ * & * & 2 \end{bmatrix}$$

Moreover, it is possible to conclude that polygon A contains polygon B if a progressive approximation of A contains a conservative approximation of B.

The quality of a progressive approximation is directly proportional to its area. This is due to the fact that, by increasing the area of progressive approximations, the probability of them intersecting is increased.

Approximations and Query Performance

The impact of approximations in the performance of query processing is not only proportional to the quality of the approximation, but also inversely proportional to the amount of operations needed to calculate the intersection between two approximations. This relation is empirically shown in [10], where different conservative approximations are taken and compared in order to verify their impact on query performance. In particular, among the conservative approximations compared in this publication, the convex hull and the minimal convex polygon with 5 corners (5-gon) were considered. While the convex hull is the optimal convex conservative approximation in terms of area minimization, it was outperformed by the minimal 5-corner convex polygon [10]. This can be considered a solid grounding of the previous observation, since the convex hull has a variable, and in average higher, number of vertexes, which lead to a higher number of operations when calculating intersections.

In the implementation of the system, only convex conservative approximations are considered, since conservative approximations provide better results than conservative approximations and calculating the intersection between non-convex polygons ($O(N \log N)$) is less efficient than between convex polygons ($O(N)$).

In order to decide which approximation type performs best in the system, the minimum bounding box, minimum bounding convex N-gon (with N ranging from 4 to 8) and the convex hull were compared against each other. The benchmark consisted in 2500 different queries of the type "find all polygons that intersect the reference polygon". The calculations of false area consider all the 2500 different reference polygons used in the queries. Finally, the number polygons in the indexing structure is slightly above 350k. The data used is from Open Street Maps regarding the geographic region of Portugal [16]. Table 3.1 shows the average percentage of false area of each type of approximation is shown. The results indicate that the minimum bounding rectangle has a great disadvantage in comparison to the remaining approximations. Given that the convex hull is the most compact convex approximation available, it is, as expected, the one with lower false area percentage.

Approximation Type	Average False Area (%)	standard deviation (%)
Minimum Bounding Rectangle	49.48	19.48
Minimum Bounding 4-gon	18.58	20.78
Minimum Bounding 5-gon	15.55	20.15
Minimum Bounding 6-gon	14.63	19.96
Minimum Bounding 7-gon	14.28	19.90
Minimum Bounding 8-gon	14.11	19.89
Convex Hull	13.94	19.88

Table 3.1: Average percentage of false area per approximation type.

In Table 3.2 the average time taken to answer the benchmark queries is shown. The results show a speed-up of 4 by using approximations. It is also worth noting that the results do not confirm the impact of the number of operations for calculating the intersection, given that the best results are obtained using the convex hull.

Approximation Type	Average Query Time (ms)	standard devitation (ms)
No Approximation	31.31	39.94
Minimum Bounding Rectangle	7.39	38.80
Minimum Bounding 4-gon	6.34	19.06
Minimum Bounding 5-gon	5.82	18.52
Minimum Bounding 6-gon	5.41	17.89
Minimum Bounding 7-gon	5.30	17.92
Minimum Bounding 8-gon	5.30	17.99
Convex Hull	5.21	18.29

Table 3.2: Average time for polygon intersection queries per approximation type.

Table 3.3 shows the time taken by the system to retrieve the polygons from the database and to calculate the corresponding approximation. The calculation of the minimum bounding rectangle has the least computational overhead, followed by the convex hull. From there, the time rises from the 8-gon to the 4-gon. This is due to the fact that in order to calculate a minimum bounding N-gon, the convex hull is firstly calculated and endpoints are iteratively removed until the expected number of vertexes is reached.

Approximation Type	Retrieval and Calculation time (seconds)
Minimum Bounding Rectangle	11.15
Minimum Bounding 4-gon	44.43
Minimum Bounding 5-gon	40.34
Minimum Bounding 6-gon	35.21
Minimum Bounding 7-gon	31.91
Minimum Bounding 8-gon	29.68
Convex Hull	15.02

Table 3.3: Time taken to retrieve 353484 polygons from the database and calculate their approximation.

Finally, Table 3.4 shows the time taken by the system to index the polygons in the Quadtree. In this case, the minimum bounding rectangle shows the best results. This can be explained as follows: considering that the domains of the nodes are rectangles, it is intuitive that the extent of the approximations (i.e. the minimum bounding box of the approximation) has a great importance. This is due to the fact that since the world is divided in cells, any difference in false area that is less than the area of the smallest cell is irrelevant in terms of performance. Moreover, the regularity of this cells and the regularity of the minimum bounding box offer greater advantage. This factors allied to the greater simplicity of the minimum bounding box explain the results obtained.

Approximation Type	Indexing time (seconds)
Minimum Bounding Rectangle	6.89
Minimum Bounding 4-gon	15.71
Minimum Bounding 5-gon	16.64
Minimum Bounding 6-gon	17.30
Minimum Bounding 7-gon	17.67
Minimum Bounding 8-gon	18.01
Convex Hull	19.14

Table 3.4: Time taken to index 353484 polygons in the Quadtree.

It now possible to conclude that the convex hull is the approximation type that provides best query performance in our practical scenario. However, the indexing time and the calculation time pose an overhead, but since there is no requirement regarding the efficiency of insertions, this is not a problem.

3.3.3 Exact Geometry Processing

If a geometric object reaches the final stage of the pipeline, the system needs to calculate the intersection matrix using the exact shapes of that geometric object and of the reference object. Therefore, this section introduces the algorithms used for calculating intersections between different types of geometric objects.

The algorithms used to calculate the intersection between two geometric objects must also take into account that the geometric objects are defined within a constrained domain, which is the square that defines the corresponding quadtree node. This means that if the intersection between two polygons is being calculated, the algorithm should only report the intersections that occur within the rectangular domain of the corresponding quadtree node.

point vs. point intersection The intersection between two points can be easily calculated by verifying whether the two points are equal or not.

point vs. linestring intersection The intersection between a point and a linestring can be calculated by verifying whether or not the point lies in one of the line segments that compose the linestring. Moreover, it is also necessary to verify whether the point is equal to one of the endpoints of these line segments, since the endpoint constitute the border of the linestring.

point vs. polygon intersection The intersection between a point and a polygon can be calculated using a point location algorithm, that verifies whether the point lies in the interior, border or exterior of the polygon. The algorithm implemented by the system is as follows:

Given a polygon P, a point p and a domain D:

1. Find the edge of $P \in D$ that is closest to p
2. If two edges are equally distant and share a common endpoint c, choose the one that forms the least internal angle with the edge p–c at point c.
3. Taking into account whether the chosen edge belongs to an internal or external ring, verify if p lies to the internal side of the edge.

linestring vs. linestring intersection

Finding the intersections between two sets of line segments is a well-known problem. In the particular case of linestrings there are certain properties that must be taken into account before designing and implementing such an algorithm. First, the data is mainly composed by sequences of connected line segments (may not be totally connected due to being clipped to the domain of the quadtree node). Secondly, the problem consists of finding intersections between two sets of line segments, i.e. intersections between segments of the same set are not relevant.

Let S_1 and S_2 denote two sets of line segments of size N_1 and N_2 respectively. Considering a brute-force approach to the problem, verifying intersections between each segment of S_1 and all segments in S_2 would take $O(N_1 \times N_2)$ time complexity. The number of intersections between S_1 and S_2 is at most $O(N_1 \times N_2)$, in a scenario where each line segment in S_1 intersects all line segments in S_2. Thus, in order to solve the problem more efficiently, the time complexity of the algorithms must be output-sensitive, that is, the complexity should depend on the size of the output in addition to the size of the input.

Considering the well-known algorithm proposed by Bentley and Ottmann in [9], applying this algorithm to geographic data would result in a time complexity of $O((N_1+N_2+K)\log(N_1+N_2))$, since the size of the input is the sum of the size of the two sequences. Moreover, term K corresponds to the number of intersections within the set $S_1 \cup S_2$. This term can therefore be as high as $(N_1+N_2)^2$, while the number of reported intersections in such worst case scenario would be $N_1 \times N_2$. Finally, since the data is composed mostly by sets of connected line segments, the number of intersections within the same set is expected to be at least equal to the size of the set minus one, if connections are considered as intersections.

Another algorithm for the calculation of intersections between line segments is the asymptotically optimal algorithm proposed by Balaban in [7]. This algorithm has a time complexity $O((N_1 + N_2)\log(N_1 + N_2) + K)$. Considering the same reasoning as before, K may reach as high as $(N_1 + N_2)^2$, making it less ideal in this particular scenario.

Since the two algorithms considered were not designed for problems with two different sets of line segments, also known as a red-blue intersection problem [8], the number of reported line segments poses an overhead to solve the problem. However, algorithms focused on this particular problem have been proposed, namely in [8]. The Heap Sweep algorithm is proposed to report the so called "purple" intersections, i.e. intersections between the two sets of line segments (red and blue). The time complexity of the Heap Sweep Algorithm is $O((N_1 + N_2 + K)\alpha(N_1+N_2)\log^3(N_1+N_2))$, where α denotes the slowly growing inverse Ackermann function. This algorithm however, despite having a non-optimal asymptotic behavior, relies on complex data structures of difficult implementation.

Finally, another algorithm for solving this problem would be an improved version of the brute-force approach. Such an algorithm consists in a simple line sweep, where the events are endpoints of the line segments and are ordered ascendantly in the x-axis. The algorithm then iterates over the event queue, and if the event corresponds to a leftmost endpoint, the corresponding line segment is added to the set of active line segments, whereas if the event corresponds to a rightmost endpoint, the corresponding line segment is removed from the set of active segments and the algorithm verifies whether that line segment intersects any of the active line segments. This leads to a $O(N_1 \times N_2)$ time complexity.

In order to choose the best approach for solving this problem within the system, the algorithms proposed by Bentley-Ottmann, Balaban and the improved brute-force algorithm were implemented and benchmarked. The results obtained are shown in Figure 3.10. The comparison shows that the improved brute-force algorithm has better performance, despite the worse asymptotic behavior. This can be due to the fact that the input sizes are relatively small and dependent on the constraints imposed on the number of vertex in each node of the quadtree. Given this small input size, the larger constants associated with the more complex data structures used by the Bentley-Ottmann and Balaban algorithms have an high impact on their performance. Moreover, the worst case scenario of this simple line sweep is $N_1 \times N_2$, whereas the worst case scenario of the other two algorithms can reach as high as $(N_1 + N_2)^2$ given that they also consider intersections between line segments of the same set.

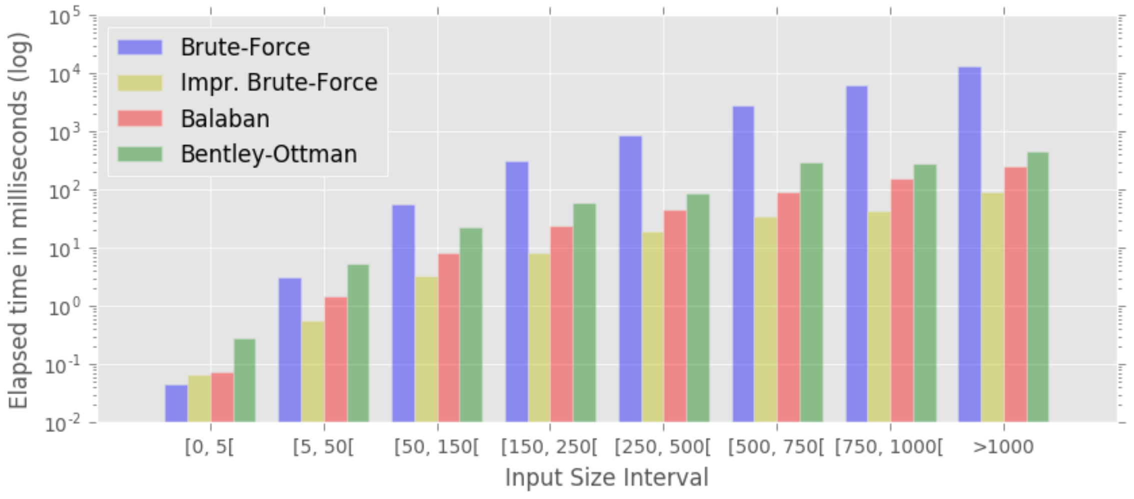


FIGURE 3.10: Average time taken by different algorithms to calculate line intersections on inputs within certain intervals

Using the same reasoning, i.e. small inputs sizes are solved faster by simpler algorithms, we found that using a regular brute-force algorithm for the smallest inputs results in better time performance. Therefore, the system uses a brute-force implementation for small inputs and the improved version for larger ones. The threshold that separates small inputs from large ones was determined through empirical experimentation. It was verified that the best results were obtained when the brute force algorithm was used for the cases where $N_1 < 20$ or $N_2 < 20$ and the improved version of the algorithm for all other cases.

polygon vs. polygon intersection

The calculation of the intersection between two polygons is accomplished with a 4-step algorithm:

1. clip each polygon to the rectangle that defines the domain.
2. build a planar graph (PSLG) with the resulting polygons.
3. model the planar graph as a Doubly Connected Edge List (DCEL).
4. calculate the 9-intersection matrix based on the DCEL.

The clipping of a polygon is accomplished by building a planar graph of the polygon and the rectangle that defines the domain. The planar graph is represented as a DCEL and the faces that are covered both by the domain and the polygon are considered as the clipped polygon. The result of this operation can be a collection of isolated points, lines and faces.

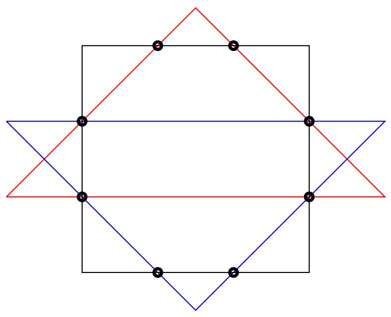
A DCEL is a data structure that contains a record for each face, edge and vertex of the planar subdivision defined by the planar graph. Since the data structure has information about the edges that bound each face, it is possible to detect which areas are bounded by which polygons, as illustrated in Figure 3.11, making it possible to calculate the 9-intersection matrix.

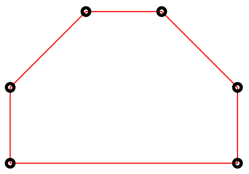
The clipped polygons are then passed as arguments to an algorithm that once again creates a planar graph representing the two polygons. This graph, again represented as a DCEL, contains information about the different faces, edges and vertexes and how they are shared between the two polygons. It important to note that it is necessary to take into account whether the edges of a clipped polygon are also edges of the original polygon. The reason is that a border intersection where one of the edges of the clipped polygons is actually part of the interior of the original polygon, is not a border intersection. Similarly, the process also needs to take into account whether a vertex of the clipped polygon is also a vertex of the original one. This process is illustrated in Figure 3.11.

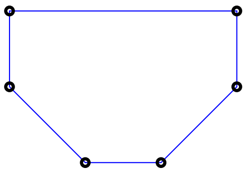
Finally, after having information about the faces, edges and vertex of the planar graph, it is possible to calculate the 9-intersection matrix using the set of rules described below. These rules apply to the calculation of a 9-Intersection Matrix between a polygon P_1 and a polygon P_2. Furthermore, the matrix cells are represented by two initials representing the three different parts of each polygon interior, border and exterior.

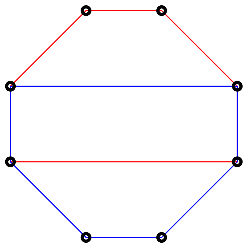
$II -$ If there is at least one face bounded by P_1 and P_2, then $II = 2$, $FALSE$ otherwise.

$IB -$ If there is at least one edge of P_2 separating two faces bounded by P_1 then $IB = 1$, $FALSE$ otherwise.


(A) Two polygons (red, blue) and the domain (black).


(B) The red polygon clipped.


(C) The red polygon clipped.


(D) planar graph of the two clipped polygons.

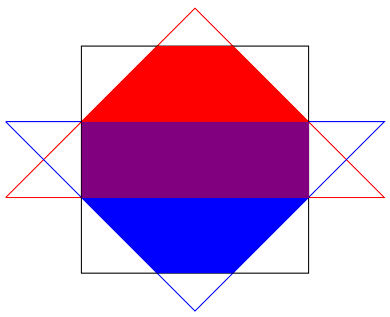

(E) Intersecting areas between the two polygons and the domain.

FIGURE 3.11: Illustration of the algorithm to compute intersections between polygons.

IE – If there is at least one face bounded by P_1 and not by P_2, then $IE = 2$, $FALSE$ otherwise.

BI – If there is at least one edge of P_1 separating two faces bounded by P_2, then $BI = 1$, $FALSE$ otherwise.

BB – If there at least one edge of the graph common to both P_1 and P_2 then $BB = 1$, else if there is at least one common node then $BB = 0$, $FALSE$ otherwise.

BE – If there is at least one face bounded by P_1 and not by P_2 then $BE = 1$, $FALSE$ otherwise.

EI – If there is at least one face enclosed by P_2 and not by P_1 then $EI = 2$, $FALSE$ otherwise.

EB – If there is at least one face enclosed by P_2 and not by P_1 then $EB = 1$, $FALSE$ otherwise.

EE – Since no polygon can cover the entire domain, $EE = 2$.

linestring vs. polygon intersection

Finally, for the case of linestrings and polygons, it is possible to apply a procedure similar to the one used in the polygons scenario. By building a DCEL representation of a Planar Graph of the linestring and the polygon clipped to the quadtree node domain, it is possible to derive conclusions using the set of rules described below, where P is the polygon and L is the linestring.

II − If there L separates two faces bounded by P or at least one endpoint of L lies inside P then $II = 1$, $FALSE$ otherwise.

IB − If there is a common edge between L and P then $IB = 1$, else if there is at least one node of L that is not an endpoint of L then $IB = 0$, $FALSE$ otherwise.

IE − If there is at least one endpoint of L that lies in the exterior of P then $IE = 1$, $FALSE$ otherwise.

BI − If there is at least one endpoint of L that lies in the interior of P then $BI = 0$, $FALSE$ otherwise.

BB − If there is a node common to L and P and that node is also an endpoint of L then $BB = 0$, $FALSE$ otherwise.

BE − If there is at least one endpoint of L that lies in the exterior of P then $BE = 0$, $FALSE$ otherwise.

EI − Since a linestring is not capable of covering a polygon, $EI = 2$.

EB − If there at least one edge of P that is not common to L then $EB = 1$, $FALSE$ otherwise.

EE − Since no linestring or polygon can cover the entire domain, $EE = 2$.

3.4 DBMS Connection

The Postgres project provides a library to manage connections with a Postgres databases. Using this library it is possible to perform queries, resulting in the ability of fetching and editing data. Since there is a need for independence in the way the system fetches data from the DBMS, the system is equipped with parser of Well Known Text (WKT)[12]. This parser is implemented using Lex and Yacc and recognizes the types *Point*, *MultiPoint*, *LineString*, *MultiLineString*, *Polygon*, *MultiPolygon*, *GeometryCollection*. The grammar and associated lexer file can be found in Appendix A.

Regarding the synchronization of data, providing propagation of changes performed in data indexed by the system to the DBMS consists solely in performing an *UPDATE* query over the DBMS connection. The propagation of data from the data requires a trigger to be associated with each and every database connected to the system. This trigger should be executed upon any change to relevant data (i.e. shared geometric data) and send a notification to the system with information about which data was changed and how.

3.5 GeoServer Integration

In order to integrate the system with GeoServer two possibilities arise:

1. Alter the code of GeoServer to provide integration with the system using a custom protocol
2. Implement a well-known protocol - Web Feature Service (WFS)[22]

While the first option can lead to a simpler and less extensive protocol, defining a protocol leads to several difficulties. It is easy to create an oversimplified protocol unable to describe slightly complex queries. Moreover, this would result in three time-consuming tasks: define the protocol, implement the client side in GeoServer and finally implement the Server side on the system. There is also the added complexity of understanding the implementation of a big open source project in order to be able to modify it.

Since the second option relies on a well established protocol, it is not necessary to define it. Moreover, GeoServer is already capable of connecting to an external WFS server off the shelf. This means that it is only necessary to implement the server side in the system. The big limitation of this approach is the large extension of the WFS standard. Nevertheless, this is the solution adopted in this architecture.

The WFS standard defines different types of requests, each with different purposes. These requests can both be encoded using xml and keyword-value pairs (KVP). In order to fulfill the requirements in Section 2.3, the requests that need to be implemented are shown in Table 3.5 with the corresponding requirements associated.

Request	Description	Requirement
getCapabilities	Describes the server, including the available functionalities and data.	GS01
describeFeatureType	Describes how a particular dataset is organized, including expected encodings used for input and output.	GS01
getFeature	Allows the client to fetch data from the server, providing support for topological queries.	GS02
transaction	Allows the client to create, modify and delete data.	GS03

Table 3.5: Description of the requests needed to fulfill the GeoServer Integration Requirements

It is worth noting that in order to implement the WFS standard, the system needs to be able to parse and write Geography Markup Language (GML)[18]. GML is an xml grammar capable of describing geographic information. In order to perform both the parsing and the construction, the implementation should take advantage of the functionalities available in the Geospatial Data Abstraction Library (GDAL).

Finally, topological queries are described using the Filter Encoding Specification [21]. This means that the implementation must also be able of parsing xml encoding filter requests.

Chapter 4

Experimental Results

This section describes an experimental analysis to the system in order to address two question:

1. What is the quality of the answers provided by system for the topological queries.
2. How does the system compare to postGIS in terms of time performance.

4.1 System Quality

In order to provide a measurement of the quality of the answers provided by the system, the chosen method is to compare them with those provided by postGIS. It is assumed that all answers provided by postGIS to topological queries are correct. In the following, we report the number of times that our approach presented a different result than that of postGIS. In order to have a memorable way of referring to these errors, non-existing relationships that are detected are considered "false positives", existing relationships that are not detected are considered "false negatives" and correctly identified relationships are considered as success. Tables 4.1 through 4.9 show the results for 600 sample queries. The error counting is done as follows: for a given query "find all objects y that verify relation r with object x", if our approach finds an object y that postGIS does not find, a false positive is counted. Similarly, if an object y is found by postGIS and not by our system, a false negative is counted. Finally, all objects y that are found by both systems are counted as "true positives" and "false negatives" are objects that are unrelated to the reference object and were not reported by our system. The tables present both the absolute count for each situation and the percentage it represents.

relationship	True Positivies	False Positives	False Negatives	True Negatives
intersects	4 $4 \times 10^{-6}\%$	0 0%	0 0%	91604396 100%

TABLE 4.1: queries between points

relationship	True Positivies	False Positives	False Negatives	True Negatives
coveredBy	5	0	0	348844195
	$1 \times 10^{-6}\%$	0%	0%	100%

Table 4.2: queries that given a point find all linestrings that are related

relationship	True Positivies	False Positives	False Negatives	True Negatives
coveredBy	804	1	0	212102195
	$4 \times 10^{-4}\%$	$5 \times 10^{-7}\%$	0%	99.9996%

Table 4.3: queries that given a point find all polygons that are related

relationship	True Positivies	False Positives	False Negatives	True Negatives
contains	126	0	0	91604874
	$1 \times 10^{-4}\%$	0%	0%	99.9999%
covers	132	0	0	91604868
	$1 \times 10^{-4}\%$	0%	0%	99.9999%

Table 4.4: queries that given a linestring find all points that are related

relationship	True Positivies	False Positives	False Negatives	True Negatives
contains	18452	0	0	348825747
	0.005%	0%	0%	99.9947%
coveredBy	3936	0	0	3936
	0.001%	0%	0%	99.9989%
covers	18452	0	0	18452
	0.005%	0%	0%	99.9947%
intersects	76111	31	0	76142
	0.002%	$8 \times 10^{-6}\%$	0%	99.9782%
overlaps	4237	0	0	4237
	0.001%	0%	0%	99.9988%
touches	30045	31	0	30076
	0.008%	$8 \times 10^{-6}\%$	0%	99.9914%

Table 4.5: queries between linestrings

relationship	True Positivies	False Positives	False Negatives	True Negatives
coveredBy	1372	0	0	212101628
	$6 \times 10^{-4}\%$	0%	0%	99.9994%
intersects	8196	0	0	212094804
	0.004%	0%	0%	99.9961%
touches	2302	0	0	212100698
	0.001%	0%	0%	99.9989%

Table 4.6: queries that given a linestring find all polygons that are related

relationship	True Positivies	False Positives	False Negatives	True Negatives
contains	31214	0	2	31216
	0.034%	0%	$2 \times 10^{-6}\%$	99.9659%
covers	31217	5	0	31222
	0.034%	$5 \times 10^{-6}\%$	0%	99.9659%

Table 4.7: queries that given a polygons find all points that are related

relationship	True Positivies	False Positives	False Negatives	True Negatives
contains	112913 0.032%	0 0%	2 6×10^{-7}%	112915 99.9676%
covers	114430 0.033%	7 2×10^{-6}%	0 0%	114437 99.9672%
touches	3497 0.001%	4 1×10^{-6}%	0 0%	3501 99.999%

Table 4.8: queries that given a polygon find all linestrings that are related

relationship	True Positivies	False Positives	False Negatives	True Negatives
contains	84881 0.04%	0 0%	3 1×10^{-6}%	84884 99.96%
coveredBy	1202 6×10^{-4}%	0 0%	0 0%	1202 99.9994%
covers	84881 0.04%	0 0%	0 0%	84881 99.96%
intersects	90300 0.043%	29 1×10^{-5}%	0 0%	90329 99.9574%
overlaps	1506 7×10^{-4}%	0 0%	0 0%	1506 99.9993%
touches	2912 0.001%	29 1×10^{-5}%	0 0%	2941 99.9986%

Table 4.9: queries between polygons

These results show that our system does not report the same results of postGIS, which is due to lower numerical precision of our system compared to that of postGIS. This smaller precision comes from the numerical instability associated with the use of floating point arithmetic.

In order to overcome this numerical instability, the implementation of the system uses an error margin $\epsilon = 1 \times 10^{-6}$ for comparison between coordinates, uses integer arithmetic by multiplying the floating numbers by 10^7 when operations such as summations and subtractions are performed, and finally, whenever coordinates are multiplied or divided, e.g. during the calculation of intersections between two lines, the implementation uses the GNU multiple precision library.

This approach to deal with numerical instability justify the lesser precision and the subsequent differences in results, since the described approach guarantees only up to 6 decimal digits of precision. For example, when the distance between a point and a polygon is less than ϵ, the point is considered to be on the border of the polygon. Similarly, if the distance between two polygons is less than ϵ, they will be considering as touching each other. Knowing that the implementation uses the "world mercator" map projection (srid 3395) and that a distance of 1×10^{-6} between two points subject to this projection amounts to less than 1×10^{-9} meters, the lesser precision should be acceptable.

4.2 Performance Comparison with postGIS

In order to compare the performance of the system against that of postgres with the postGIS extension, data from Open Street Maps regarding the geographic region of Portugal is used [16]. Similarly to the approach used before for the validation of the system, we consider 600 sample queries for each relevant combination of topological predicate, type of reference object (point, linestring or polygon) and the type of the objects that are reported.

These results were computed in a laptop equipped with an Intel Core i5 M 450 (2.4GHz, dual-core), 8GB DDR3 SDRAM and a solid state drive "Samsung 850 EVO", connected using SATA II technology. The operating system is a minimal Arch Linux installation without graphical environment. The source code is written in C++ and compiled using GNU Compiler Collection. Tables 4.10 through 4.18 present the average running times of both systems.

	postGIS		our system	
relationship	average time	standard deviation	average time	standard deviation
intersects	1.11	0.28	0.83	0.64

Table 4.10: average query time for queries that given a point find all points that are related to it.

	postGIS		our system	
relationship	average time	standard deviation	average time	standard deviation
coveredBy	15.48	8.64	0.91	0.64

Table 4.11: average query time for queries that given a point find all linestrings that are related to it.

	postGIS		our system	
relationship	average time	standard deviation	average time	standard deviation
coveredBy	2.01	0.96	1.69	1.30

Table 4.12: average query time for queries that given a point find all polygons that are related to it.

	postGIS		our system	
relationship	average time	standard deviation	average time	standard deviation
contains	2077.48	11456.26	3.35	7.68
covers	2082.11	11506.55	3.26	5.95

Table 4.13: average query time for queries that given a linestring find all points that are related to it.

Tables 4.10 through 4.18 show that for most topological queries, the system developed outperforms postGIS significantly. Namely, out of 26 types of queries, postGIS only showed better average performance in two and only showed lower standard deviation in two. This is justifiable with the fact that the developed system does not provide many of the features that postGIS does. Namely, there is no concern with persistent storage, the amount of precision is reduced to ϵ to avoid the heavy use of variable precision and there are no ACID transactions, to name

	postGIS		our system	
relationship	average time	standard deviation	average time	standard deviation
contains	7364.78	44229.91	416.40	554.92
covers	7227.70	43372.30	404.68	545.76
intersects	322.93	1679.19	387.07	517.38
coveredBy	336.38	1528.06	403.92	535.08
touches	7938.34	46420.29	398.23	525.11
overlaps	7934.83	46404.14	401.21	533.74

TABLE 4.14: average query time for queries that given a linestring find all linestrings that are related to it.

	postGIS		our system	
relationship	average time	standard deviation	average time	standard deviation
intersects	231.75	1326.19	142.56	247.59
coveredBy	189.99	1100.65	142.67	246.38
touches	5173.92	33905.16	148.09	258.77

TABLE 4.15: average query time for queries that given a linestring find all polygons that are related to it.

	postGIS		our system	
relationship	average time	standard deviation	average time	standard deviation
contains	32.21	565.96	2.42	13.11
covers	32.13	564.58	2.50	13.26

TABLE 4.16: average query time for queries that given a polygon find all points that are related to it.

	postGIS		our system	
relationship	average time	standard deviation	average time	standard deviation
contains	150.78	2502.86	80.49	134.81
covers	150.36	2498.52	77.26	130.64
touches	1998.74	36855.72	71.37	120.69

TABLE 4.17: average query time for queries that given a polygon find all linestrings that are related to it.

	postGIS		our system	
relationship	average time	standard deviation	average time	standard deviation
contains	159.67	2214.09	57.88	79.48
covers	158.95	2202.81	56.16	77.24
intersects	122.11	2133.86	55.31	73.26
coveredBy	138.81	2046.51	58.14	77.60
touches	1588.75	31069.58	52.23	71.91
overlaps	1589.17	31066.36	52.23	72.59

TABLE 4.18: average query time for queries that given a polygon find all polygons that are related to it.

a few. Moreover, the fact that the developed system uses an indexing structure with typical better querying performance, provides an added advantage.

An additional observation is the higher efficiency of point searching in our system, i.e. given a point finding the nodes of the index where it is contained. This conclusion can be drawn from Table 4.10, since the query "find all points that intersect point x" amounts to locating the nodes of the index and iterating over the objects stored in those nodes. It can be further seen that this process is approximately 1.34 times faster in the developed system. Given that this represents a recurrent operation in the indexing structure, having such better performance sets a good starting point to display better performance in other operations.

The running time is now related to some quantitative information related to the properties of the reference object, such as the area of its bounding box, its number of edges and the number of objects that are related to it considering the given predicate, i.e. the output size of the query.

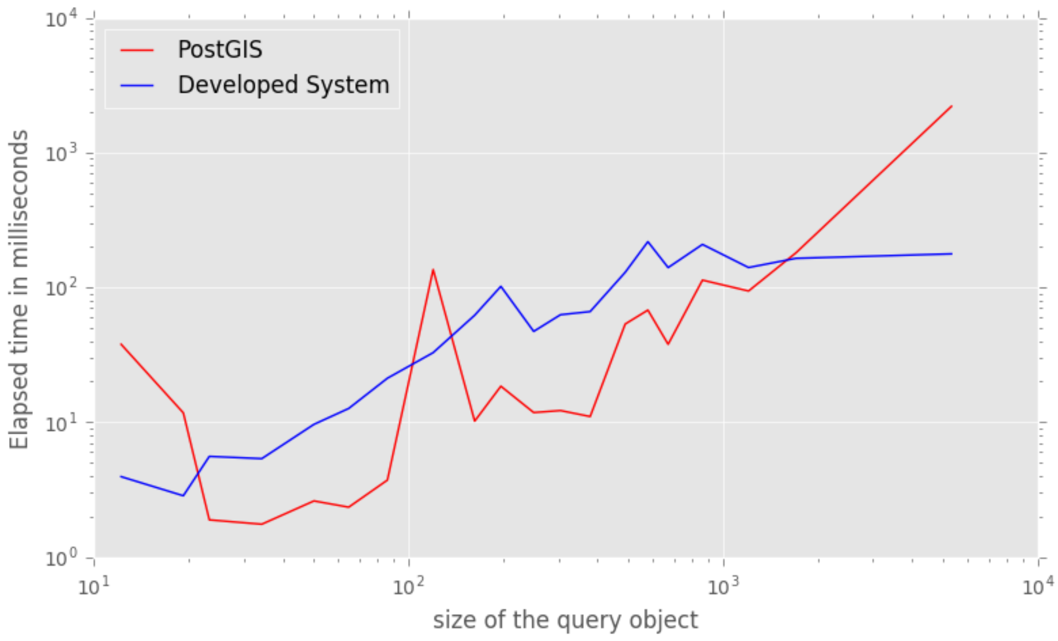


FIGURE 4.1: Average query time for the query "Find all linestrings contained by a reference polygon" with the number of edges of the reference polygon increasing along the x-axis.

It can be observed in Figures 4.1, 4.2 and 4.3 that postGIS shows an higher sensitivity to all three variations, since the respective curve usually begins lower and ends higher. Firstly, when the number of edges in the query polygon increases, the query time increases faster in postGIS than in the implemented system. This can possibly be caused by the fact that the Quadtree data structure clips the geometric objects to the nodes, therefore performing the intersection calculations between portions of the objects, thinning the impact of the edge number increase. Secondly, the impact of the output size in the implemented system is weaker than in postGIS. Finally, the impact of the area increase is more evident in postGIS. This can be explained by referring back to Table 4.10, which shows that the developed system has significantly higher performance on searching for the nodes that intersect a given point. Given that increasing

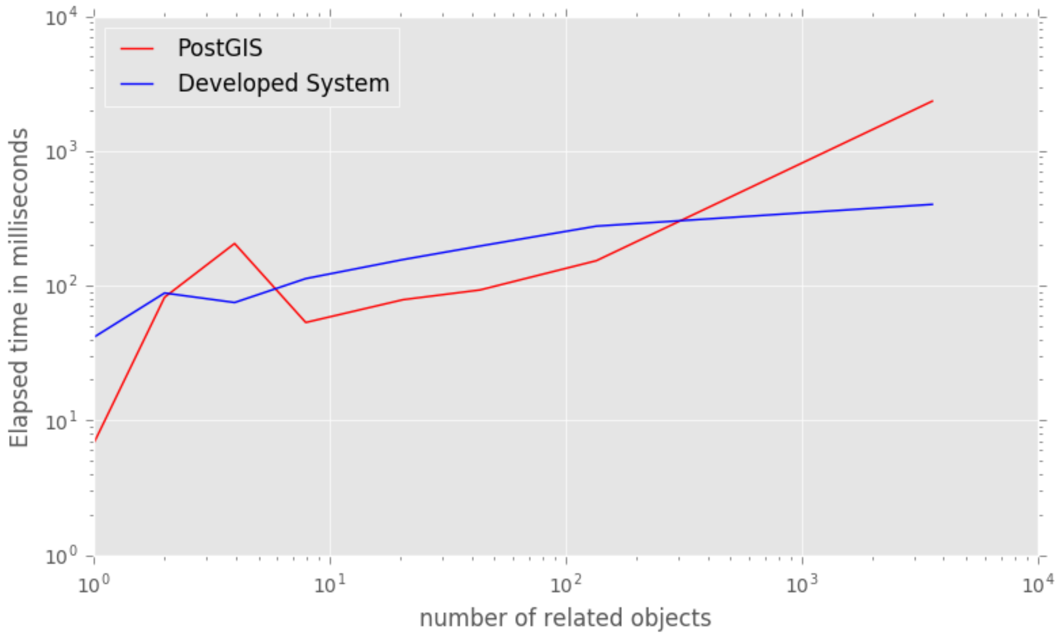


FIGURE 4.2: Average query time for the query "Find all linestrings contained by a reference polygon" with the number of contained linestrings increasing along the x-axis.

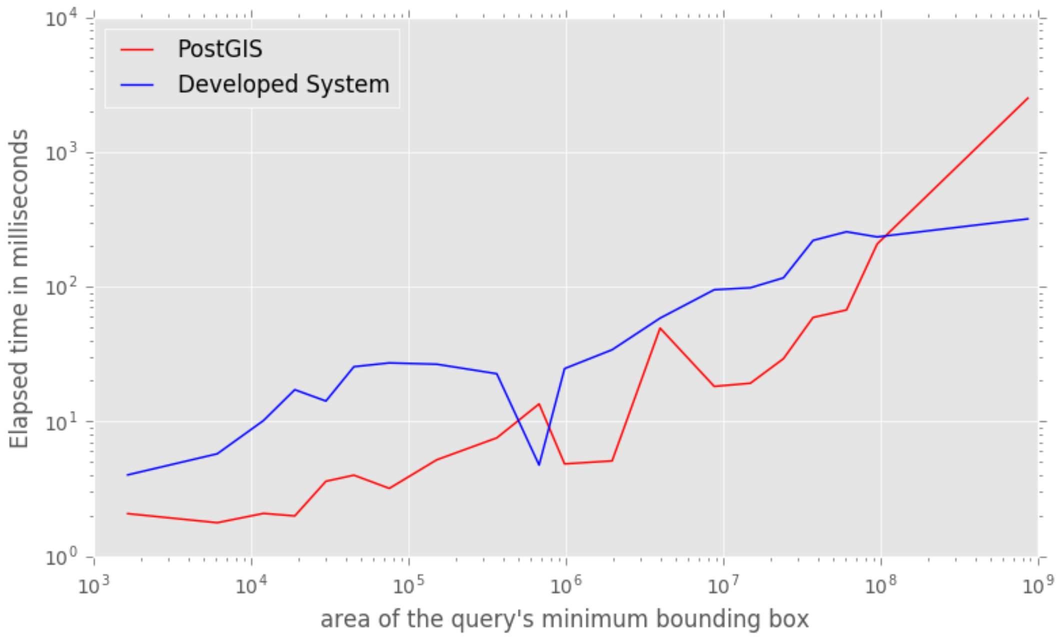


FIGURE 4.3: Average query time for the query "Find all linestrings contained by a reference polygon" with the area of the polygon's bounding box increasing along the x-axis

the area of the query polygon amounts to increasing the number of search points, the different behavior is understandable.

Finally, it is also observable that postGIS performs generally better than the developed system when the reference objects are smaller. This can be an indication of large constants in the implemented system, in comparison to postGIS.

Chapter 5

Conclusions and Future Work

This work describes a system capable of fetching geographic data from a spatial database using a standardized language (WKT). Moreover, this system is capable of indexing this data and answer topological queries on it. Moreover, as a by product, a geometry library with good intersection support was developed. Finally, when compared to postGIS, another tool equipped with the same functionality, the performance of the system was slightly superior. This set of contributions is summarized in the list below.

- ”well known text” (WKT) parser
- implementation of two dimensional data indexing using a Polygonal Map Quadtree
- implementation of a geometry library with extensive intersection support
- implementation of a topological query engine
- assessment of the practical performance of algorithms for intersection detection between two sets of line segments, namely a brute-force approach, an improvement over the brute-force approach using a line sweep, the algorithm proposed by Balaban [7] and the Bentley-Ottmann algorithm.
- assessment of the practical performance implications of using different approximations to polygons, namely the minimum bounding box, the convex hull and the minimum bounding convex N-gon with N ranging from 4 to 8.

It is also possible to draw some relevant conclusions from this project. Namely, as stated above, the performance gain in comparison to postGIS, which can be due to a variety of factors, including using a different indexing structure and disregarding certain functionalities such as persistent storage or ACID transactions. Moreover, some of the practical results obtained were somewhat surprising, for example, the convex hull being the approximation with the best performance and the improved brute-force algorithm for linestring intersection having best performance. These conclusions are summarized in the list below.

- The developed system achieved a very good performance when compared to postGIS.
- The improved version of the brute force algorithm for calculating the intersection between linestrings shows very good performance.
- The performance of topological querying is greatly impacted by the quality of the geometric approximations to the geometric objects involved.

In summary, we consider this project to be successful, given that the requirements we committed to fulfill were achieved. However, we do not consider this project to be production ready due to the lack of white box testing, despite the black box testing against postGIS.

The next steps could be in the direction of implementing the remainder of the requirements. Namely, implementing the Web Feature Service standard and implementing propagation of changes in the data between the database and the system. Moreover, extensive white box testing should be developed.

Considering a more scientific direction, the continuation of the project could contemplate the assessment of the practical performance change of using approximations composed by multiple shapes. For example, instead of using the convex hull, using a polygonal covering of the geometric object composed by a variable number of rectangles, circles or any other simple shape.

Bibliography

[1] About OGC. `http://www.opengeospatial.org/ogc`. Accessed: 27-1-2016.

[2] MySQL 5.7 Reference Manual, creating spatial indexes. `http://dev.mysql.com/doc/refman/5.7/en/creating-spatial-indexes.html`. Accessed: 10-1-2016.

[3] Oracle spatial documentation, spatial concepts - indexing of spatial data. `https://docs.oracle.com/cd/B10501_01/appdev.920/a96630/sdo_intro.htm#i877656`. Accessed: 10-1-2016.

[4] PostGIS 1.5 Manual, data management and queries - building indexes. `http://postgis.net/docs/manual-1.5/ch04.html#id361810`. Accessed: 10-1-2016.

[5] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, and J. Saltz. Hadoop-GIS: a high performance spatial data warehousing system over map reduce. *Proceedings of the VLDB Endowment*, 6(11):1009–1020, 2013.

[6] Anonymous. OpenGIS simple features specification for SQL. Technical report, Open Geospatial Consortium, 1999.

[7] I. J. Balaban. An optimal algorithm for finding segments intersections. In *Proceedings of the eleventh annual symposium on Computational geometry*, pages 211–219. ACM, 1995.

[8] J. Basch, L. J. Guibas, and G. D. Ramkumar. Reporting red-blue intersections between two sets of connected line segments. In *Proceedings of the Fourth Annual European Symposium on Algorithms*, pages 302–319. Springer, 1996.

[9] J. L. Bentley, T. Ottmann, et al. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers*, 100(9):643–647, 1979.

[10] T. Brinkhoff, H.-P. Kriegel, and R. Schneider. Comparison of approximations of complex objects used for approximation-based query processing in spatial database systems. In *Proceedings of Ninth International Conference on Data Engineering*, pages 40–49. IEEE, 1993.

[11] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *International Conference on Management of Data*, volume 14, pages 47–57. ACM, 1984.

[12] J. Herring. OpenGIS implementation standard for geographic information-simple feature access-part 1: Common architecture. 2011.

[13] E. G. Hoel and H. Samet. A qualitative comparison study of data structures for large line segment databases. In *ACM SIGMOD Record*, volume 21, pages 205–214. ACM, 1992.

[14] T. ISO. Geographic information–spatial schema. Technical report, Second Draft of ISO 19107 (15046-7), International Organization for Standardization, 1999.

[15] H.-P. Kriegel, T. Brinkhoff, and R. Schneider. Efficient spatial query processing in geographic database systems. *IEEE Data Engineering Bulletin*, 16(3):10–15, 1993.

[16] O. S. Maps. Open street maps data for the region of portugal. `http://download.geofabrik.de/europe/portugal.html`.

[17] B. Marques. Gims - topological query solver. `https://bitbucket.org/futuo/gims`.

[18] C. Portele. OpenGIS geography markup language (gml) encoding standard. Technical report, Open Geospatial Consortium, 2007.

[19] H. Samet and R. E. Webber. Storing a collection of polygons using quadtrees. *ACM Transactions on Graphics (TOG)*, 4(3):182–222, 1985.

[20] T. Sellis, N. Roussopoulos, and C. Faloutsos. The r+-tree: A dynamic index for multi-dimensional objects. In *Proceedings of the 13th International Conference on Very Large Data Bases*, pages 507–518. VLDB endowments, 1987.

[21] P. Vretanos. OpenGIS filter encoding 2.0 encoding standard v2. 0.0. Technical report, Open Geospatial Consortium, 2010.

[22] P. A. Vretanos. OpenGIS web feature service 2.0 interface standard. Technical report, Open Geospatial Consortium, 2010.

Appendices

Appendix A

WKT Grammar

```

"POINT"                    return POINT;
"LINESTRING"               return LINESTRING;
"POLYGON"                  return POLYGON;
"MULTIPOINT"               return MULTIPOINT;
"MULTILINESTRING"          return MULTILINESTRING;
"MULTIPOLYGON"             return MULTIPOLYGON;
"GEOMETRYCOLLECTION"       return COLLECTION;
","                        return COMMA;
"("                        return LPAR;
")"                        return RPAR;

[-+]?[0-9]*\.?[0-9]+       return NUMBERLIT;

<<EOF>>                    return 0;

[ \t]                      /* skip whitespace */
\n                         /* skip whitespace */
.                          /* error */
```

Listing A.1: Input data for Lex for WKT parsing

```

start: POINT           LPAR point_def           RPAR {geom = $3;}
     | LINESTRING      LPAR linestring_def      RPAR {geom = $3;}
     | POLYGON         LPAR polygon_def         RPAR {geom = $3;}
     | MULTIPOINT      LPAR multipoint_def      RPAR {geom = $3;}
     | MULTILINESTRING LPAR multilinestring_def RPAR {geom = $3;}
     | MULTIPOLYGON    LPAR multipolygon_def    RPAR {geom = $3;}
     | COLLECTION      LPAR collection_def      RPAR {geom = $3;}
     ;

point_def: _NUMBERLIT_ _NUMBERLIT_
         ;
```

```

linestring_def: point_def _COMMA_ point_def
              | linestring_def _COMMA_ point_def
              ;

polygon_def : LPAR linestring_def _COMMA_ point_def RPAR
            | LPAR linestring_def _COMMA_ point_def RPAR _COMMA_ interior_def
            ;

interior_def: LPAR linestring_def _COMMA_ point_def RPAR
            | interior_def _COMMA_ LPAR linestring_def _COMMA_ point_def RPAR
            ;

multipoint_def: point_def
              | LPAR point_def RPAR
              | multipoint_def _COMMA_ point_def
              | multipoint_def _COMMA_ LPAR point_def RPAR
              ;

multilinestring_def: LPAR linestring_def RPAR
                   | multilinestring_def _COMMA_ LPAR linestring_def RPAR
                   ;

multipolygon_def: LPAR polygon_def RPAR
                | multipolygon_def _COMMA_ LPAR polygon_def RPAR
                ;

atomic: POINT           LPAR point_def           RPAR {$$ = $3;}
      | LINESTRING      LPAR linestring_def      RPAR {$$ = $3;}
      | POLYGON         LPAR polygon_def         RPAR {$$ = $3;}
      | MULTIPOINT      LPAR multipoint_def      RPAR {$$ = $3;}
      | MULTILINESTRING LPAR multilinestring_def RPAR {$$ = $3;}
      | MULTIPOLYGON    LPAR multipolygon_def    RPAR {$$ = $3;}
      ;

collection_def: atomic
              | atomic _COMMA_ collection_def
              ;
```

Listing A.2: Grammar used in Yacc for WKT parsing