Embedding Vague Spatial Objects into Spatial Databases using the VagueGeometry Abstract Data Type

Anderson Chaves Carniel[1], **Ricardo Rodrigues Ciferri**[2], **Cristina Dutra de Aguiar Ciferri**[1]

[1]Department of Computer Science – University of São Paulo in São Carlos
13.560-970 – São Carlos – SP – Brazil

accarniel@gmail.com, cdac@icmc.usp.br

[2]Department of Computer Science – Federal University of São Carlos
15.565-905 – São Carlos – SP – Brazil

ricardo@dc.ufscar.br

Abstract. *Spatial vagueness has been required by geoscientists to handle vague spatial objects, i.e., spatial objects that do not have exact locations, strict boundaries, or sharp interiors. However, there is a gap in the literature in how to handle these objects in spatial database management systems since they mainly provide support to crisp spatial objects, i.e., objects that have well-defined locations, boundaries, and interiors. In this paper, we fill this gap by proposing VagueGeometry, a novel abstract data type that handles vague spatial objects, includes an expressive set of vague spatial operations, and its implementation is open source. Experimental results show that VagueGeometry improved the performance of spatial queries with vague topological predicates from 23% up to 84% if compared with functionalities available in current spatial databases.*

1. Introduction

Spatial database management systems (spatial DBMS) and Geographical Information Systems (GIS) mainly provide support to handle *crisp spatial objects* to represent real-world phenomena by using points, lines, and regions. Crisp spatial objects characterize spatial phenomena with exact locations and whose shape and boundary are precisely defined [Schneider and Behr 2006]. Examples are cities with their political boundaries. For their handling, spatial operations like geometric set operations (e.g., union), topological predicates (e.g., overlap), and numerical operations (e.g., distance) are defined and used in spatial queries.

However, geoscientists are increasingly interested in modeling spatial real-world phenomena that do not have exact locations, strict boundaries, or sharp interiors. This characterization leads to *spatial vagueness*. In general, it has a spatial extent that *certainly* belongs to the real-world phenomena (i.e., the *kernel*) and a spatial extent that *maybe* belongs to the real-world phenomena (i.e., the *conjecture*) [Siqueira et al. 2014].

There are a number of approaches proposed in the literature that define models to represent spatial vagueness, which can be classified as *probabilistic models* [Li et al. 2007, Zinn et al. 2007], *fuzzy models* [Kraipeerapun 2004, Dilo et al. 2007, Dilo et al. 2006, Carniel et al. 2014], and *exact models* [Clementini and Di Felice 1997,

Pauly and Schneider 2008, Pauly and Schneider 2010, Bejaoui et al. 2010]. These models introduce concepts and notions of *vague spatial objects* by formally defining spatial data types for *vague points*, *vague lines*, and *vague regions*. They also introduce vague spatial operations to handle them, i.e., *vague geometric set operations* (e.g., vague geometric union), *vague topological predicates* (e.g., vague overlap), and *vague numerical operations* (e.g., vague distance).

There are several advantages of incorporating vague spatial objects and their operations into spatial queries, such as to provide a more realistic representation of application environments, to allow users to manipulate vague spatial objects found in real-world phenomena, and to provide an efficient processing of operations on vague spatial objects. For instance, in an ecological application, users aim to manage habitats of species and polluted areas of rivers. The habitats and the polluted areas of rivers are represented by vague regions. This means that habitats have locations where species certainly appear and locations where species maybe appear. Further, rivers have unpolluted areas and areas where there is some kind of pollution. By using such data, a user can ask the following query: "Find all polluted areas of rivers that *maybe overlap* with habitats of species". In this query, spatial vagueness is represented by the *maybe overlap* predicate, which will return true when the overlap occurs to some extent, i.e., occurs with some degree of uncertainty.

To handle spatial objects in spatial applications, *abstract data types* (ADT) have been used in spatial DBMS and GIS. An ADT for spatial data types aids the use of spatial operations in spatial queries by hiding their complexities from the user. While ADTs for crisp spatial data are deeply implemented in the literature, this is not the case for vague spatial data. Although there are approaches that provide ADTs for vague spatial data [Dilo et al. 2006, Kraipeerapun 2004, Pauly and Schneider 2008, Pauly and Schneider 2010, Zinn et al. 2007], they face several limitations. First, they only provide support for a small subset of vague spatial operations. Second, they do not support textual and binary representations of vague spatial objects. Finally, they do not support SQL operators to manipulate results of vague spatial operations.

In this paper, we fill this gap in the literature. We propose a novel ADT named VagueGeometry to incorporate vague spatial objects into a spatial DBMS. VagueGeometry is based on the *exact model* since this model reuses existing concepts and implementations of crisp spatial data and formally defines a complete set of vague spatial operations. The main advantage to use implementation of crisp spatial data is that they are well defined and their efficiency is largely explored in the literature. Among the exact models [Bejaoui et al. 2010, Pauly and Schneider 2008, Pauly and Schneider 2010, Clementini and Di Felice 1997], VagueGeometry is based on the *Vague Spatial Algebra* (VASA) [Pauly and Schneider 2008, Pauly and Schneider 2010] since it formally defines simple and complex vague spatial data types. Further, VASA introduces a more expressive algebra than the models described in [Bejaoui et al. 2010] and [Clementini and Di Felice 1997] by including a huge set of operations, such as vague geometric set operations, vague topological predicates, and vague numerical operations.

VagueGeometry greatly overcomes the aforementioned limitations. Other major characteristics of the proposed VagueGeometry are described as follows.

- It offers textual and binary representations for vague spatial objects, which make possible their use to define, insert, and retrieve vague spatial objects by using

textual or binary representations. Further, these representations can be used as a way of communication and interoperability between different spatial applications.
- It implements an expressive set of spatial operations for vague spatial objects. To comply with this goal, VagueGeometry includes the specification of vague geometric set operations, vague topological predicates, and vague numerical operations. As a result, the use of VagueGeometry empowers the management of vague spatial objects in spatial applications by users.
- It supports SQL operators that allow users to handle results of vague topological predicates and vague numerical operations.
- It is open source and implemented in the open source PostgreSQL DBMS with the PostGIS spatial extension. This means that spatial applications are able to access directly a spatial DBMS containing vague spatial objects and handle these objects by using vague spatial operations accordingly.
- It includes an efficient improvement to process vague topological predicates in spatial queries.

This paper is organized as follows. Section 2 surveys related work. Section 3 summarizes the Vague Spatial Algebra. Section 4 introduces the proposed VagueGeometry. Section 5 details the improvement in the processing of vague topological predicates. Section 6 describes performance tests. Section 7 concludes the paper.

2. Related Work

There are few approaches that implement ADTs for handling vague spatial objects in spatial DBMS and GIS [Dilo et al. 2006, Kraipeerapun 2004, Pauly and Schneider 2008, Pauly and Schneider 2010, Zinn et al. 2007]. These approaches mainly differ from our proposed VagueGeometry in the practicable applicability of the user to handle vague spatial objects in spatial queries. We compare these approaches with VagueGeometry by considering the following functionalities related to the support provided by them: (i) textual representation, (ii) binary representation, (iii) vague geometric set operations, (iv) vague topological predicates, (v) vague numerical operations, (vi) SQL operators, and (vii) coupling with a spatial DBMS.

The majority of the approaches does not provide textual and binary representations (functionalities (i) and (ii)) to allow the user to insert and retrieve vague spatial objects. While in [Zinn et al. 2007] is provided input and output functions for vague spatial objects by using binary format, this is not the case for the textual representation. The approaches described in [Pauly and Schneider 2008] and [Pauly and Schneider 2010] define vague spatial objects by using extensive terminal command lines without any textual or binary representations. The approaches described in [Kraipeerapun 2004] and [Dilo et al. 2006] support options to represent vague spatial objects by using files in the format of the GRASS GIS. However, the binary and textual formats are not understandable for users and depend on a specific system, thus limiting interoperability between spatial applications. On the other hand, VagueGeometry defines textual and binary representations for vague spatial objects.

Regarding functionality (iii), the approaches described in [Zinn et al. 2007], [Pauly and Schneider 2008], and [Pauly and Schneider 2010] implement vague geometric union, intersection, and difference between vague points, lines, and regions.

The approaches described in [Kraipeerapun 2004] and [Dilo et al. 2006] do not implement the vague geometric difference between vague lines. Regarding functionalities (iv) and (v), some approaches described in [Pauly and Schneider 2008] and [Pauly and Schneider 2010] provide support to vague topological predicates and vague numerical operations, while other approaches in [Zinn et al. 2007], [Kraipeerapun 2004], and [Dilo et al. 2006] do not provide support for these operations, and therefore, limit the management of vague spatial objects and the type of queries that can be processed. Our proposed VagueGeometry implements an expressive set of spatial operations for vague spatial objects, which includes the specification of vague geometric set operations, vague topological predicates, and vague numerical operations.

Furthermore, there is no related work that support SQL operators to handle results of vague spatial operations (functionality (vi)). Although the approaches described in [Pauly and Schneider 2008] and [Pauly and Schneider 2010] propose some operators, they do not implement them. However, offering SQL operators is an important functionality since it allows users to intuitively handle results of spatial operations in SQL queries. Therefore, differently from related work, VagueGeometry supports SQL operators for vague spatial operations.

Finally, regarding functionality (vii), the approaches described in [Kraipeerapun 2004] and [Dilo et al. 2006] do not implement vague spatial objects in a spatial DBMS, while the approaches described in [Pauly and Schneider 2008], [Pauly and Schneider 2010], and [Zinn et al. 2007] offer this functionality. However, the implementation of [Pauly and Schneider 2008] and [Pauly and Schneider 2010][1] is based on the Oracle, which has license restrictions. On the other hand, VagueGeometry is an open source implementation based on the PostgreSQL.

3. Vague Spatial Algebra

A vague spatial object in the Vague Spatial Algebra (VASA) [Pauly and Schneider 2008, Pauly and Schneider 2010] is defined as a pair of crisp spatial objects of the same spatial data type, which must be disjoint or adjacent. The first object represents the *kernel* part, while the second object represents the *conjecture* part. In addition, VASA is characterized to separate the portions of space of a spatial object by considering minimum and maximum approximations. As a result, a spatial object can comprise or expand according to a minimum limit (i.e., the kernel part) and a maximum limit (i.e., the conjecture part). Formally, let $\alpha \in \{\textit{crisp point}, \textit{crisp line}, \textit{crisp region}\}$. Then, a vague spatial data type in VASA is defined by $v(\alpha) = \alpha \times \alpha$, such that for an object $o = (o_k, o_c) \in v(\alpha)$, the property $disjoint(o_k, o_c) \vee meet(o_k, o_c)$ holds. The kernel and conjecture of o are symbolized by o_k and o_c, respectively.

VASA defines the following operations to handle vague spatial objects: vague geometric set operations, vague topological predicates, and vague numerical operations. Vague geometric set operations are defined by reusing the crisp geometric set operations.Vague topological predicates are based on the three-valued logic, and thus, can return *true*, *false*, or *maybe*. Let A and B be two vague spatial objects. A vague topological predicate is evaluated by using the well-known crisp 9-intersection matrix [Schneider and Behr 2006] for the following combinations $A_k \times B_k$, $A_k \times (B_c \oplus B_k)$,

[1] http://www.cise.ufl.edu/research/SpaceTimeUncertainty/

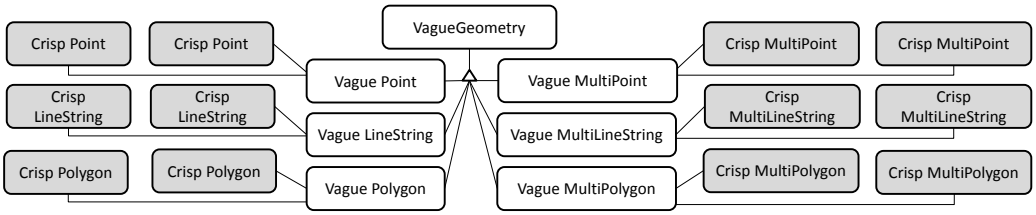


Figure 1. The vague spatial data types of VagueGeometry.

$(A_k \oplus A_c) \times B_k$, and $(A_k \oplus A_c) \times (B_c \oplus B_k)$, where $\oplus$ denotes the crisp geometric union. Finally, vague numerical operations return a pair of numeric values corresponding to a minimum and a maximum value. For instance, the area of a vague region object has a minimum value that corresponds to the area of its kernel and a maximum value that corresponds to the area of the union between its kernel and its conjecture. Details of the formal definitions for vague spatial operations of VASA are given in [Pauly and Schneider 2010].

4. The VagueGeometry Abstract Data Type

In this section, we propose VagueGeometry, an ADT to handle vague spatial objects in a spatial DBMS. VagueGeometry was implemented by using the C language and the extensibility provided by the PostgreSQL internal library. It is based on VASA, and thus we make use of the spatial operations provided by PostGIS and GEOS to implement the vague spatial data types and their operations. GEOS[2] is a C/C++ library that implements crisp spatial data types and their crisp spatial operations according to the OGC specifications[3]. A detailed documentation of VagueGeometry is available at `http://gbd.dc.ufscar.br/vaguegeometry/`.

4.1. Representation of Vague Spatial Objects

Figure 1 depicts vague spatial data types of VagueGeometry which can be simple or complex. Simple vague spatial data types named *vague point*, *vague linestring*, and *vague polygon* denote simple vague points, simple vague lines, and simple vague regions, respectively. Complex vague spatial data types named *vague multipoint*, *vague multilinestring*, and *vague multipolygon* denote complex vague points, complex vague lines, and complex vague regions, respectively. We employ this notation to follow the same notation used by the OGC specification to denote crisp spatial objects. Note that a vague spatial object of VagueGeometry is composed of a pair of disjoint or adjacent crisp spatial objects of the same spatial data type, which is showed in gray in Figure 1.

To use VagueGeometry in spatial queries, we propose textual and binary representations for vague spatial objects. We present our proposed representations by first detailing the *textual representations*. They are: (i) *Vague Well-Known Text* (VWKT), (ii) *Vague Geography Markup Language* (VGML), (iii) *Vague Keyhole Markup Language* (VKML), and (iv) *Vague Geographic JavaScript Object Notation* (vGeoJSON). These textual representations are based on the OGC specifications that use the following textual

[2]http://trac.osgeo.org/geos/
[3]http://www.opengeospatial.org/

representations for crisp spatial objects: Well-Known Text (WKT), Geographic Markup Language (GML), Keyhole Markup Language (KML), and Geographic JavaScript Object Notation (GeoJSON).

The VWKT, VGML, VKML, and vGeoJSON representations are defined as follows. Let A be a VagueGeometry object, which can assume different data types (Figure 1), formed by the kernel A_k and the conjecture A_c. Let $name$ be a function that returns a keyword representing the data type of A. For instance, $name(A)$ returns the keyword `VAGUEPOINT` if A is a simple vague point object. Finally, let ***WKT***, ***GML***, ***KML***, and ***GeoJSON*** be functions that get a crisp spatial object as input and return its respective textual representation. The textual representations for a VagueGeometry object A are:

(i) $VWKT(A) = name(A)(\ WKT(A_k);\ WKT(A_c))$
(ii) $VGML(A) =$ <vgml:$name(A)$><vgml:Kernel>$GML(A_k)$</vgml:Kernel>
<vgml:Conjecture>$GML(A_c)$</vgml:Conjecture></vgml:$name(A)$>
(iii) $VKML(A) =$ <vkml:$name(A)$><vkml:Kernel>$KML(A_k)$</vkml:Kernel>
<vkml:Conjecture>$KML(A_c)$</vkml:Conjecture></vkml:$name(A)$>
(iv) $vGeoJSON(A) =$ {"type": "$name(A)$", "kernel": $GeoJSON(A_k)$,
"conjecture": $GeoJSON(A_c)$}

We now move our discussion to the proposed *binary representation*, called *Vague Well-Known Binary* (VWKB). It is based on the *Well-Known Binary* (WKB) representation for crisp spatial objects documented in the OGC specification. Our VWKB representation is defined as follows. Let id be a function that returns an integer in the binary format symbolizing the data type of A. For instance, $id(A)$ returns 1, in the binary format, if A is a simple vague point object. Let ***WKB*** be a function that gets a crisp spatial object as input and returns its respective WKB representation. Let $endianess$ be an flag that indicates the way in which the bytes are organized in main memory (i.e., either *big-endian* or *little-endian*). The VWKB representation for a VagueGeometry object A is:

$$VWKB(A) = endianess + id(A) + WKB(A_k) + WKB(A_c),$$

where $+$ denotes the union between the serialized data.

VagueGeometry supports textual and binary representations to allow its use in different spatial applications. Hence, spatial applications based on XML or web services that use XML as communication are able to use the VGML and VKML representations. Web applications that utilize JavaScript as main language are able to use the vGeoJSON representation. Applications that manage binary files are able to use the VWKB representation. Finally, for general purpose, applications can make use of the VWKT representation. It is important to note that these representations also provide interoperability between applications since a vague spatial object has an unique representation.

4.2. Vague Spatial Operations

VagueGeometry provides support to *input and output operations*, *vague geometric set operations*, *vague topological predicates*, and *vague numerical operations*. While *input operations* transform textual or binary representations into a VagueGeometry object, *output operations* transform a VagueGeometry object into a textual or binary representation.

Vague geometric set operations get two VagueGeometry objects as input and yield another VagueGeometry object. VagueGeometry implements *vague geometric union*, *vague geometric intersection*, and *vague geometric difference*.

Regarding vague topological predicates, VagueGeometry supports *vague coveredBy*, *vague covers*, *vague crosses*, *vague disjoint*, *vague equals*, *vague inside*, *vague intersects*, *vague meets*, and *vague overlap*. These predicates are based on the three-valued logic, and can return *true*, *false*, or *maybe*. A predicate returns *true* if a relationship *certainly* occurs, *false* if a relationship *certainly not* occurs, and *maybe* if a relationship *occurs to some extent*. To deal with it, VagueGeometry also includes the VagueBool data type. As a result, a VagueBool object can assume *true*, *false*, or *maybe* as value, which correspond to the possible return values of vague topological predicates. In addition, it is possible to use crisp spatial objects as input, which is handled as a vague spatial object containing only the kernel part.

Finally, vague numerical operations supported by VagueGeometry are: *vague area of a vague region object*, *vague length of a vague line object*, and *farthest* and *nearest distance between two vague spatial objects*. These operations return two numeric values, which symbolize the minimum and the maximum values of an operation. To deal with it, VagueGeometry also includes the VagueNumeric data type. As a result, a VagueNumeric object is composed of a pair of double values, which correspond to the minimum and the maximum values returned by vague numerical operations.

As can be noted, our proposed VagueGeometry implements an expressive set of vague spatial operations, which includes the specification of vague geometric set operations, vague topological predicates, and vague numerical operations.

4.3. SQL Operators

We propose SQL operators to handle VagueBool and VagueNumeric objects, i.e., the result of vague topological predicates and vague numerical operations, respectively. For the vague topological predicates, we propose the *logical operators and* (&&), *or* (||), and *not* (!), and the *boolean operators* $\sim$, $\sim\sim$, and $\&$. Logical operators employ the three-valued logic. They get VagueBool objects as input and yield another VagueBool object. The logical operators && and || are binary operators, while ! is a unary operator. For instance, users can use these operators to specify the condition "*VG_Meets*(*vg1*, *vg2*) || *VG_Overlap*(*vg1*, *vg2*)" in a SQL query to indicate that two VagueGeometry objects $vg1$ and $vg2$ meet *or* overlap. On the other hand, *boolean operators* are unary operators that get a VagueBool object as input and have true or false as possible return values. Therefore, a boolean operator transforms a vague topological predicate into a boolean restriction. The operator $\sim$ yields true if the VagueBool object is *true* or *maybe*, and false otherwise. The operator $\sim\sim$ yields true if the VagueBool object is *maybe*, and false otherwise. The operator & yields true if the VagueBool object is *true*, and false otherwise. For instance, users can evaluate if two VagueGeometry objects *maybe* overlap by specifying the condition "$\sim\sim$*VG_Overlap*(*vg1*, *vg2*)" in a SQL query.

Regarding the vague numerical operations, we propose the binary operators $=$ and $\sim$, which get a VagueNumeric object and a numeric value as input and yield true or false. The operator $=$ yields true if the numeric value is equal to the minimum value of the VagueNumeric object, and false otherwise. The operator $\sim$ yields true if the numeric value is between the minimum and the maximum value of the VagueNumeric object, and false otherwise. For instance, users can use this operator to specify the condition "*VG_Area*(r) $\sim$ 800" in a SQL query to restrict vague region objects in the attribute r that have approximately 800 of area.

5. Efficient Processing of Vague Topological Predicates

The implementation of VagueGeometry includes an improvement for the processing of vague topological predicates. The proposed improvement, called *MBRs for Vague Topological Predicates* (*MBRVP*), makes use of *Minimum Boundary Rectangles* (MBR) of the kernel and conjecture parts of vague spatial objects to return the results of vague topological predicates when *some situations hold*. In these situations, MBRVP can avoid the use of crisp 9-intersection matrices to evaluate the vague topological predicate. As a result, the time to process spatial queries can be reduced.

We consider two situations, named *disjointness between MBRs* and *set containment between MBRs*. The disjointness between MBRs encompasses two specific cases, as depicted in Figure 2a. The first case occurs if the MBRs of the union between the kernel and the conjecture of two vague spatial objects are disjoint. The second case occurs if the MBRs of the kernel and the conjecture of two vague spatial objects are disjoint. Note that the second case can happen even when the first case holds.

Formally, let A and B be two vague spatial objects. Let also MBR_o be a MBR of a crisp spatial object o. The *disjointness between MBRs* $S_d(A, B)$ yields true if $((MBR_{A_k} \cup MBR_{A_c}) \cap (MBR_{B_k} \cup MBR_{B_c}) = \varnothing) \vee (MBR_{A_k} \cap MBR_{B_k} = \varnothing \wedge MBR_{A_k} \cap MBR_{B_c} = \varnothing \wedge MBR_{A_c} \cap MBR_{B_k} = \varnothing \wedge MBR_{A_c} \cap MBR_{B_c} = \varnothing)$, and false otherwise. By using this definition, we are able to return *true* for the vague disjoint predicate if $S_d(A, B) = true$ holds, and return *false* for the predicates of vague meets, vague intersects, vague overlap, and vague equals if $S_d(A, B) = false$ holds. Otherwise, the respective predicate is evaluated with the computation of crisp 9-intersection matrices. It includes the case of an intersection between MBRs of the conjecture parts since we cannot guarantee that the conjecture parts really intersects due to the dead space of the MBRs.

Regarding the set containment between MBRs, it also encompasses two specific cases (Figure 2b). The first case occurs if the MBR of the kernel of the first vague spatial object is not inside the MBR of the union between the kernel and the conjecture of the second vague spatial object. Hence, the interior of the kernel of the first vague spatial object intersects the exterior of the second vague spatial object. The second case occurs if the MBR of the kernel of the first vague spatial object and the MBRs of the kernel and the conjecture of the second vague spatial object are disjoint.

Formally, let A and B be two vague spatial objects. Let also MBR_o be a MBR of a crisp spatial object o. The *set containment between MBRs* $S_{sc}(A, B)$ yields true if $(MBR_{A_k} \nsubseteq (MBR_{B_k} \cup MBR_{B_c})) \vee (MBR_{A_k} \cap MBR_{B_k} = \varnothing \wedge MBR_{A_k} \cap MBR_{B_c} = \varnothing)$, and false otherwise. By using this definition, we are able to return *false* for the predicates of vague inside and vague coveredBy if $S_{sc}(A, B) = true$ holds. Otherwise, the respective predicate is evaluated. Similarly, if $S_{sc}(B, A) = true$ holds, then we can return *false* for the predicates of vague contains and vague covers, and evaluate the respective predicates otherwise.

6. Performance Evaluation

The advantages of our proposed solutions (i.e., the VagueGeometry ADT and the MBRVP improvement) were analyzed through experimental tests that process spatial queries with vague topological predicates. We analyze topological predicates since they incur high costs of processing and they are very common in spatial applications.

(a) (b)

Figure 2. Examples of the situations where the MBRVP improvement should be applied. Figure 2a: disjointness between MBRs. Figure 2b: set containment between MBRs. Black regions represent the kernel, while gray regions represent the conjecture.

6.1. Experimental Setup

We used a data set composed of 100,000 vague regions synthetically generated as follows. First, we constructed a Voronoi diagram of 200,000 crisp points randomly generated, which produced the same number of crisp regions. Second, for each crisp region, we added more points to increase its complexity. As a result, each crisp region was formed by averagely 313 points. Third, we created pairs of crisp regions that were disjoint or adjacent. Each pair was created by selecting randomly a crisp region and then by selecting the nearest crisp region that was disjoint or adjacent to the first one. We randomly assigned a crisp region as the kernel and the other crisp region as the conjecture. After creating a pair, we discarded the used regions, such that these regions were not used to create another pair. In the total, we generated 100,000 pairs of crisp regions.

We considered the following topological predicates: *vague disjoint*, *vague overlap*, *vague inside*, *vague intersects*, *vague coveredBy*, and *vague meets*. The workload was composed of 100 spatial range queries for each vague topological predicate. We also defined a query window for each spatial range query, which was composed of a vague region object that had the rectangular format for the kernel and the conjecture. Therefore, we randomly generated 100 different query windows.

We defined the following configurations: (i) *baseline* that used current functionalities provided by the PostgreSQL with the PostGIS spatial extension; (ii) *VagueGeometry* that used the proposed VagueGeometry ADT without improvements; and (iii) *VagueGeometry+* that used VagueGeometry empowered with the proposed MBRVP improvement. For the *baseline* configuration, we implemented vague topological predicates by using the Procedural Language/PostgreSQL (PL/pgSQL), which had "TRUE", "FALSE", or "MAYBE" as possible return textual values. For their use, we stored the kernel and the conjecture of each vague spatial object in separated columns in a relational table.

Note that we did not employ the approaches surveyed in Section 2 in the performance comparisons due to the following limitations. While the approaches proposed in [Zinn et al. 2007], [Kraipeerapun 2004], and [Dilo et al. 2006] *do not provide support for vague topological predicates*, the approaches described in [Pauly and Schneider 2008] and [Pauly and Schneider 2010] are *specifically implemented in Oracle*, which has license restrictions. Further, we used PostgreSQL in the performance tests to isolate the effects of the DBMS, and thus providing a fair comparison.

Table 1 depicts the SQL templates of the spatial range queries used in the three configurations. Consider the template for the *baseline* configuration. *baselineTable* is a

Table 1. Templates of the SQL spatial range queries.

Configuration	SQL Template
baseline	SELECT id FROM baselineTable WHERE R = P(kernel_geo, conjecture_geo, QW_k, QW_c)
VagueGeometry *VagueGeometry+*	SELECT id FROM vaguegeom WHERE O P(vg, QW)

table composed of three attributes: (i) *id* that is the primary key, (ii) *kernel_geo* that represents the kernel of the vague region objects, and (iii) *conjecture_geo* that represents the conjecture of a vague region object. Further, R is a textual return value that may contain "TRUE", "FALSE", or "MAYBE", P is the vague topological predicate, and QW is the query window. Regarding the *VagueGeometry* and the *VagueGeometry+* configurations, *vaguegeom* is a table that stored vague region objects in the attribute *vg* by using our proposed VagueGeometry. In addition, O corresponds to the use of the SQL operators introduced in Section 4.3. This means that the operator $\sim\sim$ was used to specify that P returned maybe, the operator $\&$ was used to specify that P returned true, and the combination of the operator $\&$ with the operator ! (i.e., $\&!$) was used to specify that P returned false. Note that the SQL templates are equivalent, i.e., they generate the same result, but using the specific functionalities provided by the corresponding configurations.

The experiments were conducted on a computer with an Intel® Core™ i7-4770 processor with frequency of 3.40GHz, 2 TB SATA hard drive with 7200 RPM, and 32 GB of main memory. The operating system was CentOS 6.5 with Kernel Version 2.6.32-431.el6.x86 64. We employed PostgreSQL 9.3.3, PostGIS 2.2.0, and GEOS 3.4.2.

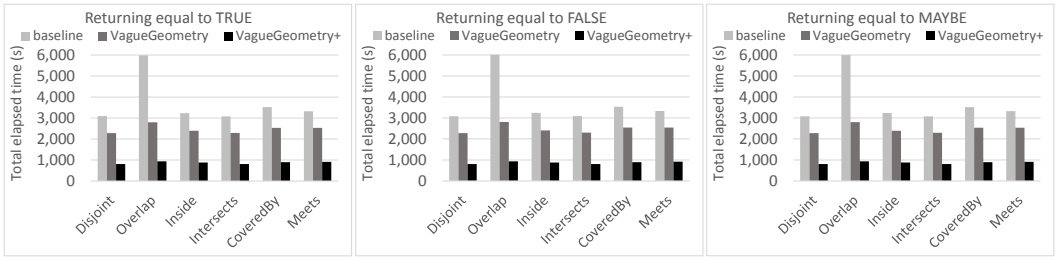
We collected the elapsed time in seconds. In detail, we executed 100 spatial range queries for each vague topological predicate and each value of return. Further, we executed each spatial range query 10 times and calculated its average elapsed time. Furthermore, we performed the tests locally to avoid network latency and we flushed the system cache after the execution of each query.

6.2. Performance Results

Figure 3 depicts the performance results. For each configuration and each return value of the three-valued logic (i.e., true, false, and maybe), we gathered similar elapsed times for processing the spatial queries. This means that the performance results showed a similar complexity for each return value.

Clearly, the performance of the *VagueGeometry* configuration overcame the *baseline* configuration. In fact, the internal structures of the VagueGeometry were more efficient than the definition of vague topological predicates by using current functionalities of the spatial DBMS. The performance gain imposed by the *VagueGeometry* configuration over the *baseline* configuration ranged from 23% up to 53%, where the performance gain is calculated as the percentage that determines how much more efficient one configuration was than another configuration.

Despite the expressive performance gains obtained by the *VagueGeometry* config-



(a) (b) (c)

Figure 3. Performance results for each vague topological predicate considering the returning values of true (a), false (b), and maybe (c).

uration, we gathered yet better results with the improvement proposed in Section 5. The *VagueGeometry+* configuration leaded to a performance gain that ranged from 72% up to 84% if compared with the *baseline* configuration. Further, compared to the *VagueGeometry* configuration, the *VagueGeometry+* configuration provided a performance gain that ranged from 63% up to 66%. This means that the use of the MBRVP improvement drastically reduced the necessity of processing crisp 9-intersection matrices in vague topological predicates.

Regarding storage space, the *baseline* configuration required 961 MB, the *VagueGeometry* configuration required 960 MB, and the *VagueGeometry+* configuration required 963 MB. We can conclude that the storage cost were almost the same. In addition, the storage of the MBRs of the kernel and the conjecture of each VagueGeometry object in the *VagueGeometry+* configuration did not introduce overhead in the execution of the spatial queries.

7. Conclusions and Future Work

In this paper, we proposed VagueGeometry, a novel abstract data type to handle vague spatial objects in the PostgreSQL with the PostGIS spatial extension. VagueGeometry empowers the management of spatial applications by offering textual and binary representations for vague spatial objects and by providing an expressive set of spatial operations, including vague geometric set operations, vague topological predicates, and vague numerical operations. As facilities, VagueGeometry introduces SQL operators for manipulating results of vague topological predicates and vague numerical operations. We also introduced MBRVP, an improvement to VagueGeometry to speed up the performance of spatial queries to process vague topological predicates.

Comparisons of VagueGeometry with current functionalities available on PostgreSQL showed that VagueGeometry provided better performance results for spatial queries with vague topological predicates. The performance gain of VagueGeometry varied from 23% up to 53%. Empowered with MBRVP, VagueGeometry provided even better results, which varied from 72% up to 84%.

Future work will deal with the extension of VagueGeometry to allow the use of index structures. Another future work refers to the development of specialized spatial join algorithms for vague spatial objects by using index structures.

Acknowledgments

The authors have been supported by the Brazilian research agencies FAPESP, CAPES, and CNPq.

References

Bejaoui, L., Pinet, F., Schneider, M., and Bédard, Y. (2010). OCL for formal modelling of topological constraints involving regions with broad boundaries. *GeoInformatica*, 14(3):353–378.

Carniel, A. C., Schneider, M., Ciferri, R. R., and Ciferri, C. D. A. (2014). Modeling fuzzy topological predicates for fuzzy regions. In *Proceedings of the ACM International Symposium on Advances in Geographic Information Systems*, pages 529–532, New York, NY, USA.

Clementini, E. and Di Felice, P. (1997). Approximate topological relations. *International Journal of Approximate Reasoning*, 16(2):173–204.

Dilo, A., Bos, P., Kraipeerapun, P., and de By, R. A. (2006). Storage and manipulation of vague spatial objects using existing GIS functionality. In Bordogna, G. and Psaila, G., editors, *Flexible Databases Supporting Imprecision and Uncertainty*, volume 203, pages 293–321. Springer Berlin Heidelberg.

Dilo, A., de By, R. A., and Stein, A. (2007). A system of types and operators for handling vague spatial objects. *International Journal of Geographical Information Science*, 21(4):397–426.

Kraipeerapun, P. (2004). Implementation of vague spatial objects. Master's thesis, International Institute for Geo-Information Science and Earth Observation.

Li, R., Bhanu, B., Ravishankar, C., Kurth, M., and Ni, J. (2007). Uncertain spatial data handling: Modeling, indexing and query. *Computers & Geosciences*, 33(1):42–61.

Pauly, A. and Schneider, M. (2008). *Quality Aspects in Spatial Data Mining*, chapter Querying vague spatial objects in databases with VASA, pages 3–14. CRC Press, USA.

Pauly, A. and Schneider, M. (2010). VASA: An algebra for vague spatial data in databases. *Information Systems*, 35(1):111–138.

Schneider, M. and Behr, T. (2006). Topological relationships between complex spatial objects. *ACM Transactions on Database Systems*, 31(1):39–81.

Siqueira, T. L., Ciferri, C. D. A., Times, V. C., and Ciferri, R. R. (2014). Modeling vague spatial data warehouses using the VSCube conceptual model. *Geoinformatica*, 18(2):313–356.

Zinn, D., Bosch, J., and Gertz, M. (2007). Modeling and querying vague spatial objects using shapelets. In *Proceedings of the International Conference on Very Large Data Bases*, pages 567–578, Vienna, Austria.