Handling Fuzzy Points and Fuzzy Lines using the FuzzyGeometry Abstract Data Type

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

[1] University of São Paulo, Brazil
accarniel@gmail.com, cdac@icmc.usp.br
[2] Federal University of São Carlos, Brazil
ricardo@dc.ufscar.br

Abstract. *Crisp spatial objects* are geometric features with exact location on the extent and well-known boundaries. On the other hand, *vague* or *fuzzy spatial objects* are characterized by uncertain or blurred boundaries and interiors. Despite the importance of fuzzy spatial data in spatial applications, few related work indeed implement them. In addition, related work do not define abstract data types to enable the management of fuzzy spatial objects by using database management systems (DBMS). In this article, we propose the *abstract data type* FuzzyGeometry to handle fuzzy spatial objects in the PostgreSQL DBMS. Its implementation is open source. It offers management for fuzzy point objects and fuzzy line objects as well as provides several operations to handle them. As a result, users are able to access PostgreSQL in order to use fuzzy spatial objects in spatial queries.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Spatial databases and GIS

Keywords: abstract data types, fuzzy spatial objects, spatial databases, spatial fuzziness

1. INTRODUCTION

Spatial applications are commonly used for spatial analysis in order to aid in the decision making process. Real-world phenomena are commonly represented by *crisp spatial objects* [Güting 1994; Schneider and Behr 2006], such as crisp points, crisp lines, and crisp regions whose can assume simple or complex structures. Crisp spatial objects have exact location in the extent, that is, their geographical coordinates clearly define their geographical positions. In addition, crisp spatial objects have well-defined boundaries that determine with exactness their limit. In order to handle crisp spatial objects in spatial Database Management Systems (spatial DBMS) and Geographical Information System (GIS), spatial operations are defined, such as crisp geometric operations (for instance, union and intersection) and crisp numerical operations (for instance, area and distance calculations).

On the other hand, real-world phenomena frequently have inexact location, uncertain boundaries, or blurred interiors [Siqueira et al. 2014]. For instance, the representation of soil mapping, fire and risk zones, oceans, lakes, and air polluted areas. In order to represent these phenomena, we use *vague spatial data types*. There are several models to represent vague spatial objects, such as *exact models* [Cohn and Gotts 1996; Pauly and Schneider 2008; Bejaoui et al. 2009; Pauly and Schneider 2010], *rough models* [Beaubouef et al. 2004], *probabilistic models* [Cheng et al. 2003; Li et al. 2007; Zinn et al. 2007], and *fuzzy models* [Dilo et al. 2007; Schneider 2008; 2014; Carniel et al. 2014]. However, the majority of existing implementations of vague spatial objects [Kraipeerapun 2004; Dilo et al. 2006; Pauly and Schneider 2008; 2010; Carniel et al. 2015b] are based on the exact models and fuzzy models due to the following factors: (i) exact models reuses existing crisp spatial operations from conventional spatial DBMS and GIS to handle vague spatial objects and (ii) fuzzy models are based on the fuzzy set theory, which allows the definition of vague spatial objects by using membership functions.

Exact models have the advantage of the use of well-known and efficient crisp spatial operations that are largely explored in the literature. As a result, implementations based on these models have an expressive set of vague spatial operations, such as discussed by Carniel et al. [2015b]. However, these implementations are not able to represent in detail a real-world phenomenon since a crisp spatial object is used to denote the well-known part and another crisp spatial object is used to denote the vague part of a vague spatial object. For instance, a sandy soil mapping is represented by two crisp region objects. One object indicates where the soil is *certainly* sandy (that is, the well-known part) and another object indicates where the soil is *possibly* sandy (that is, the vague part).

On the other hand, fuzzy models allows us to represent a vague spatial object in more details. This is done by assigning a membership degree between 0 and 1 for each point of the object. Membership degrees specify how much a point belongs to a vague spatial object. Using our previous sandy soil mapping example, we are able to assign *different membership degrees to different locations*, specifying to which extent each point belongs to the sandy soil due to different mineral compositions. Therefore, differently from exact models, we are able to represent in detail a sandy soil mapping. Vague spatial data based on the fuzzy models are defined as *fuzzy spatial data types*, such as *fuzzy points*, *fuzzy lines*, and *fuzzy regions*.

Despite the advantage to use fuzzy models, only few implementations are based on them [Kraipeerapun 2004; Dilo et al. 2006]. In addition, they face several limitations, such as: (i) there are no textual or binary representations specifically designed to handle fuzzy spatial objects, (ii) only a small set of operations are defined and implemented, and (iii) its implementation is based on a GIS and not in a spatial DBMS, which can limit the management of fuzzy spatial objects in spatial applications.

In this article, we propose an *abstract data type* (ADT) named FuzzyGeometry. The FuzzyGeometry ADT is based on the fuzzy model and is implemented as an open-source PostgreSQL extension. The main goal of this article is to serve as a specification of implementation of an ADT that handle fuzzy spatial objects. We conduct this specification by introducing the proposed FuzzyGeometry ADT. Here, we focus on the design and implementation of fuzzy points and fuzzy lines only, which already have several challenges for their definitions and implementations. To handle them, we define *input and output functions*, *fuzzy geometric set operations*, and *fuzzy spatial operations based on the fuzzy set theory*. We also extend our previous work [Carniel et al. 2015a] with the following contents: (i) we detail the specification of the implementation and serialization of fuzzy lines and fuzzy points, (ii) we define textual and binary representations to manage fuzzy lines and fuzzy points, (iii) we exemplify several fuzzy spatial operations by using graphical representations and textual representations, and (iv) we introduce a running example to demonstrate the functionalities of the FuzzyGeometry by using SQL queries.

This article is organized as follows. Section 2 summarizes related work. Section 3 describes the technical background. Section 4 presents our FuzzyGeomety ADT. Section 5 introduces the running example in order to show how to use the FuzzyGeometry in SQL queries. Finally, Section 6 concludes the article and presents future work.

2. RELATED WORK

Few works in the literature implement vague spatial data types based on the fuzzy model by using a spatial DBMS or GIS. On the other hand, exact models are also frequently adopted to represent vague spatial objects since they use well-known crisp spatial algorithms. Hence, we compare implementations based on the exact model and the fuzzy model.

Although several exact models have been proposed [Cohn and Gotts 1996; Pauly and Schneider 2008; Bejaoui et al. 2009; Pauly and Schneider 2010], only few implementations are based on them. Vague Spatial Algebra (VASA) [Pauly and Schneider 2010] is an exact model that offers several spatial operations for vague points, vague lines, and vague regions. Implementations based on this model are

detailed by Pauly and Schneider [2008; 2010][1], and Carniel et al. [2015b]. However, in this article, we propose an abstract data type for vague spatial data based on the fuzzy model. While VASA has only three levels of representation for vague spatial objects (the well-known part, the vague part, and the part that certainly does not belong to the spatial object), vague spatial objects based on the fuzzy model may have several levels in the interval $[0, 1]$. Therefore, it allows a refined and detailed representation of spatial vagueness.

For vague spatial data based on the fuzzy model, the Spatial Plateau Algebra (SPA) [Schneider 2014] provides definitions for fuzzy spatial data that reuses crisp spatial algorithms. Hence, a fuzzy spatial object, called *spatial plateau object*, is defined as a finite sequence of pairs where each pair is formed by one crisp spatial object and a membership degree in $]0, 1]$. In addition, SPA defines spatial plateau operations to handle spatial plateau objects, such as plateau geometric set operations. Although this implementation concept is proposed, spatial plateau objects were not implemented in a spatial DBMS.

Vague spatial data based on the fuzzy model were implemented by Kraipeerapun [2004] and Dilo et al. [2006]. In these approaches, the authors refer to fuzzy spatial objects as vague spatial objects. Thus, they implement the following vague spatial data types: vague point, vague line, and vague region. A vague point object is defined as a tuple (x, y, λ), where $(x, y) \in \mathbb{R}^2$ gives the location, and $\lambda \in]0, 1]$ gives the membership degree. A vague line object is defined as a finite sequence of tuples $((x_1, y_1, \lambda_1), \ldots, (x_n, y_n, \lambda_n))$ for some $n \in \mathbb{N}$, where each tuple is a vague point object in the vague line constructed by using linear interpolation. A vague region object is composed by several vague lines and the Delaunay triangulation, which is stored together with the vague region object. A membership degree of a point inside a vague region object is calculated by using linear interpolation between membership degrees of vertices of the triangle to which the point belongs. This model is implemented in the GRASS GIS[2], but not in a spatial DBMS. Further, the authors do not implement the vague geometric difference between vague lines. However, our article proposes an abstract data type implemented in a spatial DBMS (that is, the PostgreSQL) to manage fuzzy lines and fuzzy regions as well as several fuzzy spatial operations to handle them.

3. TECHNICAL BACKGROUND

In this section, we provide some needed concepts that underline our proposed FuzzyGeometry ADT. Section 3.1 summarizes different models that represent vague spatial data, while Section 3.2 summarizes needed concepts from fuzzy set theory.

3.1 Vague Spatial Data

While crisp spatial objects have exact location and well-known boundaries, vague spatial objects have inexact location, uncertain boundaries, or blurred interior. There are distinct models to represent vague spatial data, which can be classified as *exact models* [Cohn and Gotts 1996; Pauly and Schneider 2008; Bejaoui et al. 2009; Pauly and Schneider 2010], *rough models* [Beaubouef et al. 2004], *probabilistic models* [Cheng et al. 2003; Li et al. 2007; Zinn et al. 2007], and *fuzzy models* [Dilo et al. 2007; Schneider 2008; 2014; Carniel et al. 2014].

Exact models aim to reuse existing abstract data types of crisp spatial data types (for instance, crisp points, crisp lines, and crisp regions) to represent vague spatial objects. In general, vague spatial objects are defined by using two crisp spatial objects. The first spatial object represents the vague spatial part and second spatial object represents the well-known spatial part. The main exact models are *Egg-Yolk* [Cohn and Gotts 1996], *Qualitative Min-Max Model* (QMM) [Bejaoui et al. 2009], and

[1]http://www.cise.ufl.edu/research/SpaceTimeUncertainty/
[2]http://grass.osgeo.org/

Vague Spatial Algebra (VASA) [Pauly and Schneider 2008; 2010]. Egg-Yolk model defines only vague region objects, which are represented by two sub-regions that together form the egg: a sub-region denominates the white (that is, vague spatial part) and other sub-region denominates the yolk (that is, well-known spatial part) that is contained in the white part. QMM model defines vague spatial objects by using two spatial limits, a minimum limit (that is, well-known spatial part) and a maximum limit (that is, includes the minimum limit and extends to the part that possibly belongs to the spatial object). In addition, this model uses qualitative classifications of vagueness levels, such as *completely crisp*, *partially vague*, and *completely vague*. Finally, VASA defines a vague spatial object as a pair of disjoint or adjacent crisp spatial objects of the same type.

Rough models are based on the rough set theory [Pawlak 1982] that defines a lower and an upper approximation. Hence, a vague spatial object is represented by these approximations. Lower spatial approximation of an object is a subset of its upper spatial approximation. Vague spatial data represented by rough models deal with vague spatial objects with inexact location as well as inexact measures [Beaubouef et al. 2004].

Probabilistic models are based on the probability density functions [Cheng et al. 2003; Li et al. 2007; Zinn et al. 2007] and the treatment of spatial vagueness is performed through objects positions and measures. In general, these models deal with expectation of a future event based on the known-characteristics.

Fuzzy models, that is, models based on the fuzzy set theory [Zadeh 1965], assign membership degrees in the interval $[0, 1]$ for each point of a spatial object to represent spatial vagueness in different levels. There are several representations of vague spatial data by using the fuzzy set theory. Fuzzy Minimum Boundary Rectangle [Somodevilla and Petry 2004] includes the fuzzy set theory in order to define membership functions by using several Minimum Boundary Rectangles. Fuzzy spatial data types have also been defined, such as, fuzzy points, fuzzy lines, and fuzzy regions [Dilo et al. 2007; Schneider 2008; Carniel et al. 2014]. To handle them, fuzzy spatial operations are defined [Dilo et al. 2007; Schneider 2008; Carniel et al. 2014], such as fuzzy geometric set operations and fuzzy topological predicates. In addition, vague partitions and their operations are defined [Dilo et al. 2007]. In this article, we consider fuzzy spatial data types defined by Dilo et al. [2007] as design goals to implement FuzzyGeometry ADT.

3.2 Fuzzy Set Theory

Fuzzy set theory [Zadeh 1965] is an extension and generalization of the classic (crisp) set theory. In the classic theory, let X be a crisp set of objects, called *universe*. The subset A of X can be described by a function $\chi_A : X \rightarrow \{0, 1\}$, which for all $x \in X$, $\chi_A(x)$ is 1 if and only if, $x \in A$ and 0 otherwise. On the other hand, fuzzy set theory defines a function $\mu_{\tilde{A}}$ that maps all elements of X in the interval $[0, 1]$ by assigning *membership degrees* in a specific set. Further, fuzzy set theory allows that an element x has different membership values in different fuzzy sets. Let X be the universe. Then, the function $\mu_{\tilde{A}} : X \rightarrow [0, 1]$ is called of *membership function of the fuzzy set* $\tilde{A}$. Therefore, each element of fuzzy set $\tilde{A}$ has a membership degree in the interval $[0, 1]$ according to the membership function: $\tilde{A} = \{(x, \mu_{\tilde{A}}) \,|\, x \in X\}$.

Classic operations among crisp sets also are extended for the fuzzy sets. We will summarize these operations. Let $\tilde{A}$ and $\tilde{B}$ be fuzzy sets in X, then the intersection, union, and difference are defined as follows, respectively.

—$\tilde{A} \cap \tilde{B} = \{(x, \mu_{\tilde{A} \cap \tilde{B}}(x)) \,|\, x \in X \wedge \mu_{\tilde{A} \cap \tilde{B}}(x) = \min(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x))\}$
—$\tilde{A} \cup \tilde{B} = \{(x, \mu_{\tilde{A} \cup \tilde{B}}(x)) \,|\, x \in X \wedge \mu_{\tilde{A} \cup \tilde{B}}(x) = \max(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x))\}$
—$\tilde{A} - \tilde{B} = \{(x, \mu_{\tilde{A} - \tilde{B}}(x)) \,|\, \mu_{\tilde{A} - \tilde{B}}(x) = \min(\mu_{\tilde{A}}(x), 1 - \mu_{\tilde{B}}(x)\}$

An alpha-cut (α-cut) of the fuzzy set $\tilde{A}$ for a specific value α is a crisp set defined as follows, $\tilde{A}^{\geq\alpha} = \{x \in X \,|\, \mu_{\tilde{A}}(x) \geq \alpha \wedge 0 \leq \alpha \leq 1\}$. When α value is 1, the result is called of *core* of $\tilde{A}$.

Generalizations of intersection and union, replace the minimum and maximum operators by *triangular norms* (t-norm) and *triangular co-norms* (s-norm), respectively. A t-norm T is defined as a commutative, associative, non-decreasing binary operation on $[1, 0]$, with signature $T : [0, 1]^2 \to [0, 1]$ satisfying the following boundary conditions, $T(1, x) = x$ and $T(0, x) = 0$ for all $x \in [0, 1]$ [Klement et al. 2000]. Let $x, y \in [0, 1]$, some examples of t-norms are listed as follows.

—$T^*(x, y) = \begin{cases} x & \text{if } y = 1 \\ y & \text{if } x = 1 \\ 0 & \text{otherwise} \end{cases}$ (*drastic intersection*)
—$T_p(x, y) = ab$ (*product t-norm*)
—$T_l(x, y) = \max(0, x + y - 1)$ (*Lukasiewicz t-norm*)

For any t-norm there is an s-norm, which is obtained by De Morgan's laws. Hence, a s-norm is defined as a commutative, associative, non-decreasing binary operation on $[1, 0]$, with signature $S : [0, 1]^2 \to [0, 1]$ satisfying the following boundary conditions, $S(1, x) = 1$ and $S(0, x) = x$ for all $x \in [0, 1]$ [Klement et al. 2000]. Let $x, y \in [0, 1]$, some examples of s-norms are listed as follows.

—$S^*(x, y) = \begin{cases} x & \text{if } y = 0 \\ y & \text{if } x = 0 \\ 1 & \text{otherwise} \end{cases}$ (*drastic union*)
—$S_p(x, y) = x + y - xy$ (*probabilistic sum*)
—$S_l(x, y) = \min(1, x + y)$ (*bounded sum*)

The height of a fuzzy set $\tilde{A}$ is defined as the greatest membership value (sup) of the membership function of $\tilde{A}$ [Jamshidi et al. 1993], that is, $h(\tilde{A}) = sup_x[\mu_{\tilde{A}}(x)]$. A fuzzy set $\tilde{A}$ is called normal when $h(\tilde{A}) = 1$, and subnormal when $h(\tilde{A}) < 1$. In order to normalize a fuzzy set $\tilde{A}$, we apply the normalization operation, which is defined as $Norm_{\mu_{\tilde{A}}}(x) = [\mu_{\tilde{A}}(x)/h(\tilde{A})]$ for all $x \in X$.

The concentration (CON) of a fuzzy set $\tilde{A}$ decreases the fuzziness, while the dilation (DIL) of a fuzzy set $\tilde{A}$ increases the fuzziness [Jamshidi et al. 1993]. They are defined as follows:

—$\mu_{CON(\tilde{A})}(x) = [\mu_{\tilde{A}}(x)]^p$ for all $x \in X$ where $p > 1$
—$\mu_{DIL(\tilde{A})}(x) = [\mu_{\tilde{A}}(x)]^r$ for all $x \in X$ where $r \in]0, 1[$

Finally, there are some notations to textually represent a fuzzy set [Jamshidi et al. 1993]. The following definitions are textual representation of a fuzzy set $\tilde{A}$.

—$\tilde{A} = \sum_{x_i \in X} \mu_{\tilde{A}}(x_i)/x_i$ when X is finite and discrete
—$\tilde{A} = \int_x \mu_{\tilde{A}}(x)/x$ when X is continuous

Note that the signs of sum and integral denote the union of the membership degrees and the slash (/) denotes a separator.

4. THE FUZZYGEOMETRY ABSTRACT DATA TYPE

In this section, we propose a novel ADT to handle fuzzy spatial objects, called FuzzyGeometry. We implement the FuzzyGeometry ADT as a PostgreSQL extension. PostgreSQL is an open-source

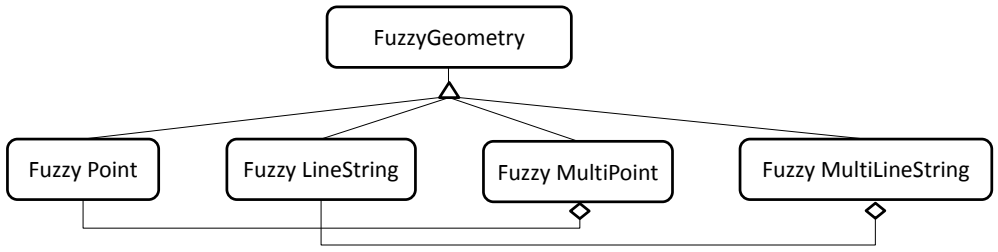


Fig. 1. The hierarchy of the FuzzyGeometry data types.

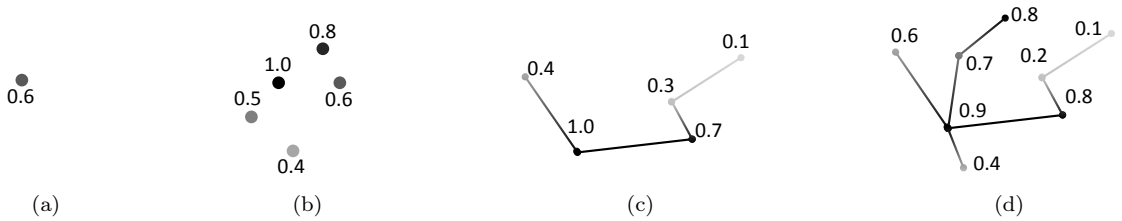


Fig. 2. Examples of FuzzyGeometry objects: (a) a fuzzy point object, (b) a fuzzy multipoint object, (c) a fuzzy linestring object, and (d) a fuzzy multilinestring object. The membership degrees are also showed, where darker parts have higher membership degrees than lighter areas.

software and it is an extensible DBMS where new ADTs can be implemented by using a low level program language (for instance, C language)[3]. FuzzyGeometry ADT is implemented in the C language by using the extensibility provided by the PostgreSQL internal library. A detailed documentation of FuzzyGeometry is available at `http://gbd.dc.ufscar.br/fuzzygeometry/`. Section 4.1 presents the FuzzyGeometry data types and how they are implemented in the PostgreSQL. Section 4.2 defines textual and binary representations of FuzzyGeometry objects. Section 4.3 details fuzzy spatial operations to handle FuzzyGeometry objects in SQL queries.

4.1 Fuzzy Spatial Data Types of the FuzzyGeometry

Figure 1 depicts the fuzzy spatial data types of the FuzzyGeometry ADT. The highest level of hierarchy is the *FuzzyGeometry data type*. The FuzzyGeometry data type is a generalization for fuzzy points and fuzzy lines, which can be simple or complex. The *fuzzy point* data type represents simple fuzzy points, while the *fuzzy multipoint* data type represents complex fuzzy points. Similarly, the *fuzzy linestring* data type represents complex fuzzy lines, while the *fuzzy multilinestring* represents complex fuzzy lines. Note that the current version of the FuzzyGeometry ADT does not provide support for fuzzy regions. Figure 2 depicts examples of FuzzyGeometry objects.

In general, FuzzyGeometry objects have a membership degree for each point in the space to denote spatial imprecision. Such membership degree is a value in the interval $[0, 1]$ that determines the spatial vagueness in a given point. In the following, we will detail each data type of the FuzzyGeometry ADT according to Figure 1 by showing its structure. Hence, these structures show the attributes that compose fuzzy spatial objects and how they are specified in the low-level implementation of FuzzyGeometry ADT.

The main structure of the FuzzyGeometry ADT is FUZZYGEOM. It handles the FuzzyGeometry data type, which is a generic data type that can be instanced as fuzzy point objects or fuzzy line

[3]http://www.postgresql.org/docs/9.5/static/xfunc-c.html

objects (Figure 1). Algorithm 1 shows the specification of FUZZYGEOM, which has the following attributes:

—The *type* attribute (line 2) is an identifier that indicates the data type of a FuzzyGeometry object (Figure 1), which can be *fuzzy point*, *fuzzy multipoint*, *fuzzy linestring*, and *fuzzy multilinestring*.

—The *srid* attribute (line 3) stores the Spatial Reference System Identifier (SRID), which indicates spatial coordinate system definitions.

—The *bbox* attribute (line 4) stores the Minimum Boundary Rectangle (MBR) of a FuzzyGeometry object. Algorithm 2 details BBOX, which is composed by the minimum and maximum coordinates of each axis, that is, x and y.

—The *data* attribute (line 5) stores the remaining data related to the FuzzyGeometry object. This is a void pointer to store any kind of data in order to aid in the conversion between FUZZYGEOM and similar structures, such as those that define a fuzzy line object.

In order to aid the handling of FuzzyGeometry objects in main memory, we create a new structure that stores a serialized array of spatial coordinates x, y and its membership degree, that is, fuzzy points. This structure is named FPOINTARRAY, which is also defined to perform generic operations in serialized arrays, such as insertion, modification, and retrieval of elements. To efficiently perform these operations, we employ main memory operations, such as *memcpy* and *memmove*. We use serialized array of fuzzy points since the PostgreSQL internal library requires that an object should be serialized to be stored in a PostgreSQL table. To handle serialized fuzzy points, we defined the structure named FPOINT.

Algorithm 3 details the structures FPOINTARRAY and FPOINT. In FPOINTARRAY (lines 1 to 5), the *serialized_fpointlist* attribute (line 2) stores the serialized array of fuzzy points, the *npoints* attribute (line 3) stores the current number of stored points, and the *maxpoints* attribute (line 5) stores the maximum number of points that can be stored in the array. FPOINT (lines 7 to 11) contains the coordinates x and y and its membership degree as attributes.

By using FPOINTARRAY, we are able to define the other structures. Algorithm 4 details FUZZYPOINT and FUZZYMULTIPOINT, which represent the fuzzy point and the fuzzy multipoint data types (Figure 1), respectively. The difference between FUZZYPOINT and FPOINT (Algorithm 3) is that the later only deals with fuzzy points to be serialized in main memory, while the former indeed handle instances of the fuzzy point data type. As a result, we are able to convert an instance of FUZZYPOINT into an instance of FUZZYGEOM. Similarly, we also are able to make conversions between instances of FUZZYMULTIPOINT and FUZZYGEOM. In order to aid these conversions (that is, explicit type conversions in C), the first three attributes of FUZZYPOINT (lines 2 to 4) and FUZZYMULTIPOINT (lines 9 to 11) are the same of the first three attributes of FUZZYGEOM (lines 2 to 4 in Algorithm 1). The last attribute of FUZZYPOINT, *point*, is a pointer to an instance of FPOINTARRAY that stores the value 1 in its *npoints* and *maxpoints* attributes since FUZZYPOINT represents the fuzzy point data type. On the other hand, the *point* attribute of FUZZYMULTIPOINT stores a value $n \in \mathbb{N}$ in the *npoints* attribute and a greater value in the *maxpoints* attribute (for instance, $2n$) since it represents the fuzzy multipoint data type.

Algorithm 1: The structure that represents the FuzzyGeometry data type (Figure 2).

```
1 typedef struct {
2   uint8_t type;
3   int32_t srid;
4   BBOX *bbox;
5   void *data;
6 } FUZZYGEOM;
```

Algorithm 2: The structure that represents a MBR of a FuzzyGeometry object.

```
1 typedef struct {
2   double xmin;
3   double xmax;
4   double ymin;
5   double ymax;
6 } BBOX;
```

Algorithm 3: The structures that store a serialized array (FPOINTARRAY) of fuzzy points (FPOINT).

```
1 typedef struct {
2   uint8_t *serialized_fpointlist;
3   int npoints;
4   int maxpoints;
5 } FPOINTARRAY;
6
7 typedef struct {
8   double x;
9   double y;
10  double u;
11 } FPOINT;
```

Algorithm 4: The structures that represent the fuzzy point data type (FUZZYPOINT) and fuzzy multipoint data type (FUZZYMULTIPOINT).

```
1 typedef struct {
2   uint8_t type;
3   int32_t srid;
4   BBOX *bbox;
5   FPOINTARRAY *point;
6 } FUZZYPOINT;
7
8 typedef struct {
9   uint8_t type;
10  int32_t srid;
11  BBOX *bbox;
12  FPOINTARRAY *points;
13 } FUZZYMULTIPOINT;
```

Algorithm 5 details the structures FUZZYLINE and FUZZYMULTILINE for the fuzzy linestring and fuzzy multilinestring data types (Figure 1), respectively. As for FUZZYPOINT and FUZZYMULTIPOINT, the first three attributes of each structure (lines 2 to 4 in FUZZYLINE and lines 10 to 12 in FUZZYMULTILINE) are the same of FUZZYGEOM (Algorithm 1). The *point* attribute of FUZZYLINE stores the fuzzy points that compose a fuzzy linestring object. Further, this attribute should has at least two fuzzy points and there is no intersection in its line segments. A line segment of a fuzzy linestring object is a consecutive pair of serialized fuzzy points (that is, instances of FPOINTs). The *interpolation* attribute indicates the type of interpolation used to calculate the membership degree of a point in a line segment. We have implemented the linear interpolation, although other types of interpolations can also be considered as possible extensions. FUZZYMULTILINE has the following attributes: *ngeoms*, *maxgeoms*, and *geoms*. The *ngeoms* attribute contains the number of fuzzy linestring stored in the *geoms* attribute, while *maxgeoms* attribute contains the maximum fuzzy linestring objects that can be stored. This means that FUZZYMULTILINE is formed by a set of pointers of instances of FUZZYLINEs stored in the *geoms* attribute.

Algorithm 5: The structures that represent the fuzzy line data type (FUZZYLINE) and fuzzy multiline data type (FUZZYMULTILINE).

```
1  typedef struct {
2    uint8_t type;
3    int32_t srid;
4    BBOX *bbox;
5    FPOINTARRAY *point;
6    uint8_t interpolation
7  } FUZZYLINE;
8
9  typedef struct {
10   uint8_t type;
11   int32_t srid;
12   BBOX *bbox;
13   int ngeoms
14   int maxgeoms
15   FUZZYLINE **geoms;
16 } FUZZYMULTILINE;
```

Algorithm 6: The structure that stores a FuzzyGeometry object in the PostgreSQL.

```
1 typedef struct {
2   uint32_t size;
3   uint8_t srid[4];
4   uint8_t data[1];
5 } FGSERIALIZED;
```

The aforementioned structures (Algorithms 1, 4, and 5) are able to handle only FuzzyGeometry data types in the main memory and cannot be stored directly in the PostgreSQL. This is due the fact that the PostgreSQL internal library[4] only handles streams of bytes with an alignment, that is, serialized data. Since a pure implementation based on stream of bytes would be complex, we employ conversions between the internal (main memory) structures and a serialization structure, named FGSERIALIZED. These conversions are efficiently performed since FPOINTARRAY deals with serialized fuzzy points.

Algorithm 6 details FGSERIALIZED, which has the following attributes: *size*, *srid*, and *data*. The *size* attribute is required by the PostgreSQL internal library and stores the size in bytes of the object to be stored in the database. The *srid* attribute is a serialized integer value and corresponds to the *srid* attribute of FUZZYGEOM (Algorithm 1). The *data* attribute stores the remaining data of a FuzzyGeometry object, that is, the MBR, an identifier of the FuzzyGeometry data type, and the spatial coordinates with their respective membership degrees. The serialization alignment is 8, which means that the final size of the object have to be a multiple of 8. Table I depicts the serialization order of each FuzzyGeometry data type. This table uses the following notation to show the size in bytes of each serialized element. Elements between <> have 4 bytes, while elements between [ ] have a size multiple of 8 bytes. Note that for the fuzzy linestring data type, we need to add a padding of 4 bytes to keep the 8-bytes alignment.

Next, we have to use the SQL CREATE TYPE command to create the FuzzyGeometry data type in the PostgreSQL, as showed in Algorithm 7. Note that only the FuzzyGeometry data type is created since it can be instanced as fuzzy points and fuzzy lines. The *internallength* parameter indicates that the size in bytes of FuzzyGeometry objects is variable since the number of coordinates is not fixed. The *input* and *output* parameters are functions in C that transform a textual representation of a FuzzyGeometry object into its internal representation as well as the inverse, respectively. Similarly, the *send* and *receive* parameters transform a binary representation into an internal representation

[4]http://www.postgresql.org/docs/9.5/static/xtypes.html

Table I. Serialization order of a FuzzyGeometry object stored in the PostgreSQL.

FuzzyGeometry Data Type	Serialization Order
Fuzzy Point	[MBR] <data type identifier> <number of the fuzzy points (0 for empty and 1 otherwise)> [x coordinate] [y coordinate] [membership degree]
Fuzzy Multipoint	[MBR] <data type identifier> <number of the fuzzy points (0 for empty and 1 otherwise)> [x coordinate] [y coordinate] [membership degree]
Fuzzy Linestring	[MBR] <data type identifier> <type of interpolation> <number of the fuzzy points (0 for empty and 2 otherwise)> <*padding*> [x coordinate] [y coordinate] [membership degree]
Fuzzy Multilinestring	[MBR] <data type identifier> <number of the fuzzy linestrings> for each fuzzy linestring i [serialization of the fuzzy linestring i]

Algorithm 7: The CREATE TYPE command to create the FuzzyGeometry data type in the PostgreSQL.

```
1  CREATE TYPE FuzzyGeometry (
2    internallength = variable,
3    input = fg_in,
4    output = fg_out
5    send = fg_send,
6    receive = fg_recv,
7    typmod_in = fg_typmod_in,
8    typmod_out = fg_typmod_out,
9    category = 'G',
10   alignment = double,
11   storage = main
12 );
```

as well as the inverse, respectively. The textual and binary representations of the FuzzyGeometry data types are presented in Section 4.2. The *typmod_in* and *typmod_out* parameters manipulate the FuzzyGeometry data types. The *category* parameter defines that a FuzzyGeometry object is a geometric data type. The *alignment* parameter determines the size of the alignment of the serialized data, which is specified to be 8 bytes. Finally, the *storage* parameter specifies the strategy of storage that the PostgreSQL will use. We specified to be *main* since it preferentially stores objects in the main table with possible compressions. The complete documentation of the CREATE TYPE command can be found in the PostgreSQL documentation[5].

[5]http://www.postgresql.org/docs/9.5/static/sql-createtype.html

Table II. Examples of FWKT representations.

FuzzyGeometry Data Type	FWKT Representation of a FuzzyGeometry object
Fuzzy Point	FUZZYPOINT(0.7/10 30)
Fuzzy MultiPoint	FUZZYMULTIPOINT(0.5/10 20 + 0.8/5 3 + 0.1/30 20)
Fuzzy LineString	FUZZYLINESTRING(0.5/1 1 + 0.7/2 2 + 1.0/4 4)
Fuzzy MultiLineString	FUZZYMULTILINESTRING((0.3/1 1 + 0.5/2 2), (0.7/3 3 + 1.0/4 4))

4.2 Textual and Binary Representations

In order to insert and retrieve FuzzyGeometry objects, we define textual and binary representations for each FuzzyGeometry data type. In addition, these representations allow interoperability between spatial applications since a FuzzyGeometry object has a unique representation. The textual definition can be used for general purpose, while the binary representation can be used in applications that communicate by using network protocols or manage binary data.

We define the *Fuzzy Well-Known Text* (FWKT) as textual representation, which is based on the textual representation of fuzzy sets (Section 3.2) and the Well-Known Text representation of the Open Geospatial Consortium (OGC)[6]. In general, the FWKT representation of a FuzzyGeometry object starts with the name of its data type and then, in parentheses, the membership degree and coordinate pairs of each point. Empty FuzzyGeometry objects, which contain no coordinates and membership degrees, can be specified by using the symbol EMPTY after the data type name. Table II shows examples of FWKT representations for each FuzzyGeometry data type (Figure 1). Let (x, y) be a coordinate pair, u be a membership degree in real interval $]0, 1]$, and $k, j \in \mathbb{N}$. Then, we define the textual representation FWKT for (i) fuzzy point, (ii) fuzzy multipoint, (iii) fuzzy linestring, and (iv) fuzzy multilinestring data types, as follows:

(i) FUZZYPOINT($u/x\ y$)
(ii) FUZZYMULTIPOINT($u_1/x_1\ y_1 + ... + u_k/x_k\ y_k$)
(iii) FUZZYLINESTRING($u_1/x_1\ y_1 + ... + u_k/x_k\ y_k$)
(iv) FUZZYMULTILINESTRING($(u_{1_1}/x_{1_1}\ \ y_{1_1} + ... + u_{k_1}/x_{k_1}\ \ y_{k_1})_1, ..., (u_{1_j}/x_{1_j}\ \ y_{1_j} + ... + u_{k_j}/x_{k_j}\ y_{k_j})_j$)

We define the *Fuzzy Well-Known Binary* (FWKB) as binary representation, which is based on the Well-Known Binary representation of the OGC. To aid in the FWKB definition, we use the $id(\tilde{A})$ function, which extracts the binary format of the data type identifier of a FuzzyGeometry object. It extracts the identifier 1 if $\tilde{A}$ is a fuzzy point object, the identifier 2 if $\tilde{A}$ is a fuzzy linestring object, the identifier 3 if $\tilde{A}$ is a fuzzy multipoint object, and the identifier 4 if $\tilde{A}$ is a fuzzy multilinestring object. Further, we use the *binary* function that extract the binary format of a double value. Let n be the number of points in a fuzzy multipoint and fuzzy linestring object, l be the number of lines in a fuzzy multilinestring object, and p be a fuzzy point with the coordinates x and y with a membership degree u. Then, we define the binary representation FWKB for (i) fuzzy point, (ii) fuzzy multipoint, (iii) fuzzy linestring, and (iv) fuzzy multilinestring data types, as follows:

(i) $endianness + id(A) + binary(u) + binary(x) + binary(y)$
(ii) $endianness + id(A) + binary(n) + \sum_{p_i \in A}^{n}(binary(u)_{p_i} + binary(x)_{p_i} + binary(y)_{p_i})$
(iii) $endianness + id(A) + binary(n) + \sum_{p_i \in A}^{n}(binary(u)_{p_i} + binary(x)_{p_i} + binary(y)_{p_i})$
(iv) $endianness + id(A) + binary(l) + \sum_{l_i \in A}^{l} FWKB \text{ of } A_{l_i}$

[6]http://www.opengeospatial.org/

The signs of sum are used to denote the union between binary data and do not perform the arithmetic sum. Further, *endianness* indicates the way in which the bytes are organized in main memory (that is, either *big-endian* or *little-endian*).

4.3 Fuzzy Spatial Operations of the FuzzyGeometry

FuzzyGeometry ADT provides support to *input and output operations*, *fuzzy geometric set operations*, and *fuzzy spatial operations based on the fuzzy set theory*. The following subsections detail these operations by providing examples and their SQL signatures.

4.3.1 *Input and Output Operations.* The input operations transform a FuzzyGeometry object represented in the FWKT and FWKB representations into the internal (main memory) format. These operations are mainly employed to insert new FuzzyGeometry objects in a PostgreSQL relational table. The output operations yield the FWKT and FWKB representations of a FuzzyGeometry object and are applied to retrieve FuzzyGeometry objects in SQL queries. The SQL signatures for input and output operations are defined as follows, respectively:

(i) *FG_FuzzyGeomFromText*(text *FWKT*, integer *SRID*) → FuzzyGeometry
(ii) *FG_FuzzyGeomFromBinary*(bytea *FWKB*, integer *SRID*) → FuzzyGeometry
(iii) *FG_AsText*(FuzzyGeometry *fg*) → text
(iv) *FG_AsFWKB*(FuzzyGeometry *fg*) → bytea

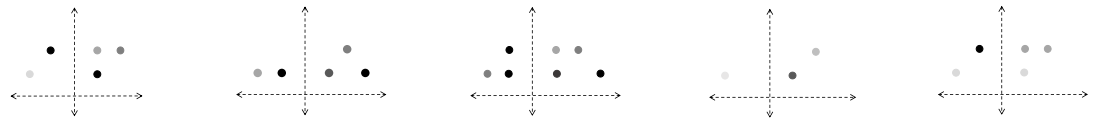
Operation (i) is the input operation for the FWKT representation, while operation (ii) is the input operation for the FWKB representation. These operations also receive the SRID as parameter. Operations (iii) and (iv) are the output operations for the FWKT and FWKB representations, respectively.

4.3.2 *Fuzzy Geometric Set Operations.* FuzzyGeometry provides support for the following fuzzy geometric set operations: union, intersection, and difference. In general, we used the formal definition provided by fuzzy models that define fuzzy spatial data (Section 3.1) to implement them. We define the following SQL signatures:

(i) *FG_Union*(FuzzyGeometry *fg1*, FuzzyGeometry *fg2*, text *s*) → FuzzyGeometry
(ii) *FG_Union*(FuzzyGeometry set *fg_field*) → FuzzyGeometry
(iii) *FG_Intersection*(FuzzyGeometry *fg1*, FuzzyGeometry *fg2*, text *t*) → FuzzyGeometry
(iv) *FG_Difference*(FuzzyGeometry *fg1*, FuzzyGeometry *fg2*, text *d*) → FuzzyGeometry

Operation (i) is the union operation between FuzzyGeometry objects, which is performed by the spatial union and the fuzzy union to calculate the membership degree of intersecting points. The fuzzy union can be executed by using a specific s-norm *s*, which can be the default union (max operator), drastic union, probabilistic sum, and bounded sum (Section 3.2). In addition, the union is only executed for FuzzyGeometry objects of the same data type. For instance, the union of two fuzzy linestring objects yields other fuzzy linestring object. In addition, the union operation can be used as an aggregate function, that is, the operation (ii). This means that, given a set of FuzzyGeometry objects, this operation yields the union among all the objects contained in this set. The strategy to compute this aggregation is to perform the union operation (i) incrementally for each FuzzyGeometry object contained in the set by considering the default union to calculate the membership degree of intersecting points. Figure 3c shows the result of the union operation between two fuzzy multipoint objects (Figures 3a and 3b) by applying the probabilistic sum as s-norm.

Operation (iii) is the intersection operation between FuzzyGeometry objects, which is performed by the spatial intersection and the fuzzy intersection to calculate the membership degree of intersecting



(a) FUZZYMULTIPOINT(0.2/-2 1 + 1/-1 2 + 1/1 1 + 0.4/1 2 + 0.6/2 2)

(b) FUZZYMULTIPOINT(0.4/-2 1 + 1/-1 1 + 0.8/1 1 + 0.6/2 2 + 1/3 1)

(c) FUZZYMULTIPOINT(0.68/-2 1 + 1/-1 2 + 1/-1 1 + 1/1 1 + 0.4/1 2 + 0.84/2 2 + 1/3 1)

(d) FUZZYMULTIPOINT(0.08/-2 1 + 0.8/1 1 + 0.36/2 2)

(e) FUZZYMULTIPOINT(0.2/-2 1 + 1/-1 1 + 0.2/1 1 + 0.4/1 2 + 0.4/2 2)

Fig. 3. Result of the fuzzy geometric set operations on two FuzzyGeometry objects (a) and (b): (c) union operation, (d) intersection operation, and (e) difference operation. Each FuzzyGeometry object has its FWKT representation.

points. The fuzzy intersection can be executed by using a specific t-norm t, which can be the default t-norm (min operator), drastic intersection, product t-norm, and Lukasiewicz t-norm (Section 3.2). The intersection can be executed between FuzzyGeometry objects of different types, and the resulting FuzzyGeometry object is the lower data type by considering the hierarchy: *fuzzy linestring* > *fuzzy point*. For instance, the intersection between fuzzy linestring objects and fuzzy point objects yields a fuzzy point or a fuzzy multipoint object composed by the commons points with membership degrees calculated by using a t-norm t. Figure 3d shows the result of the intersection operation between two fuzzy multipoint objects (Figures 3a and 3b) by applying the product t-norm as t-norm.

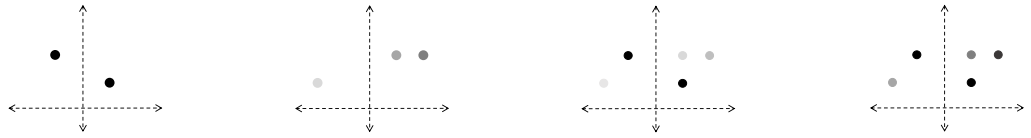
Operation (iv) is the difference operation between FuzzyGeometry objects, which is performed by the spatial difference and the membership degrees of intersecting points are calculated by using a difference operator. The supported difference operators are the fuzzy difference (Section 3.2) and bounded difference. The bounded difference is defined by $diff(a, b) = a - b$ if $a > b$ and 0 otherwise. The difference operation is only performed between FuzzyGeometry objects of the same data type. For instance, the difference between fuzzy linestring objects yields a fuzzy linestring object. Figure 3e shows the result of the fuzzy difference operation between the two fuzzy multipoint objects depicted in Figures 3a and 3b.

4.3.3 *Fuzzy Spatial Operations Based on the Fuzzy Set Theory.* FuzzyGeometry provides support for the following fuzzy spatial operations based on the fuzzy set theory: core, boundary, concentration, dilation, alpha-cut, height, and normalization. These operations can be applied in any type of FuzzyGeometry object, and they have the following SQL signatures:

(i) *FG_ Core*(FuzzyGeometry *fg*) → FuzzyGeometry
(ii) *FG_ Boundary*(FuzzyGeometry *fg*) → FuzzyGeometry
(iii) *FG_ Concentration*(FuzzyGeometry *fg*, double *p*) → FuzzyGeometry
(iv) *FG_ Dilation*(FuzzyGeometry *fg*, double *r*) → FuzzyGeometry
(v) *FG_ Alphacut*(FuzzyGeometry *fg*, double α) → FuzzyGeometry
(vi) *FG_ Height*(FuzzyGeometry *fg*) → double
(vii) *FG_ Normalization*(FuzzyGeometry *fg*) → FuzzyGeometry

Operations (i) and (ii) correspond to the core and boundary operations, which return the locations that have exact locations (locations with membership degree equal to 1) and vague locations (locations with membership degree less than 1 and greater than 0), respectively. Figures 4a and 4b depict the result of the core and boundary operations on the fuzzy multipoint object of Figure 3a.

Operations (iii) and (iv) correspond to the fuzzy concentration (iii) and dilation (iv) operations (Section 3.2), which decrease and increase the membership degrees of the locations according to p and r, respectively. These operations are useful to analyze or edit locations in order to intensify or smooth



FUZZYMULTIPOINT(1/-1 2 + 1/1 1)

(a)

FUZZYMULTIPOINT(0.2/-2 1 + 0.4/1 2 + 0.6/2 2)

(b)

FUZZYMULTIPOINT(0.04/-2 1 + 1/-1 2 + 1/1 1 + 0.16/1 2 + 0.36/2 2)

(c)

FUZZYMULTIPOINT(0.447/-2 1 + 1/-1 2 + 1/1 1 + 0.632/1 2 + 0.774/2 2)

(d)

Fig. 4. Result of the core (a), boundary (b), concentration (c), and dilation (d) operations on the FuzzyGeometry object of Figure 3a, with their FWKT representations.

the spatial vagueness. Figures 4c and 4d depicts the result of concentration and dilation operations on the fuzzy multipoint object of Figure 3a.

Operation (v) performs the fuzzy alpha-cut (Section 3.2), which filters the locations that have membership degree equal to or greater to an α value. This means that, this operation identifies locations that contain specific membership degrees. Operation (vi) corresponds to the fuzzy spatial height that returns the greatest membership degree of a FuzzyGeometry object. This operation is necessary for the fuzzy spatial normalization operation (operation (vii)). If the height of a FuzzyGeometry object is 1, than it has a core (for instance, the multipoint object in Figure 3a). Otherwise, the normalization can be used to create a core, which will be the locations with the greatest membership degree.

5. RUNNING EXAMPLE

This section provides a running example to illustrate the functionalities of FuzzyGeometry ADT by using the SQL language. The context of this running example is to manage plagues and animal routes in an ecological application. Hence, we consider the following relational table schemas *plague* and *animal*. Each table has an attribute *id* as primary key and an attribute *geo* to store FuzzyGeometry objects. While plagues are represented by fuzzy multipoint objects, animal routes are represented by fuzzy multilinestring objects. The following SQL command exemplifies the creation of the table *plague*. Note that the FuzzyGeometry data type is specified between parentheses in the attribute *geo*.

```
CREATE TABLE plague(
  id INTEGER PRIMARY KEY,
  geo FUZZYGEOMETRY(FUZZYMULTIPOINT));
```

Similarly, we are able to create the table to stored the animal routes. By using textual and binary representations (Section 4.2), we are able to insert FuzzyGeometry objects in tables. The next SQL command shows an insertion of the fuzzymultipoint object of Figure 3a in the plague table.

```
INSERT INTO plague VALUES (1,
  FG_FuzzyGeomFromText('FUZZYMULTIPOINT(0.2/-2 1 + 1/-1 2 + 1/1 1 + 0.4/1 2 +
                       0.6/2 2)', 4326))
```

Once we have inserted FuzzyGeometry objects in its respective tables, we are able to perform SQL queries on them. The following query returns the intersecting parts among all animal routes. This query shows the result of this intersection operation (Section 4.3.2) by using the FWKT representation.

```
SELECT FG_AsText(FG_Intersection(A.geo, B.geo, 'default t-norm'))
FROM animal as A, animal as B
WHERE A.id <> B.id
```

The next SQL query returns the intersecting points between the animal route with *id* equal to 5 and all the plagues of the environment. Hence, we employ the union operation as an aggregate function on all the plagues, that is, a sequence of union operations explained in Section 4.3.2. Then, we compute the intersection between this object and the specific animal route.

```
SELECT FG_AsText(FG_Intersection(P.all, A.geo))
FROM (SELECT FG_Union(geo) as all FROM plague) as P, animal as A
WHERE A.id = 5
```

The next SQL query returns the plagues with membership degrees greater than 0.8 (by using the alpha-cut operation described in Section 4.3.3). This means that these plagues need some treatment of the decision makers since they have high membership degrees.

```
SELECT FG_AsText(FG_Alphacut(P.geo, 0.8))
FROM plague as P
```

The next query changes the meaning of spatial vagueness in an animal route (*id* equal to 10) by decreasing its membership degrees. For this purpose, we use the concentration operation described in Section 4.3.3. This means that, due to some factors (for instance, a strong rain), the imprecision of this animal route is greater than the original.

```
SELECT FG_AsText(FG_Concentration(A.geo, 2))
FROM animal as A
WHERE id = 10
```

On the other hand, the final SQL query employs the dilation operation introduced in Section 4.3.3 to increase the membership degrees of another animal route (*id* equal to 11). As a result, this query returns an animal route with lesser imprecision than the original.

```
SELECT FG_AsText(FG_Dilation(A.geo, 0.5))
FROM animal as A
WHERE id = 11
```

6. CONCLUSIONS AND FUTURE WORK

This article proposes a novel abstract data type, called FuzzyGeometry, to handle fuzzy spatial objects based on the fuzzy model in PostgreSQL. Fuzzy spatial data are an important representation of real-world phenomena characterized by spatial vagueness (fuzziness), that is, inexact location or uncertain boundaries and interiors. Although it can be adequately used to represent spatial vagueness, implementations based on the fuzzy model are limited and have not been incorporated into spatial DBMS. Due to the complexity and the limited work on specification of implementation of fuzzy spatial data types, we focus on the implementation of simple and complex fuzzy points and fuzzy lines only. As a result, FuzzyGeometry ADT is an important initial work for this research area, which advances in the state of art to handle vague spatial objects based on the fuzzy model in the PostgreSQL.

Several operations have been proposed and implemented to handle FuzzyGeometry objects, such as input and output operations, fuzzy geometric set operations, and fuzzy spatial operations based on the fuzzy set theory. To deal with input and output operations, we propose textual and binary representations of fuzzy lines and fuzzy points. As a result, users can insert, retrieve, and handle fuzzy lines and fuzzy points in spatial databases by using our proposed FuzzyGeometry ADT. It is possible by executing SQL commands, which provides a flexible management of the FuzzyGeometry objects in spatial databases.

Future work will deal with a number of topics. First, the implementation for the fuzzy region data type is needed. Some possible implementations can be based on the plateau spatial algebra [Schneider

2014] and on the triangular irregular networks [Dilo et al. 2006]. Second, we will design the specification and implementation of fuzzy topological predicates [Carniel et al. 2014] since topological relationships are often used by spatial applications. Third, since we are using fuzzy set theory, we are able to use fuzzy inference [Jamshidi et al. 1993] in order to extract information from spatial fuzziness, such as discussed by Carniel et al. [2015]. Fourth, we will design and implement operations by using the PostGIS[7] spatial extension in order to convert crisp spatial objects to fuzzy spatial objects and vice versa. Finally, we aim to conduct experiments in order to evaluate the correctness and performance of the FuzzyGeometry operations.

Acknowledgments. This work has been supported by the Brazilian federal research agencies CAPES and CNPq as well as by the São Paulo Research Foundation (FAPESP). A. C. Carniel has been supported by the grants #2012/12299-8 and #2015/26687-8, FAPESP. R. R. Ciferri has been supported by the grant #311868/2015-0, CNPq. C. D. A. Ciferri has been supported by the grant #2016/04990-3, FAPESP.

REFERENCES

Beaubouef, T., Ladner, R., and Petry, F. Rough Set Spatial Data Modeling for Data Mining. *International Journal of Intelligent Systems* 19 (7): 567–584, 2004.

Bejaoui, L., Pinet, F., Bedard, Y., and Schneider, M. Qualified Topological Relations Between Spatial Objects with Possible Vague Shape. *International Journal of Geographical Information Science* 23 (7): 877–921, 2009.

Carniel, A. C., Ciferri, R. R., and Ciferri, C. D. A. An Abstract Data Type to Handle Vague Spatial Objects Based on the Fuzzy Model. In *Proceedings of the Brazilian Symposium on GeoInformatics*. Campos do Jordão, Brazil, pp. 210–221, 2015a.

Carniel, A. C., Ciferri, R. R., and Ciferri, C. D. A. Embedding Vague Spatial Objects into Spatial Databases using the VagueGeometry Abstract Data Type. In *Proceedings of the Brazilian Symposium on GeoInformatics*. Campos do Jordão, Brazil, pp. 233–244, 2015b.

Carniel, A. C., Schneider, M., and Ciferri, R. R. FIFUS: A Rule-based Fuzzy Inference Model for Fuzzy Spatial Objects in Spatial Databases and GIS. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. New York, USA, pp. 83:1–83:4, 2015.

Carniel, A. C., Schneider, M., Ciferri, R. R., and Ciferri, C. D. A. Modeling Fuzzy Topological Predicates for Fuzzy Regions. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. New York, USA, pp. 529–532, 2014.

Cheng, R., Kalashnikov, D. V., and Prabhakar, S. Evaluating Probabilistic Queries over Imprecise Data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. New York, USA, pp. 551–562, 2003.

Cohn, A. G. and Gotts, N. M. The 'Egg-Yolk' Representation of Regions with Indeterminate Boundaries. In P. A. Burrough and A. U. Frank (Eds.), *Geographic Objects with Indeterminate Boundaries*. Taylor & Francis, Great Britain, pp. 171–187, 1996.

Dilo, A., Bos, P., Kraipeerapun, P., and de By, R. A. Storage and Manipulation of Vague Spatial Objects Using Existing GIS Functionality. In *Flexible Databases Supporting Imprecision and Uncertainty*, G. Bordogna and G. Psaila (Eds.). Vol. 203. Springer Berlin Heidelberg, pp. 293–321, 2006.

Dilo, A., de By, R. A., and Stein, A. A System of Types and Operators for Handling Vague Spatial Objects. *International Journal of Geographical Information Science* 21 (4): 397–426, 2007.

Güting, R. H. An Introduction to Spatial Database Systems. *The VLDB Journal* 3 (4): 357–399, 1994.

Jamshidi, M., Vadiee, N., and Ross, T. J. *Fuzzy Logic and Control*. Prentice-Hall, 1993.

Klement, E. P., Mesiar, R., and Pap, E. *Triangular Norms*. Springer, 2000.

Kraipeerapun, P. *Implementation of Vague Spatial Objects*. M.S. thesis, International Institute for Geo-Information Science and Earth Observation, The Netherlands, 2004.

Li, R., Bhanu, B., Ravishankar, C., Kurth, M., and Ni, J. Uncertain Spatial Data Handling: modeling, indexing and query. *Computers & Geosciences* 33 (1): 42–61, 2007.

Pauly, A. and Schneider, M. Querying Vague Spatial Objects in Databases with VASA. In A. Stein, W. Shi, and W. Bijker (Eds.), *Quality Aspects in Spatial Data Mining*. CRC Press, USA, pp. 3–14, 2008.

Pauly, A. and Schneider, M. VASA: An Algebra for Vague Spatial Data in Databases. *Information Systems* 35 (1): 111–138, 2010.

[7]http://postgis.net/

Pawlak, Z. Rough Sets. *International Journal of Computer & Information Sciences* 11 (5): 341–356, 1982.

Schneider, M. Fuzzy Spatial Data Types for Spatial Uncertainty Management in Databases. In *Handbook of Research on Fuzzy Information Processing in Databases*, J. Galindo (Ed.). IGI Global, pp. 490–515, 2008.

Schneider, M. Spatial Plateau Algebra for Implementing Fuzzy Spatial Objects in Databases and GIS: spatial plateau data types and operations. *Applied Soft Computing* 16 (3): 148–170, 2014.

Schneider, M. and Behr, T. Topological Relationships between Complex Spatial Objects. *ACM Transactions on Database Systems* 31 (1): 39–81, 2006.

Siqueira, T. L., Ciferri, C. D. A., Times, V. C., and Ciferri, R. R. Modeling Vague Spatial Data Warehouses Using the VSCube Conceptual Model. *Geoinformatica* 18 (2): 313–356, 2014.

Somodevilla, M. J. and Petry, F. E. Indexing Mechanisms to Query FMBRs. In *Proceedings of the IEEE Annual Meeting of the Fuzzy Information*. pp. 198–202, 2004.

Zadeh, L. A. Fuzzy Sets. *Information and Control* 8 (8): 338–353, 1965.

Zinn, D., Bosch, J., and Gertz, M. Modeling and Querying Vague Spatial Objects Using Shapelets. In *Proceedings of the International Conference on Very Large Data Bases*. Vienna, Austria, pp. 567–578, 2007.