DOI: 10.1111/tgis.12221

RESEARCH ARTICLE

WILEY Transactions in GIS

Establishing robustness of a spatial dataset in a tolerance-based vector model

Alberto Belussi[1] | Sara Migliorini[1] | Mauro Negri[2] | Giuseppe Pelagatti[2]

[1] Department of Computer Science, University of Verona, Verona, Italy

[2] Department of Electronics, Information and Bioeng. Politecnico of Milan, Milan, Italy

Correspondence
Sara Migliorini, Department of Computer Science, University of Verona, Verona, Italy.
Email: sara.migliorini@univr.it

Abstract

Spatial data are usually described through a vector model in which geometries are represented by a set of coordinates embedded into an Euclidean space. The use of a finite representation, instead of the real numbers theoretically required, causes many robustness problems which are well known in the literature. Such problems are made even worse in a distributed context, where data is exchanged between different systems and several perturbations can be introduced in the data representation.

In order to discuss the robustness of a spatial dataset, two implementation models have to be distinguished: the identity and the tolerance model. The robustness of a dataset in the identity model has been widely discussed in the literature and some algorithms of the Snap Rounding (SR) family can be successfully applied in such contexts. Conversely, this problem has been less explored in the tolerance model. The aim of this article is to propose an algorithm inspired by those of the SR family for establishing or restoring the robustness of a vector dataset in the tolerance model. The main ideas are to introduce an additional operation which spreads instead of snapping geometries, in order to preserve the original relation between them, and to use a tolerance region for such an operation instead of a single snapping location. Finally, some experiments on real-world datasets are presented, confirming how the proposed algorithm can establish the robustness of a dataset.

KEYWORDS

computational techniques, geographic information systems, spatial relationships, vector-based

1 | INTRODUCTION

Spatial data can often be described as a set of geometric objects. As sets of objects, they are characterized not only by their geometric extent but also by the spatial relations involving an object and the surrounding ones. Topological relations are a fundamental formal tool for describing the interaction among such objects, retrieving information of interest for the user, querying or processing activities, or supporting the evaluation of the data quality, in validation activities (Pelagatti, Negri, Belussi, & Migliorini, 2009; Rodríguez, Brisaboa, Meza, & Luaces, 2010).

Many abstract models have been defined in the literature for describing the semantics of topological relations between objects embedded in a Euclidean space. In particular, they have been described using the 9-intersection matrix approach (Egenhofer & Franzosa, 1991) or other axiomatic approaches (Praing & Schneider, 2009). At the same time, several computational geometry algorithms have been implemented for their evaluation. Unfortunately, these algorithms are based on a discrete vector representation of spatial data (i.e., floating-point numbers), instead of the continuous one that is theoretically required. This can cause some robustness problems that have been deeply analyzed for instance in Belussi et al. (2012, 2013, 2015), Chen (2001), Hobby (1999), and Thompson and van Oosterom (2006). The problem due to the use of a finite number representation in the algorithm implementation is made even worse in a distributed context, such as a Spatial Data Infrastructure (SDI), in which some data perturbations can occur during the exchange of spatial datasets between different systems (Belussi et al., 2012, 2013, 2015, 2016).

A geometric dataset is said to be *robust* if it can be processed by the same algorithm in different systems always producing the same result, independently of the discrete representations adopted by such systems. In Belussi et al. (2012, 2015), a set of rules has been proposed which can be applied to a vector dataset in order to increase its robustness with respect to the topological relation evaluation. However, these rules can be applied only in an implementation model in which equality between geometric primitives (i.e. vertices and segments) requires that they are bitwise identical. This model is referred to as an *identity model* in the article. Conversely, this article concentrates on another kind of model in which equality is evaluated using a tolerance value, called here a *tolerance model*. The motivation for adopting a tolerance model reflects the inability to represent each primitive with unlimited spatial accuracy. Therefore, in the tolerance model two vertices might be not identical even if they represent the same location. This requires the introduction of a new equality test, so that all vertices within a given distance, depending on the tolerance value, are considered as the same one.

A preliminary study about the robustness in a tolerance model has been done in Belussi et al. (2013), where the authors define a set of robustness rules and discuss the problems of existing algorithms in establishing or restoring the robustness of a dataset in a tolerance model. The authors concluded by highlighting the need for an algorithm specifically tailored for the tolerance model, giving some hints about its characteristics.

The aim of this article is to propose an algorithm, called *Tolerance-based Snap Rounding* (TSR), for establishing or restoring the robustness of a spatial dataset in a tolerance model. This algorithm is inspired by the ones of the Snap Rounding (SR) family (Belussi et al., 2016; Halperin & Packer, 2002; Hobby, 1999; Packer, 2008) but tries to overcome their limitations when applied in a tolerance context. In particular, TSR tries not to collapse points into a single location and is based on the notion of cluster. At the end of the article an evaluation of the algorithm effectiveness, when applied on a real dataset, is also presented.

The remainder of the article is organized as follows: Section 2 illustrates a situation which exemplifies the robustness problem that is considered in this article. Section 3 summarizes some related results presented in the literature. Section 4 formalizes the tolerance vector model that is adopted in this article for dealing with geometric objects, and the characteristics of the addressed robustness problem. Section 5 describes in detail the TSR algorithm, while Section 6 presents some experimental results collected with respect to a real-world case. Finally, Section 7 summarizes the results and proposes some future work.

2 | MOTIVATING EXAMPLE

Let us consider the spatial dataset illustrated in Figure 1 containing some instances of two feature types: road links (linestrings) and road nodes (points). In the figure a red circle highlights a situation in which a road link touches another

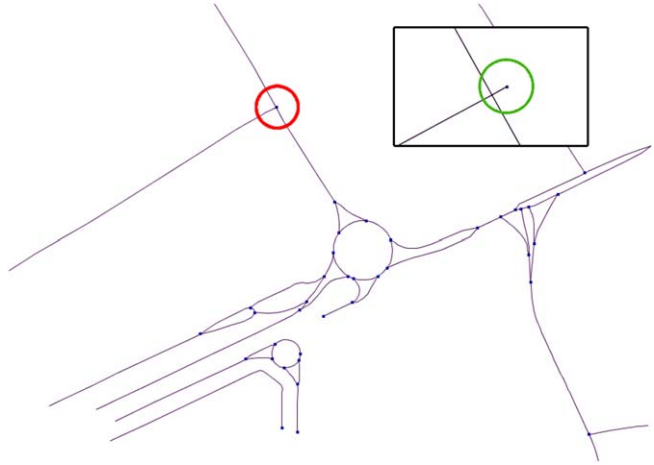

FIGURE 1 Example of dataset S_1 in which topological relations are evaluated using a tolerance approach

one. The zoom in the box on the right reveals that such a relation (*touches*) is valid only by considering a given tolerance value t which is represented by the green circle centered in the link endpoint.

Given such a situation, let us assume that the dataset is transferred from a system S_1 to another system S_2 introducing a perturbation p which moves the link endpoint away from the other touched link. If the distance between this endpoint and the other link becomes greater than t, in S_2 the relation between the two links will be evaluated as a *crosses* relation instead of a *touches*.

This data perturbation is due to an exchange between different systems and can be determined starting from the performed conversions between different formats and precisions of number representation. For instance, the GML language, Open Geospatial Consortium (OGC; 2012), an OGC and ISO standard for the exchange of spatial data, adopts a decimal encoding of coordinates represented as character strings, and the conversion from and to the floating point representation adopted by current GIS or spatial database systems can introduce perturbations. Moreover, in order to reduce the size of datasets, the number of decimal positions in the decimal representation of coordinates might also be reduced by a naïve user with respect to the one that would be required in order to keep the original precision.

The aim of this article is to propose an algorithm which makes a dataset robust with respect to the evaluation of topological relations among its geometric objects. Section 4 will clarify what it means to establish the robustness of a non-ambiguous dataset in a tolerance vector model, or restore its robustness after a perturbation occurred during a system transfer.

In the motivating example proposed here, the TSR algorithm will be applied both at the beginning on system S_1 to make the original dataset robust, and after any transfer to other systems, such as on system S_2. Indeed, after a perturbation the dataset could lose robustness, so that subsequent perturbations may lead to a loss of the original topological content and neither the topology, nor the robustness could be recovered.

3 | RELATED WORK

Geometric algorithms are usually defined under the simplified assumption that computations are performed with an infinite-precision arithmetic which cannot be actually provided by the adopted computer representation. This assumption raises great difficulties in ensuring robustness. In recent years several techniques have been proposed in order to overcome these issues. For instance, the Exact Geometric Computation model (Chen, 2001) provides a method for making the evaluation of geometric algorithms robust. This can be achieved by representing the underlying mathematical objects in an *exact* manner through the use of algebraic numbers which allow computations to be performed without errors. By definition, an algebraic number is the root of an univariate polynomial with integer coefficients. For instance, the number $\sqrt{5}$ has no finite representation, but it can be represented exactly as the pair $(X^2-5, [1, 4])$,

interpreted as the root of the polynomial X^2-5 lying in the interval $[1,4]$. These techniques have made much progress so that for certain problems the introduced performance penalty is acceptable. However, when the computation is performed on curved objects (they can be approximated in linear geometry thought linestrings having many vertices) or in three-dimensional space, the overhead is still large.

An alternative approach has been proposed, called Controlled Perturbation (CP) (Halperin, 2010), which belongs to the family of Finite-Precision Approximation techniques. This method proceeds by perturbing the input slightly but in a controlled manner such that all predicates used by the algorithm are guaranteed to be evaluated correctly with floating-point arithmetic of a given precision, and the degeneracies are removed. The algorithms of the Snap Rounding approach (Halperin & Packer, 2002; Hobby, 1999; Packer, 2008) belong to the same family of Finite-Precision Approximation techniques where you find CP. However, they are mainly based on the application of some rounding algorithms that convert an arbitrary-precision arrangement of segments into a fixed-precision representation. In Belussi et al. (2016) the authors propose a variant of the SR algorithm which tries to preserve the original relation between objects by introducing an additional operation that spreads or snaps vertices based on the original characteristics of the arrangement. However, all these solutions clash with the fundamental idea of the tolerance model, since all equal primitives are collapsed into the same one and the tolerance notion become useless.

In the geographical field, several robustness rules have been proposed in order to solve the mentioned problems, and they are to some extent applied by real systems. The most important one is based on the identification of common geometric primitives between different objects. These common primitives can be either stored once and referred to by the objects (topological structures; Egenhofer, Frank, & Jackson, 1990; Theobald, 2001) or repeated identically in all objects which share them. A GIS topology is a set of rules that models how points, lines and polygons share coincident geometries, for instance imposing that adjacent features will have a portion of common boundary. A topological data model manages spatial relationships by representing spatial objects as an underlying graph of topological primitives: nodes, faces and edges.

The identification of coincident geometries can be performed in two distinct ways: by requiring the bitwise equality between coordinates (identity model) or by considering a tolerance value during the tests (tolerance model). Some available GIS tools, such as PostGIS (OSGeo, 2014) and the JTS Topology Suite (Vivid Solutions, 2014), use the first model for implementing topological relations, while other ones, such as Esri ArcGIS (Esri, 2014), apply the second approach for topology construction. In particular, the term cluster tolerance is used to identify the distance range below which all vertices are considered identical or a vertex is considered to belong to a segment. Notice that in ArcGIS the clustering step implies the replacement of coincident vertices with a single representative point, determined considering the position of the original vertices and an assigned weight (Esri, 2010). Conversely, the tolerance model considered in this article does not include a replacement of original vertices, but only the definition of equality clusters.

In Pullar (1993) the author examines the flaws in applying a tolerance paradigm when performing a spatial overlay operation and describes the conditions needed to safely evaluate point coincidence. For this purpose, he defines the concept of *epsilon point* as a tuple (x, y, ε) representing a *xy*-coordinate and a tolerance radius ε. Since this work deals with the integration of multiple thematic layers into a single one, it deals with several different ε tolerance values. Conversely, in the case considered in this article, it is sufficient to assume a single tolerance value t for all points, since we suppose to consider datasets with a homogeneous quality and subject to the same type of perturbation.

4 | PROBLEM FORMALIZATION

This section formalizes the tolerance vector model considered in this article and the characteristics of the addressed robustness problem. In particular, it defines the topological relations of interest in the presence of tolerance and it summarizes the required robustness rules. Some definitions and propositions are taken from a previous work (Belussi et al., 2013) but are reported and sometimes enriched here in order to better formalize the problem.

4.1 | Tolerance vector model

The tolerance model is characterized by a reference tolerance threshold t used to define the relation existing between two primitives. As a general idea, the equality between two objects does not require that they are bitwise identical. The tolerance threshold is chosen according to the application context and usually is correlated to the absolute positional accuracy of the primitives collected in the considered datasets.

This article considers only datasets which can be described through two types of geometric primitives: *point* and *segment*, embedded into a 2D Euclidean space. Given such space, the set of all points is denoted as $\mathcal{P}$, while the set of all segments is denoted as $\mathcal{S}$.

Definition 1. (Vertex). A vertex $v \in \mathbb{R}^2$ is a pair of real numbers representing a 2D coordinate $v=(x,y)$, where $x, y \in \mathbb{R}$.

Definition 2. (Point). A point $p \in \mathcal{P}$ is a geometric object represented by a single vertex $v \in \mathbb{R}^2$.

Definition 3. (Segment). A segment $s \in \mathcal{S}$ is a geometric object obtained by the linear interpolation between a pair of vertices (v_1, v_2), where v_1 is called start node and v_2 end node.

Given a segment $s \in \mathcal{S}$, its start and end points are denoted by the functions *s.start* and *s.end*, respectively. In the following, the set of all points and segment end-points contained in a dataset D are generically called vertices of D and denoted as $D.\mathcal{V}$. Similarly, the set of points (segments) of D are denoted $D.\mathcal{P}$ ($D.\mathcal{S}$). The dataset D can be used for the representation of geometric objects that can be obtained through a composition of geometric primitives (points and/or segments). In other words, we can represent in D a set of *Linestring* and/or a set of *Polygon* objects (geometric types of the OGC Standards), since they can be represented as a collection of segments satisfying certain properties. This observation is crucial in order to determine the topological properties of primitives that must become robust for preserving the relations among the geometric objects contained in D.

As stated in Belussi et al. (2013), the topological relation existing between two points $p_1, p_2 \in \mathcal{P}$, or between a point $p \in \mathcal{P}$ and a segment $s \in \mathcal{S}$, or between two segments $s_1, s_2 \in \mathcal{S}$ can be determined in the tolerance model using the three predicates in Definition 5, where the following distance function is used.

Definition 4. (Distance). The function $distance(g_1, g_2)$ computes the distance between two geometric primitives g_1, g_2 as follows[1]:

$$distance(g_1, g_2) = min(\{EuclideanDistance(p_1, p_2) | p_1 \in g_1, p_2 \in g_2\}) \quad (1)$$

Definition 5. (Critical predicates). The tolerance model is characterized by the following critical predicates. In the definitions t is the tolerance threshold and the distance function is the one above defined:

- *Given* $p_1, p_2 \in \mathcal{P}$, $equal(p_1, p_2) \iff distance(p_1, p_2) \leq t$.
- *Given* $p \in \mathcal{P}$ *and* $s \in \mathcal{S}$, $belongsTo(p, s) \iff distance(p, s) \leq t$.
- *Given* $p \in \mathcal{P}$ *and* $s \in \mathcal{S}$, $leftOf(p, s)$ *(or* $rightOf(p, s)$*)* $\iff$ v lies in the left region[2] induced by s (or in the right region induced by s) $\wedge\, distance(p, s) > t$.

Notice that the condition $belongsTo(p, s)$ does not exclude that p is equal to one end-point of s. In other words, given $p \in \mathcal{P}$, if $\exists s \in \mathcal{S} : equal(p, s.start) \vee equal(p, s.end)$, then $belongsTo(p, s)$. Moreover, in order to simplify the notation, the following derived critical predicates are introduced.

Definition 6. (Derived critical predicates). The following derived critical predicates can be useful for testing topological relations:

- *Given* $p \in \mathcal{P}$ *and* $s \in \mathcal{S}$, $separate(p, s) \iff \neg leftOf(p, s) \wedge \neg rightOf(p, s) \wedge \neg belongsTo(p, s)$.
- *Given* $s_1, s_2 \in \mathcal{S}$, $intersect(s_1, s_2) \iff \exists p \in s_1, belongsTo(p, s_2)$, where $p \in s_1$ denotes any point inside the segment s_1. It is trivial to prove that the symmetric definition is equivalent.

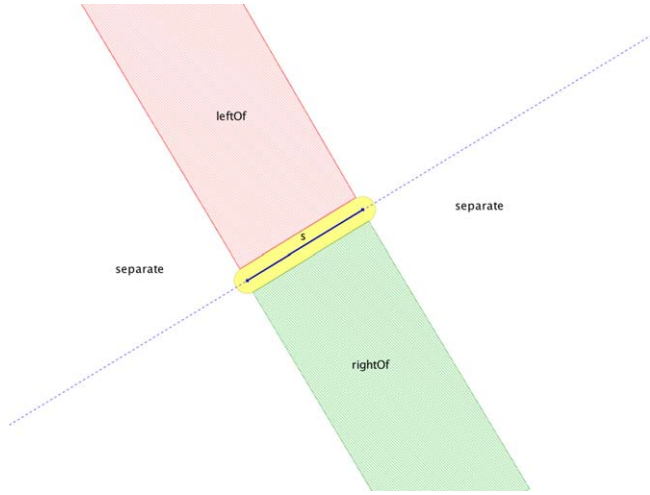

FIGURE 2 Regions defined by a segment s for the predicates $leftOf(p,s)$, $rightOf(p,s)$ and $separate(p,s)$

Figure 2 shows the regions defined by a segment s in which the different predicates are true.

The predicates in Definitions 5 and 6 are said to be critical because their evaluation can produce different results in different systems. In particular, the computation of the distance between two points or a point and a segment, when it is very close to the tolerance value t, can produce different results in different systems, thus different evaluations of the predicates. Such a problem, which is the core of the robustness issue, is formalized and treated in more detail in Section 4.3.

Proposition 1. In a tolerance vector model characterized by a tolerance threshold t, the topological relations of the 9-intersection model (Egenhofer & Franzosa, 1991) between two points, a point and a segment and two segments can be redefined using only the predicates introduced in Definition 5 and the predicate $intersect(s_1,s_2)$ of Definition 6.

Proof. Table 1 illustrates for each topological relation and for each combination of geometric primitives the expression that can be used for testing the relation on the geometries representing the primitives. Each expression uses only the predicates introduced in Definition 5 and the predicate $intersect(s_1,s_2)$ of Definition 6.

From Table 1 it is clear that the predicates $equal(p_1,p_2)$ and $belongsTo(p,s)$ have to be maintained in order to preserve the topological relations in a dataset D. The following proposition shows the need to preserve also $leftOf(p,s)$ and $rightOf(p,s)$ predicates.

Proposition 2. In a tolerance vector model characterized by a tolerance threshold t, the predicates $leftOf(p,s)$ and $rightOf(p,s)$ have to be preserved in order to preserve the topological relations between two segments and the topological relations between a point/segment and a polygon. In particular, the transitions $leftOf(p,s) \rightarrow rightOf(p,s)$, the opposite one and the transitions from $leftOf(p,s)$ or $rightOf(p,s)$ towards $belongsTo(p,s)$ have to be avoided, while the transitions $leftOf(p,s) \rightarrow separate(p,s)$, $rightOf(p,s) \rightarrow separate(p,s)$ and the opposite ones can be admissible.

Proof. When p is the end point of a segment s_1, changing its relative position with respect to another segment s_2 (i.e., transitions $leftOf(p,s_2) \rightarrow rightOf(p,s_2)$ or the opposite one) can change the relation between s_1 and s_2; for instance, a $disjoint(s_1,s_2)$ can change into $crosses(s_1,s_2)$ or vice versa. Considering a point p and a polygon pg, changing the relative position of p with respect to a segment s of the polygon boundary can change the relation between p and pg; i.e., a *disjoint*(p, pg) can change into *in*(p, pg) or vice versa. The same can happen between a segment s and a polygon pg; for instance a *disjoint*(s, pg) can change into *cross*(s, pg) or vice versa. Finally, the transition $leftOf(p,s)$ (or $rightOf(p,s)$) $\rightarrow$ *belongsTo*(p,s) can change the relation between a point p and a segment s from *disjoint*(p, s) to *in*(p, s) (or *touches*(p, s)); the same holds for the relation between a point and a polygon.

Regarding the transitions $leftOf(p,s) \rightarrow separate(p,s)$, $rightOf(p,s) \rightarrow separate(p,s)$ and the opposite ones, they are admissible, provided that the point p remains in the same half-plane defined by the segment s. In this way, p will

TABLE 1 Possible topological relations between two points $p_1, p_2 \in \mathcal{P}$, or a point $p \in \mathcal{P}$ and a segment $s=(v_1, v_2) \in \mathcal{S}$, or two segments $s_1=(v_1, v_2), s_2=(u_1, u_2) \in \mathcal{S}$. The table assumes that the segment vertices are ordered: $\forall s=(v_1, v_2) \in \mathcal{S}, v_1.x<v_2.x \vee (v_1.x=v_2.x \wedge v_1.y \leq v_2.y)$

	$p_1, p_2 \in \mathcal{P}$	$p \in \mathcal{P}, (v_1, v_2) \in \mathcal{S}$	$(v_1, v_2), (u_1, u_2) \in \mathcal{S}$
dj	$\neg equal(p_1, p_2)$	$\neg belongsTo(p, s)$	$\neg intersect(s_1, s_2)$
in	–	$belongsTo(p, s) \wedge$	$belongsTo(v_1, s_2) \wedge$
		$\neg equal(p, v_1) \wedge$	$belongsTo(v_2, s_2) \wedge$
		$\neg equal(p, v_2)$	$\neg(equal(v_1, u_1) \vee$
			$equal(v_2, u_2))$
ct	–	–	$in(s_2, s_1)$
eq	$equal(p_1, p_2)$	–	$equal(v_1, u_1) \wedge equal(v_2, u_2)$
ov	–	–	$(in(v_1, s_2) \vee in(u_2, s_1)) \wedge$
			$(in(u_1, s_1) \vee in(v_2, s_2)) \wedge$
			$\neg eq(s_1, s_2)$
cr	–	–	$intersect(s_1, s_2) \wedge$
			$\neg belongsTo(v_1, s_2) \wedge$
			$\neg belongsTo(v_2, s_2) \wedge$
			$\neg belongsTo(u_1, s_1) \wedge$
			$\neg belongsTo(u_2, s_1) \wedge$
			$\neg ov(s_1, s_2)$
tc	–	$equal(p, v_1) \vee$	$belongsTo(v_1, s_2) \vee$
		$equal(p, v_2)$	$belongsTo(v_2, s_2) \vee$
			$belongsTo(u_1, s_1) \vee$
			$belongsTo(u_2, s_1) \wedge$
			$\neg in(s_1, s_2) \wedge$
			$\neg ov(s_1, s_2) \wedge$
			$\neg cr(s_1, s_2) \wedge$
			$\neg eq(s_1, s_2)$

remain outside/inside the polygon or the segment, whose end point is *p*, or it will remain in disjoint/cross with the other segment/polygon. However, for the relation preservation between a point/segment and a polygon, this condition is not sufficient; indeed, in order to obtain the preservation of the relation, the whole set of segments s_i composing the polygon boundary must preserve the predicates $leftOf(p, s_i)$ or $rightOf(p, s_i)$ admitting only transitions towards $separate(p, s_i)$ but remaining in the same half-plane defined by s_i.

Notice that in order to ensure the transitivity of the equality relation, some form of clustering is needed among vertices. In other words, this property can be guaranteed only by subdividing vertices into clusters such that all vertices belonging to the same cluster are considered equals, while vertices assigned to different clusters are considered not equal. The problem of determining point coincidence inside a tolerance vector model is extensively treated in the following section.

However, the relation definition given in the Table 1 is independent of the clustering. Section 5 will discuss how it is possible to maintain such independence.

Definition 7. (Topological consistent dataset). In a tolerance model, a dataset *D* is said to be topological consistent if and only if $\forall p_1, p_2 \in D.\mathcal{P}, \forall s \in D.\mathcal{S}, equal(p_1, p_2) \wedge cp(p_1, s) \Rightarrow cp(p_2, s)$, where *cp* denotes one of the following critical predicates: $belongsTo(p, s)$, $leftOf(p, s) \vee separate(p, s)$ and $rightOf(p, s) \vee separate(p, s)$.

4.2 | Point coincidence in tolerance vector model

In a tolerance model, when the distance between two objects is below a certain tolerance value, they are classified as coincident. However, a naïve application of the tolerance paradigm can cause some problems, due to the transitivity of the equivalence relation. Simply assuming that if $p_1 = p_2$ and $p_2 = p_3$, implies $p_1 = p_3$ can lead in the worst case scenario to collapse all objects into a single point (Milenkovic, 1988).

Two geometrical criteria are imposed in order to define the coincidence between points in a tolerance model: *creep*, no point is moved more than t, and *shrounding*, no points are left within a distance t (Pullar, 1993). In other words, in order to ensure the transitivity of the equality relation, some form of clustering is needed among vertices.

The best way to solve the coincidence relation between points in a tolerance model, is to use the point clustering problems in order to identify a set of equivalence classes. The clustering problem is defined as the grouping of similar objects, such that objects inside the same cluster are similar to each other, while objects from different clusters are dissimilar. In our case, the objects are points and the similarity between them is defined using the Euclidean distance measure.

The clustering is also chosen to maximize the separation criteria by stipulating that coincident points minimize some measure of dissimilarity. Minimizing the dissimilarity is interpreted as minimizing the sum of the squared length from cluster points to their center. This is called a *sum of squared error clustering* (Duda & Hart, 1973) and is closely related to finding the medians of a graph (i.e. *p-median* problem) (Christofides, 1975).

Definition 8. (Euclidean *p*-median problem). Given a set $X=\{p_1, p_2, \ldots, p_n\}$ of *n points (x, y)*, find a set X' of m points $\{p'_1, p'_2, \ldots, p'_m\}$ so as to minimize the expression:

$$\sum_{i=1}^{n} \min_{1\le r\le m} \{\text{dist}(p'_r, p_i)\} \tag{2}$$

where dist() is the Euclidean distance between points and the points in X' represents the cluster centers. In other words, the problem wishes to minimize the sum of the radii that enclose points of X by circles located at centers of X'.

The Euclidean p-median problem can be re-stated using the concept of distance threshold as follows (Pullar, 1993).

Definition 9. (Distance constrained Euclidean *p*-median problem). Given a set $X=\{p_1, p_2, \ldots, p_n\}$ of n points with a distance threshold t, the distance constrained Euclidean p-median problem finds a set $X'=\{p_1, p_2, \ldots, p_m\}$ where $m \le n$ so as to minimize the expressions:

$$\sum_{i=1}^{n} \min_{1\le r\le m} \{\text{dist}(p'_r, p_i)\} < t \tag{3}$$

and

$$\forall r, s \in \{1, \ldots, m\} \text{dist}(p'_r, p'_s) > 2t \tag{4}$$

The first equation minimizes the sum of the radii that enclose points of X by circles located at cluster centers of X', requiring that the distance between each point of X and its cluster center is less than the tolerance threshold. Conversely, the second equation additionally stipulates that a minimum separation is maintained between cluster centers.

Given the solution to the constrained clustering, any two points p_i, p_j clustered together are considered part of the same equivalence class based on the relation p_i is *coincident-to* p_j. Since coincidence is an equivalence relation, then by definition the relation is reflexive, symmetric and transitive. Existing clustering algorithms will be briefly discussed in Section 5.1.

4.3 | Non-ambiguous datasets

The predicates in Definitions 5 and 6 are said to be critical, because their evaluation can produce different results in different systems. This problem is due to two main factors: the discrete representation adopted by such systems and

the perturbations that can be introduced during the exchange of data among different systems, such as an SDI context.

With regards to the first aspect, the existence of robustness problems in the execution of computational geometry algorithms which use finite numbers (e.g. floating point) for the representation of coordinates in the Euclidean space, instead of the real numbers that are theoretically required, is well known and is captured by the following definition.

Definition 10. (Numerical weakness). The numerical weakness of a set of implementations of a given algorithm on different machines is the largest distance between two points or a point and a segment such that the evaluation of the algorithm can produce different results (Belussi et al., 2013).

For example, we can consider different implementations of the $leftOf(p,s)$ predicate and conclude that the numerical weakness of this set of implementations is 10^{-12} since when the distance between p and s is greater than this threshold the results produced by all the implementations are all the same.

Many techniques have been proposed in the literature (Chen, 2001; Halperin & Packer, 2002; Hobby, 1999) for reducing or eliminating the numerical weakness in algorithm implementation. Therefore, in a given context, it is possible to assume that the numerical weakness is less than a given value w.

Definition 11. (Topological interpretation). The topological interpretation of a geometric dataset is the evaluation of all possible critical predicates between the primitives of a given dataset D (Belussi et al., 2013).

Recall that, given $p_1, p_2 \in D.\mathcal{P}$ and $s \in D.\mathcal{S}$ the critical predicates are: $equal(p_1,p_2)$, $belongsTo(p_1,s)$, $leftOf(p_1,s) \vee separate(p_1,s)$ and $rightOf(p_1,s) \vee separate(p_1,s)$.

Definition 12. (Topologically non-ambiguous dataset). A dataset D is topologically non-ambiguous if and only if different predicate implementations on different machines always produce the same topological interpretation on D.

Proposition 3. Given a dataset D and a context characterized by a numerical weakness w, if D satisfies the following rules, then it is non-ambiguous in the tolerance model (Belussi et al., 2013):

1. $\forall p_1, p_2 \in D.\mathcal{P}\ (equal(p_1,p_2) \Rightarrow distance(p_1,p_2) \leq t-w)$
2. $\forall p_1, p_2 \in D.\mathcal{P}\ (\neg equal(p_1,p_2) \Rightarrow distance(p_1,p_2) > t+w)$
3. $\forall p \in D.\mathcal{P}, \forall s \in D.\mathcal{S}\ (belongsTo(p,s) \Rightarrow distance(p,s) \leq t-w)$
4. $\forall p \in D.\mathcal{P}, \forall s \in D.\mathcal{S}\ (\neg belongsTo(p,s) \Rightarrow distance(p,s) > t+w)$

Proof. Suppose that D satisfies all rules. Since D satisfies Rule 1, if two vertices p_1, p_2 are equal, than it holds that $distance(p_1,p_2) < t-w$, and this guarantees that in a context of numerical weakness w, $distance(p_1,p_2) < t$ is always evaluated to true by all implementation of the predicate *equal*. The satisfaction of Rule 2 by D allows us to apply a similar reasoning to the case in which p_1, p_2 are not equal, thus proving that the $equal(p_1,p_2)$ predicate is always evaluated to false, thus it is non-ambiguous.

Moreover, since D satisfies Rule 3, if a vertex p belongs to a segment s, then it is true that $distance(p,s) < t-w$, and this guarantees that in a context of numerical weakness w, $distance(p,s) < t$ is always evaluated to true by all implementation of the predicate *belongsTo*. In a similar way, by exploiting the satisfaction of Rule 4, it can be proved, for the case in which p does not belong to s, that $distance(p,s) > t$ is non-ambiguous, thus concluding that the $belongsTo(p,s)$ predicate is non-ambiguous. Finally, Rule 4 also guarantees that the $leftOf(p,s) \vee separate(p,s)$ ($rightOf(p,s) \vee separate(p,s)$) predicate is non-ambiguous, indeed the condition $distance(p,s) > t+w$, when $\neg belongsTo(p,s)$, ensures that it cannot be evaluated to true in any implementation. Therefore, possibly p can only shift outside the left region induced by s, thus degenerating the *leftOf* predicate into a *separate* predicate.

This article concentrates on the second problem, related to the data perturbation that can occur during exchange activities, which is formalized in the following section.

4.4 | Robust datasets

During a data exchange in a distributed context, such as an SDI, some perturbations can be introduced in the data. This phenomenon can be described formally as follows.

Definition 13. (δ-perturbation). Given a number $\delta \in \mathbb{R}$, a δ-perturbation of a dataset *D* is a copy of *D* where each coordinate of its geometries is arbitrarily modified by an amount $\varepsilon < \delta$ (Belussi et al., 2013).

Definition 14. (δ-robust dataset). A dataset *D* is δ-robust if and only if the same topological interpretation is produced by different algorithm implementations on any δ-perturbation of *D* (Belussi et al., 2013).

The perturbations considered in this article can be arbitrarily applied to each coordinate of a geometric primitive. In particular, two kinds of perturbations can be distinguished: conservative perturbation and non-conservative perturbation. A *conservative perturbation* ensures that the equality classes induced by clustering are preserved, namely vertices that are equal before a perturbation remains equal also after the perturbation. Conversely, in a *non-conservative perturbation*, changes applied to equal vertices can produce vertices that are no longer equal. In the following, only conservative perturbations are considered.

In order to make a dataset robust with respect to conservative perturbations, the following two rules are introduced.

Definition 15. (TME rule). The tolerance model equality rule (TME) requires that the maximum distance (max_d) between two equal vertices (i.e., points or segment end-points) is less than $(t-2\delta)$: $max_d < t-2\delta$ (Belussi et al., 2013).

Notice that the coefficient 2 is needed because two primitives can move away from each other in opposite directions.

Definition 16. (TMD rule). The tolerance model disjointness rule (TMD) requires that the minimum distance (min_d) between two points, or between a point and a segment is greater than $(t+2\delta)$: $min_d > t+2\delta$ (Belussi et al., 2013).

As for the previous case, the coefficient 2 is needed, because two primitives can move close in opposite directions. In the following a pair of primitives satisfying the TMD rule are said to be *well-separated*.

Proposition 4. Given a dataset *D*, a necessary condition to satisfy TME is that $t > 2\delta$.

Proof. If $t \leq 2\delta$, rule TME requires that $max_d = 0$, since the distance between two primitives cannot be a negative value. This corresponds to a degeneration of the equality condition to the one required by the identity model: two primitives are equal if they are bitwise identical. This is not acceptable in a tolerance model, thus $t > 2\delta$ is a necessary condition to apply the TME rule.

Proposition 5. Given a dataset *D*, if it satisfies TME and TMD rules, then it is δ-robust in the tolerance model.

Proof. The proof is similar to the one presented for Proposition 3, where the rule TME preserves the $equal(v_1, v_2)$ predicate and the $belongsTo(v, s)$ predicate when they are true, while TMD rule preserves the same predicates when they are false and guarantees the δ-robustness of the $leftOf(v, s)$ ($rightOf(v, s)$) predicate.

Therefore, given Propositions 4 and 5, the tolerance model with tolerance threshold *t* can guarantee the δ-robustness of a dataset *D* only for values of δ that satisfy the condition $\delta < t/2$.

5 | ALGORITHM FOR ROBUST DATASETS

This section presents an algorithm for establishing or restoring the robustness of a dataset in a tolerance-based vector model. This algorithm is inspired by the ones of the Snap Rounding (SR) family (Halperin & Packer, 2002; Hobby, 1999; Packer, 2008). However, while they do not consider a tolerance value but aim to establish an identity between equal primitives, the algorithm proposed in this article tries to introduce the concept of tolerance during the performed operations. For this reason, it is called *Tolerance-based Snap Rounding* (TSR). Another difference between SR algorithms and TSR is that while the first ones do not necessarily preserve the original relations between primitives, since they collapse

all geometries that are not well-separated, TSR tries to preserve the original relation as much as possible, similarly of what Belussi et al. (2016) did in the identity model.

TSR uses a terminology similar to the one of SR which is summarized by the following definition. In particular, in a tolerance model the concept of arrangement is weakened, since equality is based on the tolerance concept rather than on the identity one. Moreover, the dataset considered in this article contains points as well as segments.

Definition 17. (Weak-arrangement). Let D a dataset composed of a collection of segments and a collection of points in the plane. The weak-arrangement $\mathcal{A}$ of D is the decomposition of the plane into vertices, edges and faces induced by the geometries:

- A vertex of $\mathcal{A}$ is a segment end-point, a tolerance-intersection between two segments, or a point. Notice that segment intersections are computed with respect to the given tolerance value. The set of vertices in an arrangement are denoted as $\mathcal{A}.\mathcal{V}$.
- An edge of $\mathcal{A}$ is a connected set of points belonging to one segment of $D.\mathcal{S}$. Each edge starts and ends at vertices of the arrangement. The set of edges in $\mathcal{A}$ are denoted as $\mathcal{A}.\mathcal{E}$.
- A face of $\mathcal{A}$ is a subset of points of the plane not contained in any segment of $D.\mathcal{S}$.

The following subsections illustrate in detail each phase of the TSR algorithm.

5.1 | Clustering techniques

Given a dataset D containing a set of points and a set of segments, a preliminary clustering phase has to be applied on its vertices $\mathcal{V}$, in order to identify those that are equal with respect to the chosen tolerance t.

Several different algorithms have been defined in the literature for clustering spatial data (Chandra, 2011). The criteria for deciding upon a particular one depend on the specific application, since many of them have been defined in the context of spatial data mining.

In general, clustering techniques can be classified into three main categories: (1) partitional clustering; (2) hierarchical clustering; and (3) locality-based clustering. In partitional clustering, objects are subdivided into clusters such that objects in a cluster are more similar to each other than they are to objects in other clusters. These algorithms use the concept of k-medoids, representative objects whose average dissimilarity to all the objects in the cluster is minimal, e.g. CLARANS (Ng & Han, 1994), or the concept of k-means, namely the definition of some objective function which represents the belonging to a particular cluster e.g. DENCLUE (Hinneburg & Keim, 1998).

The *hierarchical clustering* algorithms are based on the definition of some hierarchical data structures that are used to build clusters through a sequence of incremental steps. An example of this kind of algorithms is BIRCH (Zhang, Ramakrishnan, & Livny, 1996) which uses an hierarchical data structure called CF-tree (Clustering-Feature tree) to dynamically and incrementally cluster the data points in a way similar to the construction of a spatial index.

Finally, *locality-based clustering* groups objects based on their local relationships. For instance, *density-based* algorithms rely on the density of objects, e.g., DBSCAN (Ester, Kriegel, Sander, & Xu, 1996), while random-distribution algorithms assume that points inside the cluster are uniformly distributed, e.g. DBCLASD (Xu, Ester, Kriegel, & Sander, 1998).

This article considers the family of partitional clustering algorithm known as *k-means*, where the objective function is defined in terms of the Euclidean distance as reported in Definition 9. In particular, the *k-means* algorithm (MacQueen, 1967) is based on a general iterative scheme for finding the local optimum minimum solution: it starts by locating each point in a cluster by itself, and then repeatedly combines two "nearest" clusters into one. There are several different versions of the *k*-means algorithm; one of these is Lloyd's algorithm (Kanungo et al., 2002), which uses scalar data and assumes that each cluster is represented by its *centroid*, namely the average across all points in the cluster.

The *k-means* algorithm initializes the *k* clusters by picking one point per cluster, for instance by randomly choosing *k* points, or by randomly selecting the first one and then choosing the other ones as far as possible from the selected.

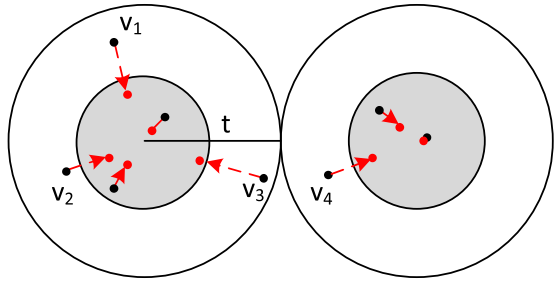

FIGURE 3 Example of application of the GATHERING() procedure to two adjacent clusters. Vertices in the same cluster will be moved inside the gray region, called cluster *kernel*, in a position identified by the red points, which is proportional to their original distance from the cluster center

Given any set X' of k cluster centers, for each center $p' \in X'$, let $V(p')$ denotes its neighborhood, that is the set of points for which p' is the nearest neighbor. At each algorithm iteration, the points are placed in the cluster whose current centroid is the nearest and the point p' is moved to the centroid of $V(p')$ (E-step). Then $V(p')$ is updated by recomputing the distance from each point to its nearest center (M-step). The procedure is executed until the value of the objective function remains the same for the next iteration.

Lloyd's algorithm converges rapidly but speed often comes at the cost of quality. In general, the quality of the result depends upon the initial centroids (Yuan, Meng, Zhang, & Dong, 2004). For instance, the choose of the initial centroids have to consider the distribution of the points.

Many other heuristics have been defined for the implementation of the k-means algorithm which try to combine the speed of Lloyd's with a greater quality of the solution. However, the TSR algorithm proposed in this article is orthogonal to the specific implementation of the clustering method, so it is left to the reader to analyze the various alternatives (Chandra, 2011). This article simply assumes that a preliminary clustering phase has been performed on D producing a set of clusters $\mathcal{C}$. Moreover, in order to preserve the tolerance-intersection existing between two segments, an additional vertex has to be added inside each segment, which represent such an intersection point. These two vertices will be placed inside the corresponding segment, at a distance less than or equal to t from each other, in a way they will belong to the same cluster (i.e. they are considered as the same point).

5.2 | Vertex gathering

In a tolerance model, in order to ensure the transitivity property of the equality relation, two different strategies can be applied: (1) use the notion of cluster in the definition of the topological relations; or (2) ensure that the primitives belonging to the same cluster are at a distance less than or equal to t, while primitives belonging to different clusters are at a distance greater than t. This article considers the second option because it allows one to maintain the definition of topological relations independent of the notion of cluster and does not require storing the found clusters. Therefore, given the set of clusters $\mathcal{C}$, an additional phase is necessary which brings together the vertices of a given cluster and turns away vertices that belong to different clusters.

Figure 3 illustrates two adjacent clusters containing several vertices. Each cluster is represented by a circle with ray t. Notice that vertices v_3 and v_4 are at a distance less than t but the clustering process has determined that they belong to two different clusters, because their nearest centroids are different. In order to ensure that any vertex of the first cluster is placed at a distance greater than t from any vertex of the second cluster, and at distance less than or equal to t from any vertex of the same cluster, they have to be moved inside the gray region which is a circle with ray $t/2$ centered in the middle of the cluster. This region will be called the cluster *kernel*, while the overall region with ray t will be called the cluster *support*. The movement will be proportional to the original distance of the vertex from the pixel center: i.e. the more a vertex is close to the support boundary, the more will be moved towards the kernel, so that at the end of this adjustment it is guaranteed that all vertices are inside the cluster kernel.

The choice to move vertices proportionally towards the cluster center, instead of just snap to the center, has been made in order to preserve the nature of the tolerance model. A snapping technique similar to SR will instead collapse all equal points into a single one, transforming a tolerance model into an identity one. Moreover, the proposed algorithm tries to reduce the difference between the original dataset and its robust transformation.

This operation is formalized by the procedure GATHERING() in Algorithm 1, in which function APPROACHING(), illustrated in Algorithm 2, determines the new vertex position. In order to make the dataset robust, as well as non-ambiguous, it is sufficient to build a kernel using a ray $t/2-\delta$; function APPROACHING() shifts vertices supposing to build a kernel with this ray. Finally, in Algorithm 1 function *replace-vertex*() applied to a weak-arrangement $\mathcal{A}$ substitutes all occurrences of a vertex v with another vertex v', included when it is used in edge definitions.

Algorithm 1. GATHERING operation performed on vertices of a weak-arrangement $\mathcal{A}$, given a set of clusters $\mathcal{C}$, a tolerance threshold t and a desired robustness level δ. It produces as output a modified arrangement $\mathcal{A}^*$.

```
1: procedure GATHERING (C, A, t, δ)
2:     A* ← A
3:     for c ∈ C do
4:         for v ∈ c do
5:             v' ← APPROACHING(v, c, t, δ)
6:             A*.replace-vertex(v, v')
7:         end for
8:     end for
9:     return A*
10: end procedure
```

Algorithm 2. APPROACHING operation performed on a vertex v belonging to a cluster c, given a tolerance threshold t and a desired robustness level δ. It produces as output an updated vertex v' which lies in the kernel of c.

```
1: procedure APPROACHING(v, c, t, δ)
2:     v' ← v
3:     v'.x = c.kernel.x + (v.x − c.kernel.x) · (t/2 − δ)/t
4:     v'.y = c.kernel.y + (v.y − c.kernel.y) · (t/2 − δ)/t
5:     return v'
6: end procedure
```

Figure 3 shows an example of application of the GATHERING() procedure to two adjacent clusters: vertices are moved towards the cluster center, so they are all contained in the cluster kernel, included vertices v_1, v_2, v_3 and v_4 which are originally outside it.

After the application of the GATHERING() procedure the following conditions are satisfied and the relations among vertices can be considered non-ambiguous and robust:

- $\forall v_1, v_2 \in \mathcal{V}, \mathit{equal}(v_1, v_2) \Rightarrow \mathit{distance}(v_1, v_2) \leq t-2\delta$.
- $\forall v_1, v_2 \in \mathcal{V}, \neg \mathit{equal}(v_1, v_2) \Rightarrow \mathit{distance}(v_1, v_2) > t+2\delta$.

This gathering operation allows definition of relations between vertices which do not depend on the clustering algorithm. In other words, operations on geometric primitives are defined considering only the tolerance threshold t and not the cluster to which each involved vertex belongs.

Proposition 6. Given a weak-arrangement $\mathcal{A}$ for a dataset D, the application of Algorithm 1 produces a modified arrangement $\mathcal{A}^*$, such that:

$$\forall v_1, v_2 \in \mathcal{A}^*.\mathcal{V} : equal(v_1, v_2) \vee v_1, v_2 \text{ are well}-\text{separated}$$

and the relation equal is robust.

Proof. After the application of Algorithm 1 all vertices in $\mathcal{A}^*.\mathcal{V}$ are contained inside the cluster kernels. Indeed, the vertices inside the kernel of a cluster c centered in $(c.kernel.x, c.kernel.y)$ satisfies the condition $X^2+Y^2<(t/2-\delta)^2$, where X and Y are the coordinates of the vertex with respect to the kernel center. After the application of the procedure APPROACHING() to a vertex v such coordinates are $(\Delta_x \cdot (t/2-\delta)/t, \Delta_y \cdot (t/2-\delta)/t)$[3]; by substituting them in the condition above, we obtain $\Delta_x^2+\Delta_y^2<t^2$, which is always satisfied in the weak-arrangement $\mathcal{A}$ considering two vertices belonging to the same cluster k. Therefore, given two vertices $v_1, v_2 \in \mathcal{V}$, if v_1, v_2 belongs to the same kernel k, then their distance min_d is less than $2(t/2-\delta)$, thus the relation $equal(v_1, v_2)$ is non-ambiguous, since min_d is less than t. Moreover, $equal(v_1, v_2)$ is robust, since min_d is less than $(t-2\delta)$. Conversely, if v_1, v_2 belongs to two different kernels k_1 and k_2, then their distance is greater than $2(t/2+\delta)$, thus they are well-separated.

5.3 | Segment restore

The GATHERING() procedure described in the previous section makes it possible to obtain a dataset whose vertices are equal or are well separated and ensures that these relations are non-ambiguous and robust. Also, in order to make the dataset robust, each vertex and each non-incident segment must be well separated. Therefore, another phase is necessary which approaches or separates a segment from a vertex, in order to make the *belongsTo* relation non-ambiguous and robust as well.

Algorithm 3. RESTORE procedure applied to a weak-arrangement $\mathcal{A}$ and a set of clusters $\mathcal{C}$. It produces a modified weak-arrangement $\mathcal{A}^*$ which is well separated.

```
1: procedure RESTORE (A, C)
2:     A* ← A
3:     K ← Critical-Clusters(A*, C)
4:     while K ≠ ∅ do
5:         for c ∈ K do
6:             for e ∈ c.inter-edges do
7:                 v ← Choose-Vertex(c, e, A)
8:                 {e1, e2} ← e.split-edge(e, v)
9:                 e1.os ← e.os; e2.os ← e.os
10:                A*.add-edge(e1, e2)
11:                A*.remove-edge(e)
12:            end for
13:        end for
14:        K ← Critical-Clusters(A*, C)
15:    end while
16:    return A*
17: end procedure
```

Such a phase is represented by the procedure RESTORE() in Algorithm 3. The while cycle (rows 4-14) will be repeatedly performed, until all vertices and all segments become well separated. The main idea is the following one: for each segment $s \in \mathcal{S}$, which passes through a cluster c containing other vertices without passing through its kernel, an additional vertex v is introduced in order to split s into two segments that pass through the cluster kernel or outside it, on the basis of the relation between their original segment and the cluster.

Each edge $e \in \mathcal{E}$ of an arrangement $\mathcal{A}$ maintains a reference to the segment from which it has been generated, represented by the property *e.os*. A collection of edges generated from the same original segment *s* will be referred to in the following as a *chain*.

The procedure RESTORE() is based on the detection of the clusters that show a non-robust relation with an edge of the arrangement. Such clusters are called critical clusters.

Algorithm 4. CRITICAL-CLUSTERS procedure which determines which clusters in $\mathcal{C}$ are critical with respect to the edges of a weak-arrangement $\mathcal{A}$.

```
1: procedure CRITICAL-CLUSTERS (A, C)
2:     K ← ∅
3:     for c ∈ C do
4:         for e ∈ E do
5:             if e.intersect(c) ∧ ¬e.intersect(c.kernel) then
6:                 c.inter−edges.add(e)
7:                 K ← K ∪ {c}
8:             end if
9:         end for
10:    end for
11:    return K
12: end procedure
```

Definition 18. (Critical cluster). Given a weak-arrangement $\mathcal{A}$ and a set of clusters $\mathcal{C}$ induced by a tolerance value t, a cluster $c \in \mathcal{C}$ is said to be critical if and only if $\exists e \in \mathcal{E}$ such that e intersects *c.support* \ *c.kernel*, namely it intersects the cluster support without passing through its kernel.

Figure 4 illustrates an example of a cluster which is traversed by two edges, one of which intersects it without passing through its center. This is a critical cluster since it produces a not-robust situation: the above edge cannot be considered either well-separated or coincident with the vertices in the cluster.

Given a weak arrangement $\mathcal{A}$ and a set of clusters $\mathcal{C}$ for it, the RESTORE() procedure in Algorithm 3 determines the set of all critical clusters *K* and iterates through them in order to solve such critical situations. In particular, for each cluster $c \in K$, it re-routes each edge *e* which intersects it, by adding a new vertex *v* and splitting *e* into two edges e_1 and e_2. Procedure CRITICAL-CLUSTERS() determines which clusters of $\mathcal{C}$ are critical with respect to the edges of $\mathcal{A}$ and populates the property *c.inter-edges* containing for each cluster *c* the set of edges in $\mathcal{E}$ that intersects *c* without passing through its kernel.

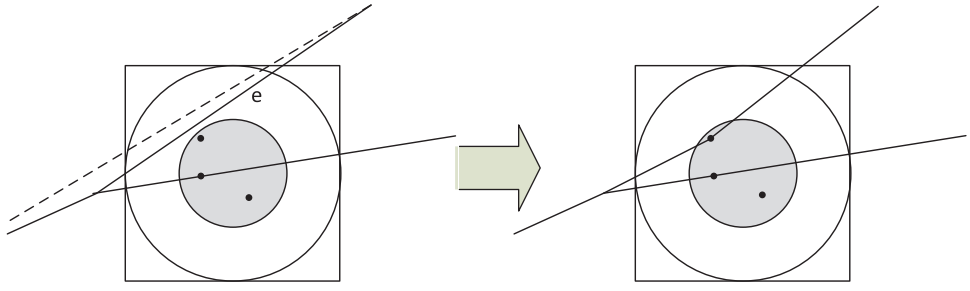

FIGURE 4 (a) Example of critical cluster: the solid edge *e* passes through a cluster without passing through its kernel. This produces a not-robust situation since the edge is neither well-separated nor coincident with the cluster vertices. The dashed line represents the original segment for the edge *e*; and (b) Result of application of the REROUTE() procedure to edge *e*: since the original segment passes through the cluster, then case (a) occurs and the edge *e* has to be split by adding a new vertex *v* inside the cluster kernel

The choice of the vertex v performed during the RESTORE procedure is a critical point of the algorithm, because it has to solve the current critical situation. Such a choice is performed by the CHOOSE-VERTEX() procedure illustrated in Algorithm 5. First, two situations can be distinguished: (a) the original segment $e.os$ passes through the cluster c; or (b) the original segment $e.os$ passes outside c.

Algorithm 5. CHOOSE-VERTEX procedure which takes as input a critical cluster c, an edge e, a set of all clusters $\mathcal{C}$, and returns a vertex v for re-routing the edge e outside the cluster c.

```
1: procedure CHOOSE-VERTEX(c, e, C)
2:     if e.os.intersects(c.support) then
3:         v ← CLOSER-VERTEX(c, e)
4:     else
5:         d_min ← +∞; v ← ⊥
6:         U ← USEFUL-CLUSTERS(c, e)
7:         if U ≠ ∅ then
8:             v ← Closer-USEFUL-VERTEX(U, e, c)
9:         end if
10:        if U=∅ ∨ v=⊥ then
11:            F ← FREE-CELL(c, C)
12:            if F ≠ ∅ then
13:                v ← Closer-USEFUL-VERTEX(F, e, c)
14:            else
15:                v ← CLOSER-VERTEX(c, e)
16:            end if
17:        end if
18:    end if
19:    return v
20: end procedure
```

In case (a) the chain produced by splitting e has to pass through the cluster kernel for maintaining the original relation. This case is treated by rows 2-3 of Algorithm 5 where the procedure CLOSER-VERTEX() identifies an appropriate vertex v of the cluster kernel for performing the split. In particular, the main idea is to choose a location inside the kernel that maintains the minimum distance between the new potential chain and its original segment.[4] Therefore, in order to implement such an idea while simplifying the choice process, the procedure CLOSER-VERTEX() returns one of the existing vertices inside the cluster c that minimize the distance between the new chain and its original segment, as illustrated in Algorithm 6.

Let us consider the situation in Figure 4a, where edge e passes through a critical cluster without passing through its center. Its original segment is identified by the dashed line which the cluster also traverses. Procedure CLOSER-VERTEX() tests the three vertices in the cluster in order to find the one that minimizes the distance between the potential chain obtained by splitting e and its original segment. The situation produced for e by the Restore procedure is the one in Figure 4b.

Algorithm 6. CLOSER-VERTEX procedure chooses the best vertex of the cluster c for re-routing the edge e. It returns the vertex inside c that minimizes the distance between the new chain and the original segment of e. Notation $\langle e_1, e_2 \rangle$ represents a chain composed by two edges e_1 and e_2.

```
1: procedure CLOSER-VERTEX(c, e)
2:     d ← ⊤; v ← ⊥;
3:     for x ∈ c do
4:         {e1, e2} ← split e at x;
```

```
5:        y ← distance(⟨e1, e2⟩, e.os);
6:        if y < d then
7:            d ← y; v ← x;
8:        end if
9:    end for
10:   return v
11: end procedure
```

In case (b) a new vertex v has to be added outside c so that the chain obtained by splitting e in v does not pass through c any more. For dealing with this case we introduce the definition of a cluster frame.

Definition 19. (Cluster frame). Given a weak-arrangement $\mathcal{A}$ and a cluster $c \in \mathcal{C}$ induced by a tolerance value t, the cluster frame of c is the set of clusters nearby c and is computed in the following way. A 9-cell grid g is built around c such that the central cell contains c and each cell has a side length of 2t. The frame of c (denoted as *c.frame*) is the set of clusters that intersects the grid cells built around the cluster c.

Figure 5 illustrates an example of a cluster frame for a cluster c which is contained at the grid center. The cluster frame of c is represented by all clusters which intersect the grid even if they are not completely contained in a cell.

Again two cases can be distinguished: (b_1) there exists a cluster $x \in c.frame$ such that $e.os$ passes through x while e does not, or (b_2) there are no clusters in *c.frame* such that only the original segment $e.os$ passes through it. Procedure USEFUL-CLUSTERS() illustrated in Algorithm 7 returns the set of clusters of *c.frame* which are intersected only by the original segment and not by e. This second condition is necessary in order to exclude the clusters to which the edge e has already been anchored by the initial procedure GATHERING() or by previous execution of the while cycle in the procedure RESTORE().

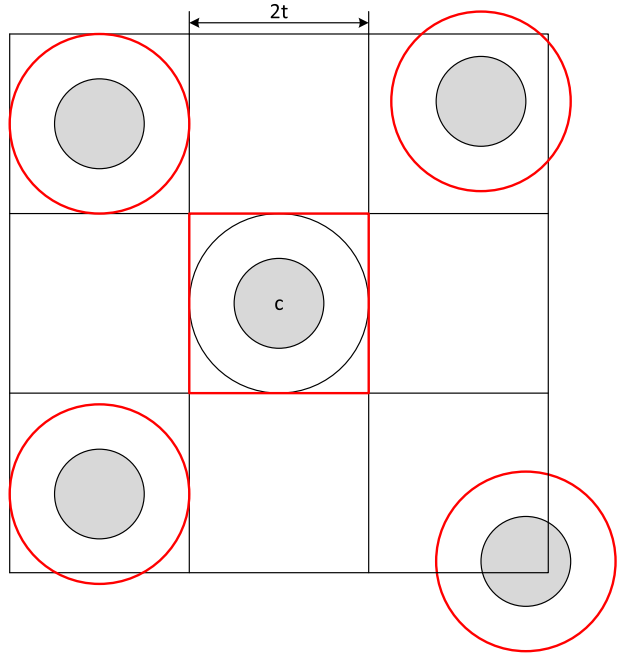

FIGURE 5 Example of frame for a cluster c placed at the center of the grid. The clusters composing the frame of c are those highlighted in red. Notice that a cluster belongs to the cluster frame even if it is not completely contained in a grid cell

Algorithm 7. USEFUL-CLUSTERS procedure which identifies among all clusters in the frame of c the ones that intersect the original segment of e but do not intersect e.

```
1: procedure USEFUL-CLUSTERS(c, e)
2:     U ← ∅;
3:     for c ∈ c.frame do
4:         if e.os.intersects(c)∧¬e.intersects(c) then
5:             U ← U ∪ {c}
6:         end if
7:     end for
8:     return U
9: end procedure
```

Case (b_1) is represented by rows 7-9 in Algorithm 5. In particular, the clusters identified by USEFUL-CLUSTERS() can be used for re-routing the edge outside c. The procedure CLOSER-USEFUL-VERTEX() reported in Algorithm 9 searches for clusters in the set U of useful ones, those whose centroid moves the potential new chain outside c, and among all candidate centroids chooses the one that minimizes the distance from the original segment. Figure 6 shows an example of situation b1: the original segment represented by the dashed line passes through a cell which contains a cluster while the edge does not. The chain obtained by adding a vertex at this cluster center does not pass through the central cluster any more.

Algorithm 8. FREE-CELL procedure which determines the cells in the frame of c which do not contain any cluster and returns as candidate vertices their center points.

```
1: procedure FREE-CELL(c, C)
2:     F ← GRID-CELLS-AROUND(c)
3:     for f ∈ F do
4:         for k ∈ c.frame do
5:             if k.intersect(f) then
6:                 F ← F \ {f}
7:             end if
8:         end for
9:     end for
10:    R ← ∅
11:    for f ∈ F do
12:        R ← R ∪ {f.center}
13:    end for
14:    return R
15: end procedure
```

If no vertices can be found by the procedure CLOSER-USEFUL-VERTEX() or there are no useful clusters (case b_2), a new cluster has to be built around c, as explained by rows 10-18 in Algorithm 5. Procedure FREE-CELL() determines the cells of the grid around c that are not occupied by any other cluster. In particular, function GRID-CELLS-AROUND() returns the grid cells around a particular cluster. Given such cells those that are occupied by a cluster in $c.frame$ will be removed. The remaining free cells are candidates for accommodating a new cluster containing only the vertex v. Procedure CLOSER-USEFUL-VERTEX() is applied in this case by considering the center of the cells in F as candidate vertices v and it verifies if any of these is able to move the chain outside c. Among all candidate clusters, it chooses as the result-vertex the center that minimizes the distance from the original segment. Figure 7 illustrates an example of this situation where there are no useful clusters around c. Procedure FREE-CELLS() returns cells 1, 2 and 3: cell 3 is not useful for re-routing e outside the cluster, while between cell 1 and 2, the last one is chosen because it produces a smaller distance from $e.os$.

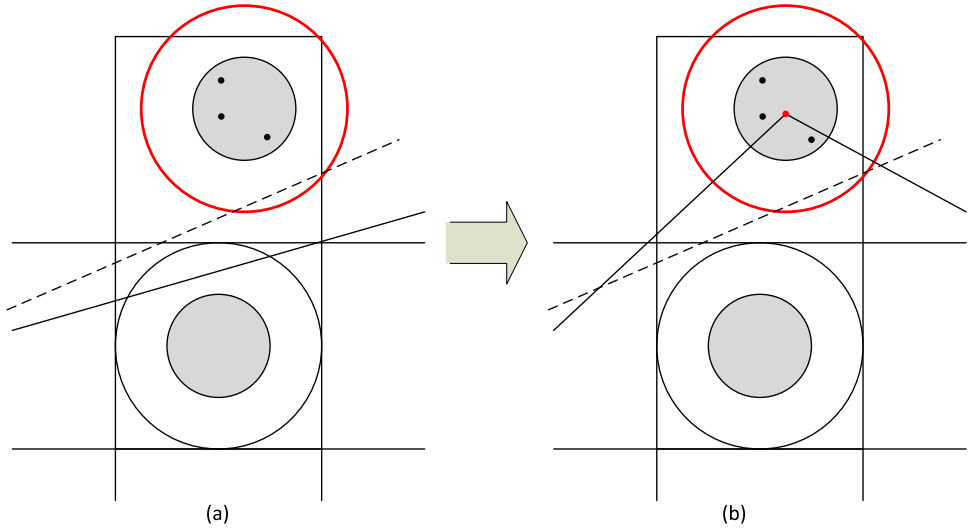

FIGURE 6 (a) Example of critical cluster: the solid edge *e*, produced at a certain stage of the TSR algorithm, passes through a cluster without passing through its kernel. The dashed line represents the original segment for the edge *e*; and (b) The result of the application of the REROUTE() procedure to edge *e*: since the original segment does not pass through the cluster but there exists a useful cluster such that *e.os* passes through it while *e* does not, case (b1) occurs and the edge *e* has to be split by adding a new vertex *v* at this cluster centroid

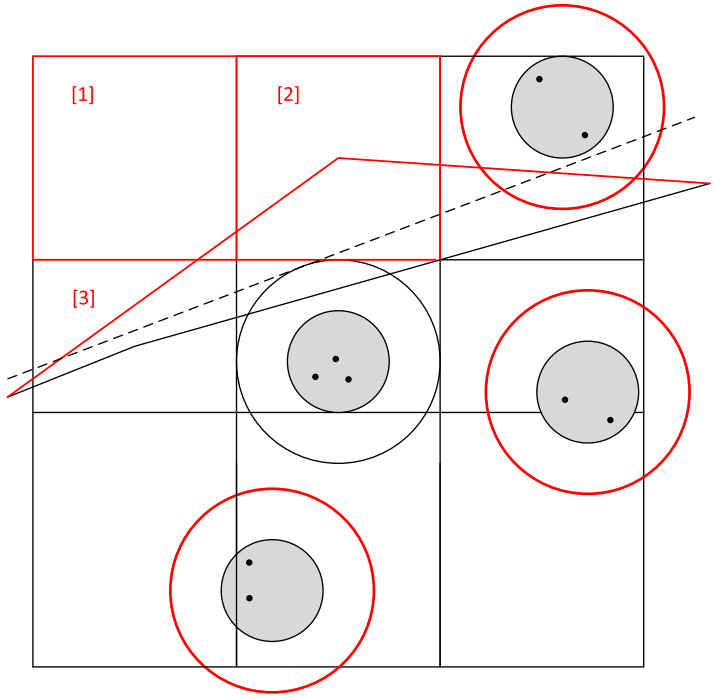

FIGURE 7 Example of critical cluster: the solid edge *e*, generated at a given stage of the TSR algorithm, passes through a cluster without passing through its kernel. The dashed line represents the original segment for the edge *e*. The red line represents the result of the application of the REROUTE() procedure to edge *e*: since there are no useful clusters, then case (b2) occurs. Procedure FREE-CELLS() returns cells 1, 2 and 3: cell 3 is not useful for re-routing *e* outside the cluster, while between cell 1 and 2, the last one is chosen because it produces a smaller distance from *e.os*

Finally, in the case where no free cell exists that moves the chain outside the cluster c, the algorithm chooses the vertex v as in case (a) (line 15 of Algorithm 5), because there is no possibility of rerouting e without passing through an existing cluster. In real cases, geometric primitives are usually not particularly dense; thus the latter case, which modifies the original relations, rarely occurs.

Algorithm 9. CLOSER-USEFUL-VERTEX procedure which chooses among all useful clusters in U the one which re-routes e outside c and minimizes the distance between the new potential chain and the original segment of e. Notation $\langle e_1, e_2 \rangle$ represents a chain composed by edges e_1 and e_2.

```
1: procedure CLOSER-USEFUL-VERTEX(U, e, c)
2:     d ← +∞; v ← ⊥;
3:     for u ∈ U do
4:         {e1, e2} ← split e at u.centroid
5:         if ¬e1.intersect(c)∧¬e2.intersect(c) then
6:             x ← distance(⟨e1, e2⟩, e.os)
7:             if x < d then
8:                 d ← x; v ← u.centroid;
9:             end if
10:        end if
11:    end for
12:    return v
13: end procedure
```

5.4 | TRS algorithm

The TRS algorithm is illustrated in Algorithm 10. It starts by gathering the vertices contained in the arrangement $\mathcal{A}$ built from the input dataset D in order to make them well separated, and then applies the Restore procedure to establish or restore the dataset robustness.

Proposition 7. Given a weak-arrangement $\mathcal{A}$ for a dataset D, the application of Algorithm 10 produces a modified arrangement $\mathcal{A}^*$, which corresponds to a robust dataset.

Proof. Algorithm 10 firstly applies the GATHERING() procedure which makes coincident points at a distance less than $t-2\delta$ (TME rule for points) and different points well-separated at a distance grater than $t+2\delta$ (TMD rule for points).

Secondly, it applies the procedure RESTORE() which terminates when no critical clusters can be found. Therefore, any segment s which passes through an existing cluster, will pass through its kernel; namely, any coincident vertex and segment has a distance less than or equal to $t-2\delta$ (TME rule for point/segment). Conversely, the same procedure RESTORE() guarantees that vertices and segments belonging to two different clusters are at a distance greater than $t+2\delta$ (TMD rule for point/segment).

As proved in Proposition 5, if a dataset satisfies rules TME and TMD, it is δ-robust in the tolerance model.

Algorithm 10. The TSR algorithm applied to a weak-arrangement $\mathcal{A}$ with a tolerance value t. It produces a modified arrangement $\mathcal{A}^*$ which corresponds to a robust dataset.

```
1: procedure TSR(C, A, t, δ)
2:     A1 ← GATHERING(C, A, t, δ)
3:     A* ← RESTORE(A1, t)
4:     return A*
5: end procedure
```

Proposition 8. The TSR algorithm has a complexity of $O(n^3 \log n)$ in the worst case scenario where n is the number of vertices in the arrangement $\mathcal{A}$.

Proof. The TSR algorithm firstly performs a GATHERING() phase which iterates for all clusters in the set $\mathcal{C}$ and then for each vertex in the cluster it performs an approaching to the cluster center. The APPROACHING() procedure takes a constant time to be performed, therefore the overall GATHERING() procedure takes an $O(n)$ time.

Procedure RESTORE() firstly determines the set of critical clusters through the procedure CRITICAL-CLUSTERS(). This procedure iterates for all clusters in $\mathcal{C}$ and for each of them finds the set of edges which intersect it without passing through its center. This is the most time consuming operation, because in the worst case scenario the number of clusters is $O(n)$ while the number of edges is $O(n^2)$, thus the complexity of the procedure CRITICAL-CLUSTERS() is $O(n^3)$. This complexity can be reduced by optimizing CRITICAL-CLUSTERS(); this can be done by using a plane sweep algorithm, which reduces the number of tests to be performed to $O(m \log m)$ where m is the number of edges, namely $O(n^2 \log n)$.

Given the set of critical clusters K which have a worst case cardinality of $O(n)$, the RESTORE() procedure iterates for each critical cluster and solves the corresponding critical situation. The procedure CHOOSE-VERTEX() performs different operations based on the specific situation: in case (a)[5] the procedure CLOSER-VERTEX() has a complexity of $O(h)$ where h is the number of vertices in a cluster c; in case (b1) and (b2) the procedure firstly determines the set of useful clusters, where procedure USEFUL-CLUSTERS() has a complexity of $O(l)$ where l is the number of clusters in *c.frame*. Secondly, in case (b1), if there are some useful clusters, procedure CLOSER-USEFUL-VERTEX() determines the closer vertex by iterating through the set of useful clusters. In the worst case where all clusters in *c.frame* are useful, it has a complexity of $O(l)$. Conversely, if there are no useful clusters or in case (b2), the procedure FREE-CELL() is performed, it has a complexity of $O(9l) = O(l)$ where l is again the number of clusters in *c.frame*. Given the set F of FREE-CELL() the resulting vertex v is chosen with a complexity of $O(l)$ in case F is not empty, or $O(h)$ in case F is empty. Therefore, the overall complexity of the main loop (rows 4-14) inside the RESTORE() procedure is $O(n) \cdot \max\{O(h), O(l) + \max\{O(l), O(h)\}\} + O(n^2 \log n)$. In the worst case, the number of vertices in the cluster c can be n, but in this case the number of clusters cannot be n; more specifically, if the cardinality of the cluster c is k, the total number of clusters can be at most $O(n-k+1)$. Anyway, the complexity of the main loop is dominated by the complexity of the CRITICAL-CLUSTERS() procedure, thus it is again $O(n^2 \log n)$.

The final aspect to consider is the number of times the main loop of the RESTORE() procedure is performed. In the worst-case scenario, there is a cluster for each vertex, each cluster is critical and only one critical cluster is solved during a loop iteration. In this case, the main loop is performed $O(n)$ times producing an overall complexity of $O(n^3 \log n)$.

Notice that, the configuration that produces the worst case for complexity is quite impossible, because if a cluster is built for each vertex, then the dataset is sparse and there will be very few critical clusters. Conversely, if the dataset is dense, producing many critical situations, the number of clusters is low. The experiments illustrated in the following section demonstrate that the algorithm can be applied to real datasets with acceptable execution times.

6 | IMPLEMENTATION AND EXPERIMENTS

The TSR algorithm proposed in the previous section has been tested on a real-world dataset describing the road network of some valleys in the Alpine region of Northern Italy. In particular, it contains 12,262 linestrings, representing road links.

The main aim of these tests was to verify that the introduction of some perturbations on the original dataset can modify the topological relations initially computed, while if the dataset is made robust with the TSR algorithm the existing topological relations do not change even in the presence of perturbations. For this reason, the topological relations with tolerance presented in Section 4 have been implemented in PostGIS (OSGeo, 2014), then the relations existing between any pair of linestrings with intersecting MBR have been computed with a tolerance of 10^{-2} (in accordance with the metric accuracy of the dataset), producing the 29,343 relations summarized in Table 2.

The original dataset *DS* has then been modified by introducing a random perturbation. In particular, four different perturbations have been applied with δ equal to 10^{-6}, 10^{-5}, 10^{-4} and 10^{-3}, producing four corresponding datasets *D*

TABLE 2 Topological relations computed on the 29,343 pairs of linestrings with intersecting MBR contained in the original dataset *DS* in Figure 8, considering a tolerance threshold $t=10^{-2}$

Relation	# Occurrences
Disjoint (DJ)	15,580
Touch (TC)	12,313
Crosses (CR)	637
In (IN)	5
Overlap (OV)	808

S_{p_6}, DS_{p_5}, DS_{p_4} and DS_{p_3}. The upper bound 10^{-3} for the perturbation range satisfies the condition mentioned at the end of Section 4.4 for which t must be greater than 2δ, while the lower bound 10^{-6} has been chosen since perturbing with the lower value of δ does not alter the relations among linestrings.

Given such datasets, the set of existing topological relations has been computed again for each of them, as in the previous test, in order to check which relations (of the 29,343 relations computed on *DS*) have been changed. Table 3 shows the differences between the topological relations computed on *DS* and the ones computed on DS_{p_6}, DS_{p_5}, DS_{p_4} and DS_{p_3}, respectively. The first column identifies the kind of transformation: $r_1 \rightarrow r_2$ means that an r_1 relation in the original dataset *DS* has become an r_2 relation in the perturbed dataset DS_{p_x}. The obtained transformations match the relations: overlaps (*ov*), touches (*tc*), crosses (*cr*) and disjoint (*dj*).

An implementation of the algorithm has been developed in Java 7u51 using the Java Topology Suite API (Vivid Solutions, 2014) for performing the required geometric operations, and has been executed on an machine with an Intel i7 4770K processor and 16GB of RAM. This implementation uses a fixed grid with a cell side of $2t$ in order to perform the initial clustering and represents a preliminary experiment of algorithm applicability: it can be further improved by using more sophisticated clustering techniques and data structures. In particular, the performed experiments reveal that the applied clustering techniques determine the number of different topological relations between the original dataset *DS* and its robust version DS^{δ}. As explained in Section 5.1, the quality of the result depends upon the initial centroids and their choice has to consider the distribution of points. Moreover, notice that modifications on the tolerance value t will affect the result of the initial clustering phase and eventually the number of identified critical clusters,

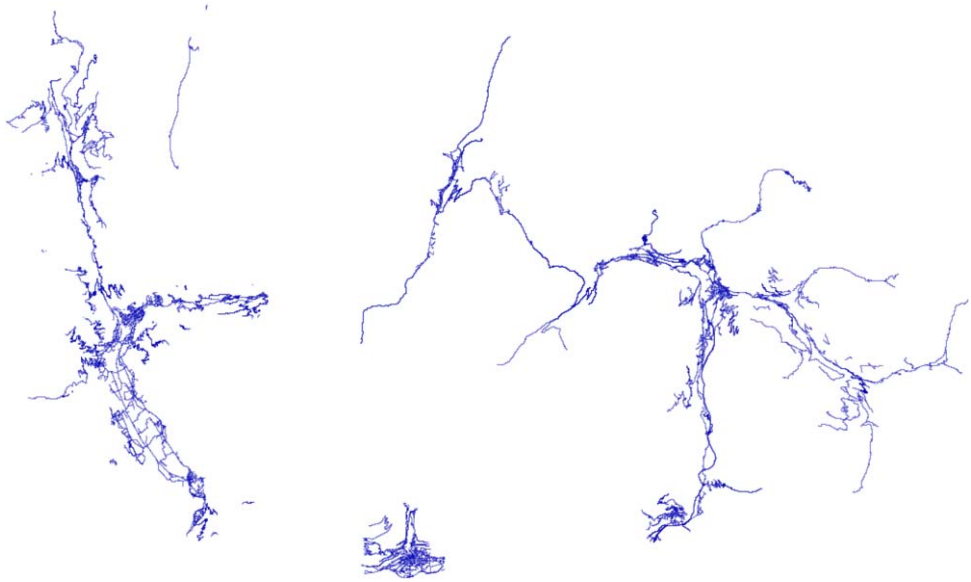

FIGURE 8 Real-world dataset considered during the experiments

TABLE 3 Changes between the topological relations computed on the original not-robust dataset *DS* and the ones computed on its corresponding perturbed versions DS_{p6}, DS_{p5}, DS_{p4} and DS_{p3}, for values of δ between 10^{-6} and 10^{-3}

Change	$\delta=10^{-6}$	$\delta=10^{-5}$	$\delta=10^{-4}$	$\delta=10^{-3}$
ov → *tc*	–	–	7	50
ov → *cr*	–	–	1	4
tc → *ov*	–	2	5	58
tc → *dj*	–	2	12	162
tc → *cr*	–	–	3	34
dj → *tc*	–	3	17	170
cr → *ov*	–	–	1	3
cr → *tc*	–	–	5	27
total	0	7	51	508
	(0.00%)	(0.02%)	(0.17%)	(1.73%)

namely smaller values of *t* could produce more critical situations because vertices or segments that are very near to each other are considered to be separate instead of coincident.

The aim of the performed tests was to verify that TSR establishes or restores the robustness of a dataset independently from the quality or nature of the clustering process. Therefore, given the original dataset *DS* described above, the TSR algorithm has been applied considering the mentioned perturbation values, producing four datasets which are robust with respect to a maximum perturbation of the corresponding entity, called DS^{δ_6}, DS^{δ_5}, DS^{δ_4} and DS^{δ_3} respectively. Notice that, the maximum admissible perturbation value is used by the algorithm in order to determine the cluster kernel; therefore, the amount of movement induced by the GATHERING() and REROUTE() procedures can be different in the four cases.

Table 4 contains some interesting information about the TSR execution for each considered perturbation value δ. In particular, the number of input road links, the number of clusters and critical clusters contained in the original dataset, the number of iterations performed by the algorithm and the number of produced segments. In the four cases the RESTORE() procedure performs only three iterations of the main loop to obtain the result, even if the number of clusters is about 200,000, confirming that the provided complexity analysis represents a worst-case scenario that rarely occurs. Similarly, the number of critical clusters is an order of magnitude less than the number of total original clusters.

Finally, the four robust datasets, DS^{δ_6}, DS^{δ_5}, DS^{δ_4} and DS^{δ_3}, obtained from TSR, have been perturbed, each one with the corresponding admissible maximum perturbation and the set of topological relations has been computed again in order to verify that the algorithm ensures the robustness of the dataset. The tests confirm that all relations computed on each robust dataset DS^{δ_x} are maintained even in the presence of perturbations, namely in its perturbed version $DS^{\delta_x}_{p_x}$.

TABLE 4 Metrics describing about the TSR execution on the original dataset considering different perturbation values δ

p	#LS	#CL	#CCL	#IT	#Edges
10^{-6}	12,262	198,427	18,331	3	229,008
10^{-5}	12,262	198,427	18,355	3	229,039
10^{-4}	12,262	198,427	18,549	3	229,345
10^{-3}	12,262	198,427	20,287	3	231,838

Column #LS = contains the number of linestrings contained in the input dataset; #CL = contains the number of initial clusters, #CCL contains the number of initial critical clusters, #IT reports the number of main-loop iterations performed by restore, and #Edges the number of segments contained in the resulting dataset.

7 | CONCLUSION

This article deals with the potential robustness problems affecting a spatial dataset that is exchanged between different systems. In particular, it considers a vector representation of geometric primitives based on a tolerance model, namely a model in which equality is based on a tolerance threshold rather than on the identity between geometric primitives. Given the results presented in Belussi et al. (2013) about the applicability of existing algorithms for establishing the robustness of a dataset in the tolerance model, this article proposes a new algorithm for this purpose, which is called Tolerance-based Snap Rounging (TSR) and has been inspired by those in the Snap Rounding (SR) family.

Two key ideas differentiate TSR from SR algorithms: (1) in order to preserve as much as possible the original relation between two primitives, an additional operation is added to the snapping one, which moves away primitives that are initially disjoint; and (2) a tolerance region is considered during the snapping or spreading operation, instead of a single location (pixel), as usually done in the SR approach.

A preliminary implementation of the algorithm has been developed in order to test the applicability of the technique on a real-world dataset which was initially not-robust. The obtained results encourage the development of a more sophisticated implementation as a future work. In particular, the possible improvements regard both the applied clustering technique and the use of more efficient data structures for reducing the overall complexity. The performed experiments reveal that the applied clustering algorithm influences the number and kind of topological relations that are preserved between the original dataset and its robust version. Therefore, it has to be chosen very carefully considering both the geometric and semantic dataset characteristics. At the same time the development of particular data structures can reduce the complexity of the operations performed at any iteration in order to identify the set of critical clusters and choose the useful splitting vertex.

NOTES

[1] This function returns the distance between the geometries that are stored in the dataset, while a function that estimates the effective distance among the real objects that are represented by those geometries has not been defined in the proposed model.

[2] The left (right) region induced by s is the set of points belonging to the half-plane on the left (right) of s and having a orthogonal projection on s.

[3] $\Delta_x = v.x - c.kernel.x$, $\Delta_y = v.y - c.kernel.y$

[4] The distance between the potential chain and its original segment is computed as the average distance between the endpoints of the chain edges and the original segment.

[5] See the discussion about the procedure CHOOSE-VERTEX() in Section 5.3.

REFERENCES

Belussi, A., Migliorini, S., Negri, M., & Pelagatti, G. (2012). *Robustness of spatial relation evaluation in data exchange*. Paper presented at the Twentieth ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Redondo Beach, CA, 446–449.

Belussi, A., Migliorini, S., Negri, M., & Pelagatti, G. (2013). *On robust interpretation of topological relations in identity and tolerance models*. Paper presented at the Twenty-first ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Orlando, FL, 468–471.

Belussi, A., Migliorini, S., Negri, M., & Pelagatti, G. (2015). Impact of data representation rules on the robustness of topological relation evaluation. *Geoinformatica*, 19, 185–226.

Belussi, A., Migliorini, S., Negri, M., & Pelagatti, G. (2016). Snap rounding with restore: An algorithm for producing robust geometric datasets. *ACM Transactions on Spatial Algorithms & Systems*, 2, 1, 1.

Chandra, E. A. V. (2011). A survey on clustering algorithms for data in spatial database management systems. *International Journal of Computer Applications*, 24, 19–26.

Chen, L. (2001). *Exact geometric computation: Theory and applications* (Unpublished doctoral dissertation). New York University, New York.

Christofides, N. (1975). *Graph theory: An algorithmic approach*. Orlando, FL: Academic Press.

Duda, R. O., & Hart, P. E. (1973). *Pattern classification and scene analysis*. Chichester, UK: Wiley Interscience.

Egenhofer, M. J., Frank, A. U., & Jackson, J. P. (1990). *A topological data model for spatial databases*. Paper presented at the First Symposium on Design and Implementation of Large Spatial Databases, Santa Barbara, CA, 271–286.

Egenhofer, M. J., & Franzosa, R. (1991). Point-set topological spatial relations. *International Journal of Geographical Information Systems*, 5, 161–174.

ESRI. (2010). *Understanding geometric processing in ArcGIS*. Retrieved from http://www.esri.com

ESRI. (2014). *ArcGIS*. Retrieved from http://www.esri.com/

Ester, M., Kriegel, H. P., Sander, J., & Xu, X. (1996). *A density-based algorithm for discovering clusters in large spatial databases with noise*. Paper presented at the Second International Conference on Knowledge Discovery and Data Mining, Portland, OR, 226–231.

Halperin, D. (2010). *Controlled perturbation for certified geometric computing with fixed-precision arithmetic*. Paper presented at the Third International Congress Conference on Mathematical Software, Kobe, Japan, 92–95.

Halperin, D., & Packer, E. (2002). Iterated snap rounding. *Computational Geometry: Theory and Applications*, 23, 209–225.

Hinneburg, A., & Keim, D. A. (1998). *An efficient approach to clustering in large multimedia databases with noise*. Paper presented at the Fourth International Conference on Knowledge Discovery and Data Mining, New York, 58–65.

Hobby, J. (1999). Practical segment intersection with finite precision output. *Computational Geometry: Theory and Applications*, 13, 199–214.

Kanungo, T., Mount, D., Netanyahu, N., Piatko, C., Silverman, R., & Wu, A. (2002). An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 24, 881–892.

MacQueen, J. (1967). Some methods for classification and analysis of multi-variate observations. In L. M. Le Cam & J. Neyman (Eds.), *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, (pp. 281–297). Berkeley, CA: University of California Press.

Milenkovic, V. (1988). Verifiable implementations of geometric algorithms using finite precision arithmetic. *Artificial Intelligence*, 37, 377–401.

Ng, R. T., & Han, J. (1994). *Efficient and effective clustering methods for spatial data mining*. Paper presented at the Twentieth International Conference on Very Large Data Bases, Santiago de Chile, Chile, 144–155.

Open Geospatial Consortium (OGC). (2012). *OGC Geography Markup Language (GML): Extended Schema and Encoding Rules, Version 3.3.0*. Retrieved from https://portal.opengeospatial.org/files/?artifact_id=46568.

OSGeo. (2014). *PostGIS 2.1*. Retrieved from http://postgis.net

Packer, E. (2008). Iterated snap rounding with bounded drift. *Computational Geometry: Theory and Applications*, 40, 231–251.

Pelagatti, G., Negri, M., Belussi, A., & Migliorini, S. (2009). *From the conceptual design of spatial constraints to their implementation in real systems*. Paper presented at the Seventeenth ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Seattle, WA, 448–451.

Praing, R., & Schneider, M. (2009). Topological feature vectors for exploring topological relationships. *International Journal of Geographical Information Science*, 23, 319–353.

Pullar, D. (1993). *Consequences of using a tolerance paradigm in spatial overlay*. Paper presented at AutoCarto 11, Minneapolis, MN.

Rodríguez, M. A., Brisaboa, N., Meza, J., & Luaces, M. R. (2010). *Measuring consistency with respect to topological dependency constraints*. Paper presented at the Eighteenth SIGSPATIAL International Conference on Advances in Geographic Information Systems, San Jose, 182–191.

Theobald, D. M. (2001). Topology revisited: Representing spatial relations. *International Journal of Geographical Information Science*, 15, 689–705.

Thompson, R. J., & van Oosterom, P. (2006). *Interchange of spatial data-inhibiting factors*. Paper presented at the Ninth AGILE International Conference on Geographic Information Science, Visegrád, Hungary.

Vivid Solutions. (2014). *JTS Topology Suite*. Retrieved from http://www.vividsolutions.com/jts/JTSHome.htm

Xu, X., Ester, M., Kriegel, H. P., & Sander, J. (1998). *A distribution-based clustering algorithm for mining in large spatial databases*. Paper presented at the Fourteenth International Conference on Data Engineering, Orlando, FL, 324–331.

Yuan, F., Meng, Z. H., Zhang, H. X., & Dong, C. R. (2004). A new algorithm to get the initial centroids. *Machine Learning and Cybernetics*, 2, 1191–1193.

Zhang, T., Ramakrishnan, R., & Livny, M. (1996). *BIRCH: An efficient data clustering method for very large databases*. Paper presented at *the 1996 ACM SIGMOD International Conference on Management of Data*, Montreal, Quebec, 103–114.

SUPPORTING INFORMATION

Additional supporting information may be found in the online version of this article at the publisher's website.

How to cite this article: Belussi A, Migliorini S, Negri M, Pelagatti G. Establishing robustness of a spatial dataset in a tolerance-based vector model. Transactions in GIS. 2017;21:722–747. https://doi.org/10.1111/tgis.12221