International Journal of Geographical Information Science
Vol. 00, No. 00, Month 200x, 1–26

RESEARCH ARTICLE

Rank-based Strategies for Cleaning Inconsistent Spatial Databases

Nieves R. Brisaboa[a], M. Andrea Rodríguez[b], Diego Seco[ab], Rodrigo A. Troncoso[b]
[a]*Database Laboratory, University of A Coruña, Spain*
[b]*Department of Computer Science, University of Concepción, Chile*

(*Received 00 Month 200x; final version received 00 Month 200x*)

A spatial dataset is consistent if it satisfies a set of integrity constraints. Although consistency is a desirable property of databases, enforcing the satisfaction of integrity constraints might not be always feasible. In such cases the presence of inconsistent data may have a negative effect on the results of data analysis and processing and, in consequence, there is an important need for data-cleaning tools to detect and remove, if possible, inconsistencies in large datasets. This work proposes strategies to support data cleaning of spatial databases with respect to a set of integrity constraints that impose topological relations between spatial objects. The basic idea is to rank the geometries in a spatial dataset that should be modified in order to improve the quality of the data (in terms of consistency). An experimental evaluation validates the proposal and shows that the order in which geometries are modified affects both the overall quality of the database and the final number of geometries to be processed to restore consistency.

Keywords: Data cleaning; spatial inconsistency; spatial dataset; inconsistency graph

1. Introduction

Spatial databases are the core of applications such as planning, navigation, and cadastral systems. They are typically implemented as object-relational or extended-relational models that support query languages to manipulate geo-referenced data and provide spatial indexing and efficient algorithms for spatial query processing (Güting 1994). A database, and therefore a spatial database, is said to be consistent if it satisfies a set of integrity

ISSN: 1365-8816 print/ISSN 1362-3087 online
© 200x Taylor & Francis
DOI: 10.1080/13658816.2014.965711
http://www.informaworld.com

constraints. These integrity constraints define valid states of the data and are usually expressed in a language that also defines the data schema (logical representation).

Although consistency, as part of the data quality problem, is widely recognized as one of the critical issues in the use of spatial databases (Borges *et al.* 2002, Davis *et al.* 2005, Veregin 1999), most spatial database management systems (SDBMS) provide mechanisms for enforcing only domain constraints that just validate values of spatial attributes (e.g. a polygon is a simple closed polyline), leaving to database designers the task of checking other types of domain-application integrity constraints. The typical enforcement of database consistency is done as new data is inserted or updated in the database. However, in many cases, mechanisms for enforcing the satisfaction of those constraints at the time of database updates are not necessarily available or even feasible due to, among other reasons, time lag updates, use of legacy data, technical restrictions, or integration of different data sources. Because the presence of inconsistent data may have a negative effect on the results of data analysis and processing, there is an important need for specifying constraints tailored to spatial data, implementing efficient techniques to check their satisfaction, and designing data-cleaning tools to remove, if possible, inconsistencies in large datasets.

Several studies in the relational context have addressed the use of dependency constraints for data cleaning (Chaudhuri *et al.* 2006, Jin *et al.* 2003). In comparison to the relational context, spatial databases offer new alternatives and challenges for the use of integrity constraints. This is mainly due to the use of complex attributes to represent geometries, their combination with thematic attributes, and the nature of spatial (topological) relations. Consider, for example, a database that stores data (i.e. geometry plus others types of attributes) about land parcels and buildings. One would like to keep data in the database that satisfy some basic topological relations between land parcels and between land parcels and buildings. In particular, land parcels can only touch each other or be disjoint, and a building must be within the boundaries of a land parcel. If for some reason, the geometries stored in the database do not satisfy any of these constraints (e.g. two geometries of land parcels overlap), we would say that the database is inconsistent.

To clean a spatial database, there may be theoretically infinite ways to modify a geometry because the space is continuous. In a real setting, however, a data-cleaning process of a spatial database is usually guided by orthophotos[1]. Technical staff in charge of repairing an inconsistent spatial database takes one geometry at a time, compares it with an orthophoto, and repairs the whole geometry (in case it contains any error). Here, repair means to modify the geometry in the database such that it matches with the corresponding geometry in the orthophoto. In consequence, the orthophoto is always considered to be the truth that satisfies the integrity constraints imposed to the database. The key factor in this one-geometry-at-a-time process is the order in which geometries are checked and modified to restore consistency, which can be regarded as a ranking.

Within the spatial domain context, no much work is found addressing an automatic or semiautomatic strategy for data cleaning. Although there exist previous efforts to define procedures to restore consistency with respect to topological inconsistency (Servigne *et al.* 2000, Xie *et al.* 2010, Rodríguez *et al.* 2013), they are far from being systematic methods to be applied automatically. This is not a trivial problem because there are many different ways to transform a geometry to restore consistency, and conflicts between geometries cannot be always treated independently. In addition, even applying a simple strategy to restore consistency, such as the one proposed by Rodríguez *et al.* (2013) that shrinks

[1]Miguel R. Luaces (gisEIEL Project), personal communication.

geometries to eliminate conflicting regions, leads to a hard problem. In such cases, one could use approximation algorithms or try to reduce the problem to treatable cases.

Instead of addressing the automatic modification of geometries to restore consistency in spatial databases, this work proposes strategies to rank the geometries that should be modified to improve the quality of a spatial database. This is important not only to diminish the number of geometries to modify, but also to determine a subset of geometries that can be modified to significantly improve the quality of the database. The latter is even more important when, for some reason, it is not possible to modify all geometries and the effort has to focus on few but relevant geometries. This work concentrates on restoring consistency with respect to topological dependency constraints, because they are widely used in practice and because there exist consistency measures defined for them (Brisaboa *et al.* 2014). In summary, the main contributions of this work are:

- We formalize the data-cleaning process in terms of an inconsistency graph, which shows the computational complexity of solving this problem and provides alternative approximation approaches.
- We propose a practical approach (approximation algorithms) that ranks geometries in terms of different measures.
- We evaluate the different ranking strategies with respect to a random-order of geometries. To do so, we use datasets created from a generalization process on consistent real data.

The organization of the paper is as follows. Section 2 revises related work concerning spatial integrity constraints and data cleaning. Section 3 introduces the data model and the integrity constraints upon which the proposed strategies are presented in Section 4. The experimental evaluation is given in Section 5, and conclusions and future research directions are addressed in Section 6.

2. Related Work

Spatial inconsistency refers to a contradiction between stored data and spatial integrity constraints, which can be classified into *domain* and *semantic* constraints. Domain constraints (a.k.a. *topological constraints*) specify admissible values of spatial attributes, whereas semantic constraints (a.k.a. *topo-semantic integrity constraints*) associate the semantics of the modeled entities with spatial properties, in particular, with topological relations between spatial objects (Cockcroft 1997).

In terms of the specification of spatial integrity constraints, related work addresses the specification of topological constraints (Davis *et al.* 1999) and spatial semantic constraints (Hadzilacos and Tryfona 1992, Mäs 2007, Servigne *et al.* 2000). Recently, the work by Bravo and Rodríguez (2009, 2012) formalizes a set of spatial integrity constraints and studies the satisfiability problem for these constraints. These constraints have also been extended to deal with spatio-temporal data that represent regions that evolve in time (del Mondo *et al.* 2013).

Consistency, as part of the notion of data quality, is widely recognized as one of the critical issues in the use of spatial databases (Borges *et al.* 2002, Davis *et al.* 2005, Veregin 1999). In this context, some studies have addressed strategies for data cleaning with respect to constraints that impose topological relations. Servigne *et al.* (2000) defined a methodology for spatial consistency improvement that suggests to change geometries through translation, reshaping, removing and splitting. They proposed particular situations when applying these changes. Although this work is useful for showing how to correct inconsistency, it falls short to provide a systematic way to do it. Moreover, it does not consider interaction between inconsistencies such that making one change on a geometry to solve a conflict may produce new conflicts between this and other geometries.

Deng *et al.* (2003) also used transformations of geometries to restore consistency. Their work defines a generalized algorithm for calculating the best location of a new point created by snapping a group of points within a fuzzy tolerance. This extends existing snapping functions in commercial GIS packages by providing a related error propagation for accuracy assessment. This algorithm was applied by Deng *et al.* (2005) to match vertexes and formalize boundary inconsistency by vertical projection. Similarly, Xie *et al.* (2010) proposed correcting methods of topological inconsistency based on buffer operations, Delaunay triangulation, skeleton extraction, among others. Despite the contributions made by these previous works, there is no comparison among them and no consideration of the interaction of different transformations when considering a set of integrity constraints.

With the objective of formalizing consistency query answers from inconsistent spatial databases,Rodríguez *et al.* (2013) defined a repair semantics based on shrinking geometries with respect to topological dependency constraints. The work defines a distance measure to identify a new database instance that differs minimally from the original inconsistent instance and satisfies the integrity constraints. This new instance is called a minimal repair of the inconsistent database. In principle, one could use the concept of minimal repair to restore consistency. This concept considers the interaction of a set of integrity constraints, however, finding a minimal repair of an original inconsistent database was shown to be intractable in general (Rodríguez *et al.* 2013).

Related to the data-cleaning problem, inconsistency measures can characterize the quality of the database and provide a mechanism to prioritize conflicts to solve. Inconsistency measures of databases with respect to constraints that impose topological relations require to compare the topological relations between objects in the database with respect to the expected topological relations between them. A qualitative approach to define similarity measures compares topological relations using the semantic distance between relations as defined by a conceptual neighborhood graph (Papadias *et al.* 1998). This type of measure suffers the disadvantage that it does not distinguish pairs of objects that, holding the same topological relation, correspond to different spatial configurations. A quantitative approach uses the metric refinement of geometric representations to characterize and, potentially compare, topological relations. In this context, the work in Godoy and Rodríguez (2004) characterizes topological relations as combinations of overlapping areas and distances between objects represented by their minimum bounding rectangles. With the goal of studying the geometry associated with natural-language spatial terms, the work in Egenhofer and Shariff (1998) and Egenhofer and Dube (2009) also defines a set of metric refinements consisting of splitting measures and closeness measures to characterize topological relations between a line and a region, or between regions. The work in Egenhofer and Dube (2009) concludes that a combination of up to three of these measures determines uniquely a topological relation. All these quantitative approaches can define a similarity function by comparing relations between pairs of geometries.

Focusing on the quantification degree of consistency of spatial datasets with respect to topological dependency constraints, the work in Brisaboa *et al.* (2014) and Rodríguez *et al.* (2010) is the first to introduce formally some inconsistency measures. These measures compare the topological relation that two geometries hold with respect to the topological relation they should hold. They are inspired by metric refinements of topology relations and apply to any kind of geometries, that is, relations between points, lines, regions, and their combinations. As the strategies presented in this paper are related with these measures, we give more details on them in Section 3.

Although there are some attempts to define the consistency degree in relational databases (Hunter and Konieczny 2005, Martinez *et al.* 2007, Ordonez *et al.* 2007), these measures have not been used for data-cleaning purposes.

3. Preliminaries

Current models of spatial database systems are typically seen as extensions of the relational data model (object-relational model), with the definition of abstract data types to specify spatial attributes. In this section, we review the spatial database model and the integrity constraints used in this work.

Spatial databases. A database schema defines the structure of the data that can be stored in the database. A spatio-relational database schema, in particular, is composed of a finite set of relations, each of them with an ordered set of attributes that take values from a specific domain. Thematic attributes take values from alphanumeric domains and spatial attributes (i.e. geometries) take values from the power set of $\mathbb{R}^2$. Following current implementations of Spatial Database Management Systems (SDBMs), spatial attributes in the 2D[1] space are defined by finite sets of points, lines, or polygons. A spatial database schema also specifies built-in spatial predicates, in particular, topological relations (e.g. Disjoint, Within, etc.) and spatial operators (e.g. area, length, etc.) over spatial attributes. These spatial predicates are then used as part of spatial query languages.

A database instance is the state (content) of a database at a particular time instant. It is composed of a set of tuples for each relation in the database schema. Each tuple is an ordered set of values, each one taking a value from the domain of the corresponding attribute.

Example 3.1 Consider the database whose instance is shown in Figure 1. The schema of the database contains two relations: (1) The relation LandP, denoting land parcels, contains thematic attributes *idl* and *name*, and a spatial attribute *geometry*, of data type *polygon*. (2) The relation Building contains thematic attributes *idb* and *idl*, representing the identification of a building and the identification of the land parcel it belongs to, respectively, and a spatial attribute *geometry*.

Formally, the database schema is composed of the relation schemes (i.e. predicates) $\mathsf{LandP}(idl, name, geometry)$ and $\mathsf{Building}(idb, idl, geometry)$. The database instance is composed of tuples of the form $\mathsf{LandP}(l_i, n_i, g_i)$ and $\mathsf{Building}(b_j, l_j, g_j)$, where l_i and l_j are values from the domain of the attribute *idl* (note that l_i may be equal to l_j), n_i is a value from the domain of the attribute *name*, b_j is a value from the domain of the attribute *idb*, and g_i and g_j are values from the domain of spatial attributes.

In addition to show the database instance in the form of tables, Figure 1 visualizes the geometries (values of the spatial attributes) of buildings represented as dark polygons, and the geometries of land parcels represented as white polygons.

□

For data manipulation purposes, languages of spatial databases define binary topological relations with fixed semantics. This work focuses on a subset of topological relations for 2D objects that are currently implemented in spatial query language specifications

[1]It is possible to extend this work to a 3D space by considering the corresponding set of topological relations.

LandP		
idl	*name*	*geometry*
idl_1	n_1	g_1
idl_2	n_2	g_2
idl_3	n_3	g_3

Building		
idb	*idl*	*geometry*
idb_1	idl_1	g_4
idb_2	idl_2	g_5
idb_3	idl_3	g_6

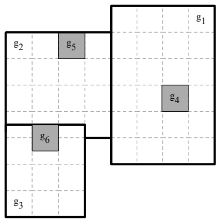


Figure 1. Example of a spatial database instance

proposed by ISO (2004) and the Open Geospatial Consortium (OGC) (OpenGis 1999). The analysis for topological relations between 3D spatial objects (Lee and Kwan 2005, Zlatanova *et al.* 2004) is left as future work. The set of topological relations that we consider here includes a subset of base relations (Egenhofer and Franzosa 1991, Randell *et al.* 1992), and derived relations such as Intersects, Within, and Contains, which are defined as a conjunction of base relations.

Table 1 provides the definitions of the topological relations used in this work, which were extracted from the OGC Simple Feature Specification (OpenGis 1999). In this table, given a geometry x, $\partial(x)$ indicates its boundary, and $dim(x)$ its dimension, where $dim(x)$ is equal to 0 if x is a point, 1 if it is a line, and 2 if it is a polygon.

Relation	**Definition**
Disjoint(x, y)	True if $x \cap y = \emptyset$
Touches(x, y)	True if $x \cap y \subseteq (\partial(x) \cup \partial(y))$
Equal(x, y)	True if $x = y$
Within(x, y)	True if $x \subseteq y$
Contains(x, y)	True if $y \subseteq x$
Overlaps(x, y)	True if $x \cap y \neq \emptyset$, $x \cap y \neq x \neq y$, and $dim(x \cap y) = dim(x) = dim(y)$
Crosses(x, y)	True if $x \cap y \neq \emptyset$, $x \cap y \neq x \neq y$, and $dim(x \cap y) < max(dim(x), dim(y))$
Intersects(x, y)	True if $x \cap y \neq \emptyset$

Table 1. Definition of topological relations by the Open Geospatial Consortium (OpenGis 1999)

Integrity constraints. Integrity constraints define valid states of a database. The work in this paper concentrates on a subtype of spatial semantic constraints described by Bravo and Rodríguez (2012), in particular, on topological dependency constraints (TDs), which impose the topological relations that should hold between two spatial objects of particular semantics (e.g. two land parcels should touch each other or be disjoint). Let us consider the following example to show the kind of constraints of interest for this work.

Example 3.2 (cont. Example 3.1) Taken the database of the previous example, we would like to enforce that attributes *idl* and *idb* are unique in both relations, and that the *idl* in Building is a subset of values in *idl* of LandP. These are traditional dependency constraints in relational databases. However, there are other constraints that cannot be captured by traditional relational constraints. For example, we need integrity constraints to ensure that land parcels with different *idl* cannot internally intersect (i.e. they can only be disjoint or touch) and that building blocks must be inside their respective land parcels. Following the notation given by Bravo and Rodríguez (2012), these constraints can be expressed by topological dependency constraints. These constraints impose topological relations (Egenhofer and Franzosa 1991, Randell *et al.* 1992) between geometries of spatial objects. □

More formally in first-order logic, let T be a topological relation or a conjunction of topological relations, $R(\bar{x}_1, y_1)$ and $P(\bar{x}_2, y_2)$ be relations of a database representing types of spatial objects (e.g. land parcel or building), where $\bar{x}_1$ and $\bar{x}_2$ are non-empty sequences of thematic attributes (i.e. non-spatial attributes), and y_1 and y_2 are spatial attributes. The topological constraints (TDs) addressed in this work are of the form:

$$\forall \bar{x}_1, \bar{x}_2, y_1, y_2 \ (R(\bar{x}_1, y_1) \wedge P(\bar{x}_2, y_2) \wedge \phi \rightarrow T(y_1, y_2)), \tag{1}$$

where ϕ is an optional conjunctive formula of the form $w_1 \neq z_1 \wedge \cdots \wedge w_l \neq z_l$ with $w_i \in \bar{x}_1$ and $z_i \in \bar{x}_2$, for all $i \in [1 \ldots l]$. The inclusion of ϕ is used to express constraints where R and P are the same predicate, and the variable must instantiate different tuples.

Example 3.3 Consider the database in Example 3.1, the following TDs express that land parcels should not internally intersect and that buildings should be within only the land parcel they belong to[1].

$$\forall l_1, l_2, y_1, y_2 (\mathsf{LandP}(l_1, y_1) \wedge \mathsf{LandP}(l_2, y_2) \ \wedge \ (l_1 \neq l_2) \rightarrow \mathsf{Touches}(y_1, y_2) \vee \mathsf{Disjoint}(y_1, y_2)) \tag{2}$$

$$\forall l_1, b_1, y_1, y_2 (\mathsf{LandP}(l_1, y_1) \ \wedge \ \mathsf{Building}(b_1, l_1, y_2) \rightarrow \mathsf{Within}(y_2, y_1)) \tag{3}$$

$$\begin{aligned} \forall l_1, l_2, b_1, y_1, y_2 (&\mathsf{LandP}(l_1, y_1) \wedge \mathsf{Building}(b_1, l_2, y_2) \wedge (l_1 \neq l_2) \\ &\rightarrow \mathsf{Touches}(y_1, y_2) \vee \mathsf{Disjoint}(y_1, y_2)) \end{aligned} \tag{4}$$

Let us consider now that instead of having geometries as illustrated in Figure 1, we have geometries as shown in Figure 2. Then, the database instance in Figure 2 is inconsistent due to the conflict between geometries with respect to the expected topological relations. In particular, g_1 and g_2, and g_2 and g_3, overlap when they should touch. Also, g_6 and g_2 partially overlap when they should touch or be disjoint.

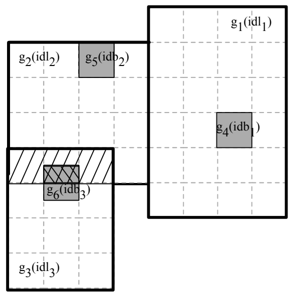


Figure 2. Example of an inconsistent spatial database instance (hatched areas represent overlapping areas and $g(k)$ denotes geometry g of tuple with key k)

□

TDs do not allow the expression of all possible forms of constraints over spatial data. For example, we cannot express that "a building must be inside of a land parcel", which is a spatial referential constraint (Bravo and Rodríguez 2012). Consequently, there could be a building that is not inside of any land parcel unless we impose another constraint

[1]This assumption might not be real in some territories, but it is just used for example purposes.

over the values of the attribute *idl* in Building. Despite the simplicity of topological dependency constraints, they are useful to express many meaningful constraints and there exist inconsistency measures with respect to these constraints in spatial databases (Brisaboa *et al.* 2014) that, as we show later, are useful to guide the data-cleaning process of an inconsistent spatial database.

In the formalization of integrity constraints, an interesting problem is the problem of *satisfiability*, that is, whether or not there exists a non-empty database instance that satisfies a particular set of integrity constraints. This problem is not only fundamental for spatial constraints but also for any type of integrity constraints because, before checking if a database is consistent with respect to a set of constraints, one needs to know if those constraints do not contradict each other.

The following example shows that for topological dependencies, a set of integrity constraints is not always consistent, and therefore, finding polynomial-time algorithms to check satisfiability becomes of interest (see Bravo and Rodríguez (2012) for details in checking satisfiability of topological dependency constraints).

Example 3.4 Consider the following constraint:

$$\forall l_1, l_2, y_1, y_2(\mathsf{LandP}(l_1, y_1) \land \mathsf{LandP}(l_2, y_2) \rightarrow \mathsf{Touches}(y_1, y_2) \lor \mathsf{Disjoint}(y_1, y_2))$$

This constraint will also be checked when $idl_1 = idl_2$ and therefore, it will not be satisfied because a geometry cannot touch or be disjoint with itself. □

In what follows, we will always assume to work with a satisfiable set of topological dependency constraints.

Inconsistency measures. The work by Brisaboa *et al.* (2014) introduces inconsistency measures to quantify the degree of consistency of spatial datasets with respect to topological dependency constraints. This work defines a violation degree between geometries, which is in agreement with the concept of partial consistency (Abdelmoty and Jones 1997). In particular, given an expected topological relation between geometries of two objects with particular semantics, a violation-degree measure quantifies how different is the topological relation between the geometries from the expected relation expressed by a topological dependency constraint.

In order to do that, these measures take into account the following criteria that were previously validated by Brisaboa *et al.* (2011) with human subject testing: (1) The *external distance* between disjoint geometries, which has an impact on conflicts risen by the separation of geometries when they must intersect. (2) The *overlapping area* of geometries that are internally connected (i.e. geometries for which hold any of the relations in Table 1 but Disjoint or Touches), which has an impact on conflicts when geometries must be externally connected (i.e. geometries that must touch). (3) The *crossing length* that represents the length of the minimum segment of a curve that crosses another curve or surface, which has an impact on conflicts when geometries must be externally connected.

As an example of these measures, consider the case when two surfaces g and g' are disjoint but they should touch each other. The degree of violation is computed as $\frac{D_{g,g'} \times L_{min}}{S_g}$, where $D_{g,g'}$ is the separation between g and g', L_{min} is the minimum length at the scale of representation of the dataset, and S_g is the area of g. Note that L_{min} is used to transform a distance to an area, and S_g is used to normalize the result to the range $[0, 1]$.

4. Rank-based strategies for data cleaning

In the following subsections, we introduce a formal framework for the data-cleaning process of spatial databases based on a graph-based representation of inconsistencies and then, a practical approximation approach to this process. Recall from the introduction that we assume an oracle-based process in which one geometry is repaired at a time in accordance with an orthophoto (or any other oracle assumed to be the truth). The graph-based modeling of the problem simplifies its understanding and allows a more formal reasoning about it. In particular, it formally shows the complexity of solving this problem and provides alternative approximation approaches. See Appendix A for a theoretical analysis of the problem and the proof that it is a computational hard problem.

4.1. *Graph-based representation of spatial inconsistency*

A database instance D violates a constraint of the form (1) when there are tuples $(a_1, \ldots, a_n, g_1) \in R$ and $(b_1, \ldots, b_m, g_2) \in P$ in D such that $R(a_1, \ldots, a_n, g_1) \wedge P(b_1, \ldots, b_m, g_2) \wedge \phi$ is true but $T(g_1, g_2)$ is false. We say in this case that g_1 and g_2 are in conflict with respect to the topological relation T. More formally, and to simplify notation, we introduce a logical formula $\mathit{Confl}_{D,R,P,\psi}(t_1, t_2)$ that is true when tuples $t_1 = (a_1, \ldots, a_n, g_1)$ and $t_2 = (b_1, \ldots, b_m, g_2)$ in the database instance D are in conflict with respect to a topological dependency ψ of the form (1) with topological relation T.

Definition 4.1: Let D be an inconsistent database instance with n tuples and Ψ be a set of integrity constraints of the form (1). The *inconsistency graph* of D is an undirected graph $G_{D,\Psi} = (V, E)$, where (i) V is a set of nodes that identify tuples in the database instance; (ii) $E = \{\{t_i, t_j\} | t_i, t_j \in V \wedge \exists(\psi \in \Psi \wedge P, R \in D)(\mathit{Confl}_{D,P,R,\psi}(t_i, t_j))\}$; and (iii) each $t_i \in V$ has a degree greater or equal to 1. By imposing condition (iii), inconsistency graphs do not have isolated nodes.

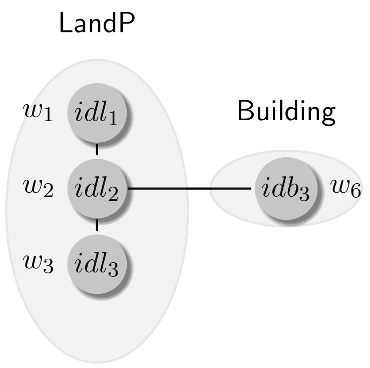
□

Each node in an inconsistency graph can be annotated with a weight assessing its importance in the quality of the dataset. In particular, we are interested in weighting schemes that represent the potential benefit of modifying the geometry associated with the node in the quality of the dataset. These weights can be defined in different ways. Without having any extra knowledge about the semantic relevance of spatial objects or contextual information, in this paper we propose the use of the inconsistency measures presented in Brisaboa *et al.* (2014). The measures in Brisaboa *et al.* (2014) were normalized in order to obtain an overall quality measure of the database, where each geometry counts at most up to 1.0 in the aggregation of the violation degree of all geometries in the database. Because in this work we are not interested in an overall quality measure, we omit the normalization of the measures.

In addition, we propose a more simple weighting schema based on the degree of the node in the inconsistency graph. This schema relies on the idea that geometries that participate in a larger number of conflicts should be first modified because this will most likely eliminate more than one conflict and, in consequence, reduce the number of geometries to be processed. Indeed, this schema is a simplification of the inconsistency measures in Brisaboa *et al.* (2014).

Example 4.2 Consider the database instance and integrity constraints in Example 3.3. All geometries in this database are polygons. This instance is inconsistent because land parcels with geometries g_1 and g_2, and g_2 and g_3 overlap when they should touch. In addition, the building with geometry g_6 partially overlaps the land parcel with geometry g_2.

A graph representation of the inconsistent database instance is shown in Figure 3. This graph contains a set V of nodes, one for each tuple that participates in one or more conflicts.



Node	Weight	Degree	IM
idl_1	w_1	1	4
idl_2	w_2	3	7.5
idl_3	w_3	1	3.5
idb_3	w_6	1	0.5

Figure 3. A graph-based representation of an inconsistent spatial database (left) and the weights assigned with the two different weighting schemes: degree and inconsistency measures (right)

The variable w_i next to each node can take different values depending on the type of weights as the table on the right of Figure 3 indicates. The calculations of inconsistency measures are given in square units represented in the underline light grid in Figure 2. The degree is derived from the graph. For the inconsistency measures, when geometries that overlap should touch (e.g. land parcels), the area of overlapping between geometries in conflict is the violation degree of inconsistency. When a geometry that partially overlaps should be within another geometry (e.g. a building within a land parcel), non-included area is then the violation degree of inconsistency[1]. More precisely, the weight w_2 associated with idl_2 is calculated (using inconsistency measures) as:

$$w_2^{IM} = |g_2 \cap g_3| + |g_2 \cap g_1| + |g_6| - |g_2 \cap g_6|,$$

with $|\,|$ denoting the area of a geometry and $\cap$ the spatial intersection of geometries. □

Let D be a database instance with a TD ψ of the form (1) such that R and P are relational predicates in ψ constrained by a topological relation T. If $Confl_{D,R,P,\psi}((\bar{u}_1, g_1), (\bar{u}_2, g_2))$ is true, it is possible to restore the consistency of D by modifying g_1, g_2, or both to make $T(g_1, g_2)$ true. Instead of studying the geometric transformations needed to restore consistency, we propose strategies that guide the cleaning process of a database by defining an ordered sequence of geometries that should be modified to totally or partially restore the consistency of a spatial database.

[1]Table 4 in Brisaboa *et al.* (2014) summarizes all the inconsistency measures for surfaces. Recall that, unlike Brisaboa *et al.* (2014), in this work we do not normalize inconsistency measures to the range $[0, 1]$.

4.2. *Practical-approximation approach to the cleaning process*

In the data-cleaning process, we have to find a minimum ordered set of geometries whose modification restores the consistency of the database. This is a one-geometry-at-a-time process, where the order in which geometries are taken is considered to be a ranking.

In Appendix A we formalize the problem of determining a minimum set of geometries that restore consistency and show that this is a NP-Complete problem[1] via a reduction of the vertex cover problem to our problem. As consequence, we should apply approximation algorithms. Next we propose practical heuristics inspired by well-known theoretical approximation algorithms for the vertex cover problem.

For our problem, we want to get an ordered set of nodes such that the modification of geometries in this order increases rapidly the quality of the database and minimizes the number of changes needed to restore consistency. This ordered set is even more important if, for some reason, it is impossible to repair the whole dataset. Unfortunately, geometries that should be modified are unknown a priori (they are only recognized once the orthophoto is inspected on a one-by-one geometry analysis). So, the only possible automatic optimization is to guess from the inconsistency graph which geometries should be modified, being one or both of the end nodes of edges in the graph. This suggests an heuristic-based process to obtain the smallest set of geometries that should be modified: select first geometries that have a greater impact on the quality of the database.

In practice, the modification of geometries in order to restore consistency depends on the correctness of the geometry with respect to an orthophoto. Thus, no all geometries that participate in conflicts should be modified even if their modification may create a consistent database. To make it clearer, consider two geometries that overlap when they should touch. To restore consistency, one can theoretically modify one of both geometries; however, by using the orthophoto, only one of these alternatives can be used. Indeed, it can be the case that after the modification of a geometry, this geometry remains in conflict with another geometry that should be also modified to eliminate the edge (i.e. the conflict) from the inconsistency graph. The following example illustrates this case.

Example 4.3 (cont. Example 4.2) Assume that the orthophoto associated with the inconsistent instance of Example 4.2, as shown in Figure 4(a), reveals that the correct instance should be as shown in Figure 4(b). An approximation algorithm according to the degree of nodes of the initial inconsistency graph will select first geometry g_2 of the tuple with key idl_2 and degree 3 to be modified. However, when modifying geometry g_2 based on the orthophoto, the conflict between g_2 and g_3 is not completely corrected. □

We consider first a simple approximation algorithm called St_Cleanning, which is shown in Figure 5. This is an approximation algorithm for data cleaning that uses a priority queue of tuples (i.e. nodes in the inconsistency graph) whose geometries must be modified but whose priority, defined by weights, does not change along the process. Although we later present a more effective dynamic version of this strategy, we introduce first the static algorithm because it is simpler to understand and less computationally expensive.

The following example illustrates the applicability of the algorithm.

Example 4.4 (cont. Example 4.3) Consider the database instance in Figure 4(a) with inconsistency graph and weights in Figure 3, and the associated orthophoto in Figure 4(b).

[1]A problem for which there is no known algorithm that can decide in polynomial time if a set of geometries is the minimum set needed to restore consistency.

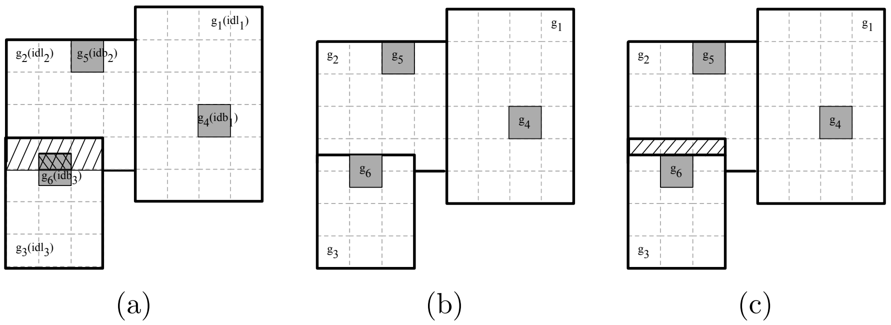


Figure 4. Selection of geometries – partial correction: (a) inconsistent instance, (b) consistent instance according to orthophoto, and (c) inconsistent instance after modification of geometry g_2

```
1  Algorithm: St_Cleanning
2  Input: An inconsistency graph $G = (V, E)$ of $D$, a max-Heap $Q$ priority queue of nodes in $G$ ordered by
         weight, and the set of TDs $\Phi$.
3  Output: $C$ is a ranking of nodes that indicates the order in which they should be repaired.
4  $C \leftarrow \emptyset$;
5  $E' \leftarrow E$;
6  while $E' \neq \emptyset$ do
7      $u \leftarrow delete\_max(Q)$;
8      if $exists\ e \in E'\ such\ that\ u\ is\ an\ end\ node\ of\ e$ then
9          //Manually repair $u$
10         $C \leftarrow C \cup \{u\}$;//Append $u$ to the ranking $C$
11         foreach $v \in direct\_neighbors(G, u)$ do
12             Check conflict between tuples $u$ and $v$ with respect to TDs in $\Phi$;
13             if $there\ is\ no\ conflict\ between\ u\ and\ v$ then
14                 $E' \leftarrow E' \setminus \{(u, v)\}$;
15 return
```

Figure 5. A “static” approximation algorithm for the total cleaning sequence

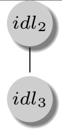
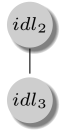
Iteration	Priority Queue	Graph
1	$\{\boldsymbol{idl_2}, idl_1, idl_3, idb_3\}$	
2	$\{\boldsymbol{idl_1}, idl_3, idb_3\}$	
3	$\{\boldsymbol{idl_3}, idb_3\}$	Empty

Table 2. Execution of Algorithm St_Cleanning for the database instance in Example 4.4

The steps followed by Algorithm St_Cleanning using the inconsistency measure-based weights are show in Table 2, where the priority queue is at the beginning of the iteration and the graph is at the end of the iteration (thus, idl_1 and idb_3 do not appear as they are removed from the graph during the first iteration).

As the example shows, the priority queue is only modified by eliminating at each iteration the node with highest priority (highlighted with bold font). The algorithm, however, checks at each iteration if there are still conflicts in the database. Note also that, at each iteration, it is necessary to check if the node with highest priority is still inconsistent (line 8), as this may not be the case due to previous repairs of other elements.

For example, in the second iteration, idl_1 is already consistent due to the repair of idl_2 in the previous iteration. Thus, the algorithm wastes an iteration. □

In each cycle the algorithm takes one node from the priority queue. Using an implementation of graphs with adjacency lists, checking for edges of an end node that should be covered (i.e. eliminated from the inconsistency graph) is linear with respect to the number of neighboring nodes. Even more, an edge is revised at most twice, one for each end node, so there are at most $|V|$ cycles with at most $2|E|$ revisions of edges. This leads to a time cost of $O(|V| + |E|)$, which matches the complexity of the 2-approximation algorithm for vertex cover. We now prove that this algorithm is also a 2-approximation. In other words, this algorithm (in the worst case) will find a set that is at most twice the minimum set needed to eliminate inconsistencies from the database, which is equivalent to isolating all the nodes in the inconsistency graph.

Proposition 4.5: Algorithm St_Cleanning finds a set that has, in the worse case, twice the number of elements of a minimum set needed to isolate all the nodes in the inconsistency graph.

Proof: Let A be the set of edges that are selected to make condition in line 8 true. Note that this is the set that makes the condition for the *while* in line 6 true. In order to eliminate these edges, any optimal set of nodes must include at least one of the end nodes of the edge. Let C^* be the size of the optimal set, then $C^* \geq |A|$. For each of the edges in the set A, we get at most 2 nodes in the set determined by the algorithm. Thus, let C be the size of the set determined by the algorithm, then $C \leq 2|A|$. Combining both expressions, $C \leq 2C^*$, thereby proving the theorem.

□

A second alternative to select geometries for modification in the cleaning process uses a dynamic priority queue, that is, a priority queue where weights of nodes are updated in each iteration of the cleaning process. The strategy follows a simple procedure to guide the data-cleaning process: i) take the node with highest weight, ii) modify (if necessary) the corresponding geometry, iii) recompute the weight of all nodes in its adjacency list (i.e. direct neighbors), iv) iterate until consistency has been restored (or resources dedicated to the process have been exhausted). Notice that the static version of the algorithm described above can be devised as a restricted version of this strategy in which the third step is omitted. Figure 6 presents this dynamic strategy. The algorithm is described in its more general form, which corresponds with the use of inconsistency measures. In Appendix B, we describe some optimizations that may apply for simpler weighting schemes, such as degree.

Example 4.6 (cont. Example 4.4) Table 3 illustrates the steps followed by the dynamic strategy for the same database instance in Example 4.4. This strategy needs one iteration less than the static one because idl_1 is removed from the priority queue after the first iteration. In other words, the dynamic strategy detects in advance some elements that do not need to be considered any more, thus avoiding idle iterations. This is not the only improvement of the dynamic strategy, which, in general, reorders the priority queue to demote the elements which inconsistencies were partially (or totally) repaired in the iteration.

□

In Appendix B we provide an efficient implementation of this strategy and show that each iteration (lines 7-19) takes $O(n_u \log |V|)$ time, where n_u is the number of neighbors of the node processed in that iteration (i.e. the node with highest priority). In addition, the

```
Algorithm: Dy_Cleanning
Input: An inconsistency graph $G = (V, E)$, a max-Heap $Q$ priority queue of nodes in $G$ ordered by weight.
Output: $C$ is a ranking of nodes that indicates the order in which they were repaired.
$C \leftarrow \emptyset$;
$E' \leftarrow E$;
while $E' \neq \emptyset$ do
    $u \leftarrow delete\_max(Q)$;
    //Manually repair $u$
    $C \leftarrow C \cup \{u\}$;//Append $u$ to the ranking $C$
    $g^u \leftarrow$ repaired geometry associated with $u$;
    foreach $v \in direct\_neighbors(G, u)$ do
        Check conflict between tuples $u$ and $v$ with respect to TDs in $\Phi$;
        $repinc \leftarrow$ inconsistency between $v$ and $g^u$;
        $G[v].weight \leftarrow G[v].weight - (G[v][u].weight - repinc)$;
        if there is no conflict between $u$ and $v$ then
            $E' \leftarrow E' \setminus \{(u, v)\}$;
        else
            $G[v][u].weight \leftarrow repinc$;
        $decrease\_key(G[v].weight, v, Q)$;
return
```

Figure 6. A "dynamic" data-cleaning strategy

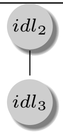
Iteration	Priority Queue	Graph
1	$\{\boldsymbol{idl_2}, idl_1, idl_3, idb_3\}$	
2	$\{\boldsymbol{idl_3}, idb_3\}$	Empty

Table 3. Execution of Algorithm Dy_Cleanning for the database instance in Example 4.4

proof of the approximation ratio of St_Cleanning can be easily adapted for the dynamic case, so we can state the following proposition:

Proposition 4.7: Algorithm Dy_Cleanning finds a set that has, in the worse case, twice the number of elements of a minimum set of nodes needed to isolate all the nodes in the inconsistency graph.

The proposed dynamic strategy does not work in the general case, where new conflicts might appear due to the modification of an object. The following example illustrates this case.

Example 4.8 Consider geometries shown in Figure 7(a) and assume that a topological dependency constraint specifies that these geometries should be disjoint or touch. Then, these geometries are inconsistent since g_2 overlaps g_3. Now consider that the algorithm selects g_2 to be modified and this modification produces geometries in Figure 7(b) based on an orthophoto in Figure 7(c). Then, after modification of g_2, a new conflict between g_1 and g_2 appears.

□

An algorithm for the general case is costly because it requires to check for all possible new conflicts. Note, however, that even though new conflicts might appear, a geometry that is repaired will never be modified again (because the orthophoto is assumed to be the truth). Thus, this leads to an $O(n^2)$ algorithm, with n the number of tuples in the database. We do not include this algorithm in our experimental evaluation, because the appearance of new conflicts in a data-cleaning process guided by our strategy is not very likely. Notice that these are "conflicts hidden by other conflicts" (i.e. incorrect objects

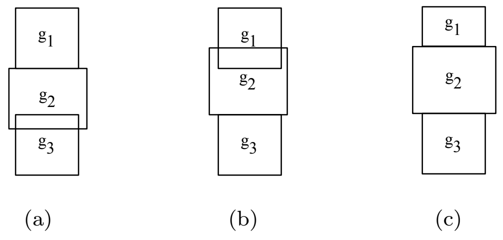


Figure 7. Example where new conflicts appear in the cleaning process: (a) original inconsistent database, (b) inconsistent database after g_2 modification, and (c) final consistent database

that are not in conflict at the beginning of the process but that, after the repair of some other incorrect object, become inconsistent with it). Indeed, it is the case in our experimental evaluation that Dy_Cleanning completely repairs the database, so no such hidden conflicts exist. A deeper study of this kind of conflicts, and the proposal of an optimized algorithm for such scenarios is left as an open problem.

5. Experimental evaluation

In this section we evaluate the static and dynamic strategies presented in the previous section, and also the different alternatives to assign weights to nodes in the inconsistency graph. Let us first recap all the evaluated heuristics:

- bl_random: baseline solution that emulates no-expertise at-all.
- bl_area: baseline heuristic relying on the idea that larger objects are better candidates to restore consistency. Although this heuristic fits in the general schema presented in Section 4 (the weight of each node corresponds to the area of its geometry), we include it as a baseline because of its simplicity.
- st_degree: variant of the static strategy in which the weight of each node corresponds to its degree in the inconsistency graph.
- dy_degree: dynamic version of the previous one.
- st_measures: variant of the static strategy in which weights are assigned according to inconsistency measures adapted from Brisaboa *et al.* (2014).
- dy_measures: dynamic version of the previous one.

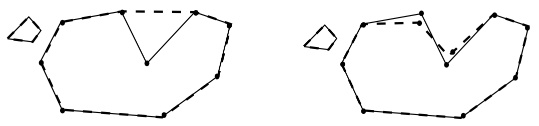
All these heuristics provide a ranking according to which the objects in the dataset should be repaired (or checked for correctness). Obviously, if we go through the whole ranking, the dataset ends being correct, but not with the same number of iterations among the different ranking strategies. Thus, the goal of this evaluation is to show how fast can each heuristic obtain a correct dataset (or which heuristic improves quality the most in less iterations). Note that we assume the existence of integrity constraints on the datasets, so not all the objects in the dataset have to be checked but just those that violate some integrity constraints. The use of integrity constraints also applies for the baseline solutions (i.e. our baselines do not check all the objects but just those violating some integrity constraints). It is also important to recall that an object violating an integrity constraint may not contain any errors (because its inconsistency may be due to other geometries). One final clarification is that baseline strategies (bl_random and bl_area) can also have static and dynamic variants. In our experiments we just include the dynamic versions, which show better performance.

In the following experiments we use a real dataset consisting of 1,024 county subdivisions in the state of New York. This dataset is part of the TIGER/Line Shapefile from the U.S. Census Bureau[1], which is known for not containing inconsistencies in the data. We introduce inconsistencies in the dataset by means of a simplification algorithm available in the GeoTools Toolkit[2], an open source Java library that provides tools for geospatial data analysis and processing. Using this simplification algorithm we generate four inconsistent datasets, each of which contains errors in the 5%, 10%, 25%, and 50% of the objects, respectively. Modified objects were selected uniformly at random in such a way that the set of modified objects in each dataset is a subset of the modified objects in the subsequent dataset. For example, modified objects in the 5% dataset are also part of the modified ones in the 10% dataset. The amount of error introduced in each object can be easily controlled by the tolerance applied by the simplification algorithm (see Douglas and Peucker (1973) for further details about the parameters of this algorithm). We randomly select the tolerance applied to each object in the set {0.0001, 0.001, 0.01}. In this way, each dataset contains objects with different degree of error.

In order to compare the different heuristics, we first need to define a measure of quality of datasets. We could use the inconsistency measures presented in Brisaboa *et al.* (2014) for this purpose, but this could benefit the st_measures and dy_measures heuristics (indeed, this is the case and we do not include these results in the paper). As we know the correct (original) dataset, we can define more objective measures of the quality of a dataset. Notice that these measures cannot be used in a real data-cleaning process as they need to know the correct dataset in advance. In this evaluation, the original dataset plays the role of an oracle upon which we can define quality measures whose explanations follow. The original dataset is also used to repair (substitute) the geometries in the execution of the strategies.

We evaluate quality as the inverse of the amount of error of the inconsistent versus the original database (oracle), and we explore three different measures of error: (i) the number of objects in the inconsistent dataset that are not equal to its corresponding object in the original dataset, (ii) the overall number of points in the representation of geometries that do not correspond (or that are missing) to the points in the original dataset, and (iii) the overall difference of area between the inconsistent and the original dataset. We use the following example to illustrate that these three evaluations represent very different ideas, and thus, show different behaviors.

Example 5.1 Consider the following two datasets, each of them containing an original (correct) versus a modified (incorrect) geometry. In each dataset, geometries drawn with continuous line are the original ones.



(a) Dataset 1 (b) Dataset 2

Figure 8. Illustration of differences between evaluations

One of the simplest possible evaluations is based on the number of objects in the dataset that contain errors (i.e. objects that are not equal to its corresponding object

[1]http://www.census.gov/geo/maps-data/data/tiger.html
[2]GeoTools (http://www.geotools.org)

in the original dataset). In the example of Figure 8 this evaluation considers that the quality of both datasets is the same, which may not be realistic if we try to relate it with the cost of manually repairing them. However, this simple evaluation provides a preliminary distinction between the different strategies.

Our second definition of quality is based on the overall number of points of geometries in the inconsistent dataset that do not correspond (or that are missing) with points of geometries in the original dataset. Given an original geometry $g_o = \{g_{o,1}, g_{o,2}, \ldots, g_{o,n}\}$ and its corresponding modified geometry $g_m = \{g_{m,1}, g_{m,2}, \ldots, g_{m,n'}\}$, where n may not be equal to n', we define the difference (error) between these two geometries as $(g_o \cup g_m) \setminus (g_o \cap g_m)$, where $\cup, \cap$, and $\setminus$ represent set operations union, intersection, and difference, respectively. The example in Figure 8 shows a case with lower error according with this definition (Dataset 1) and higher error (Dataset 2).

Our last definition is based on the difference of areas between geometries. Given an original geometry g_o and its corresponding modified geometry g_m, the area-based error is equal to $area((g_o \cup g_m) \setminus (g_o \cap g_m))$, where $\cup, \cap$, and $\setminus$ represent spatial operations union, intersection, and difference, respectively. Unlike previous example, this definition considers that the quality of Dataset 1 is lower than the quality of Dataset 2. $\square$

5.1. *Evaluation based on number of incorrect objects*

Figure 9 shows the results of the experimentation with this evaluation. Y-axes show error (determined by the number of geometries that differ with respect to the original database) and x-axes show iteration in the cleaning process (note that each unit represents 5 iterations). In these graphs, the most rapidly decreases the error the better.

A first conclusion is that the use of rank-based strategies clearly improves the data-cleaning process, as it drastically reduces the number of geometries to be processed to about a 20%. For example, in the 5% dataset, dynamic strategies repair the whole dataset in 10 iterations, which is less than the 20% of the 51 inconsistent objects at the beginning of the process. This improvement is consistent in the other datasets.

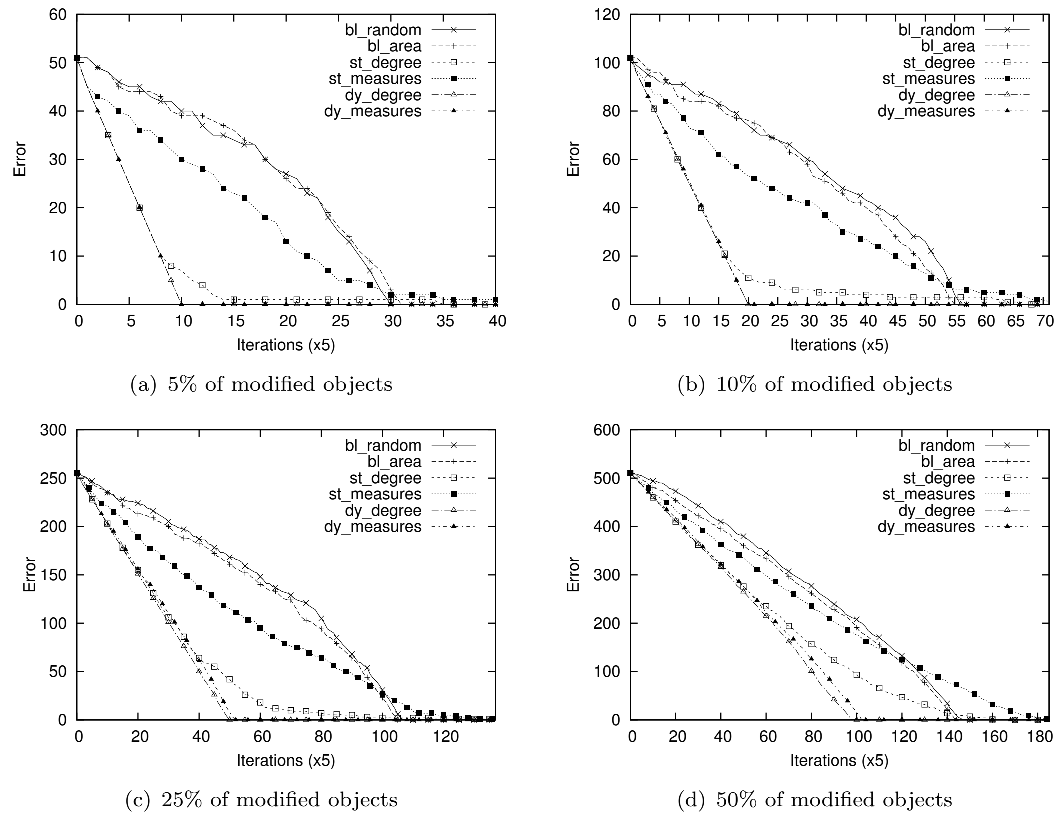
A second conclusion is that baseline solutions are far from being competitive, especially in the datasets with fewer number of objects with error. The dataset with 50% of modified objects is shown for completeness but it is not realistic because real datasets usually contain much lower percentage of objects with errors. Although baseline solutions correct all the objects in less iterations than the two static strategies (because they are dynamic and recompute the ranking at each iteration), they need more iterations to improve the quality of the dataset significantly.

Among the static strategies, st_degree performs much better than st_measures. A surprising result is that st_degree performs similar to the more complex (and computationally expensive) dynamic strategies. Note that all of them achieve an improvement of quality of about 80% (with respect to the initial error of the dataset) in almost the same number of iterations. However, the improvement of the remaining 20% is much slower for the st_degree.

Our last conclusion is that both dynamic strategies perform similar and outperform all other strategies.

5.2. *Evaluation based on number of incorrect points*

This evaluation is inspired in the cost of repairing objects with errors, because the larger this error is, the larger the number of points that should be modified. It should be clear



(a) 5% of modified objects

(b) 10% of modified objects

(c) 25% of modified objects

(d) 50% of modified objects

Figure 9. Evolution in the quality of the database based on incorrect objects

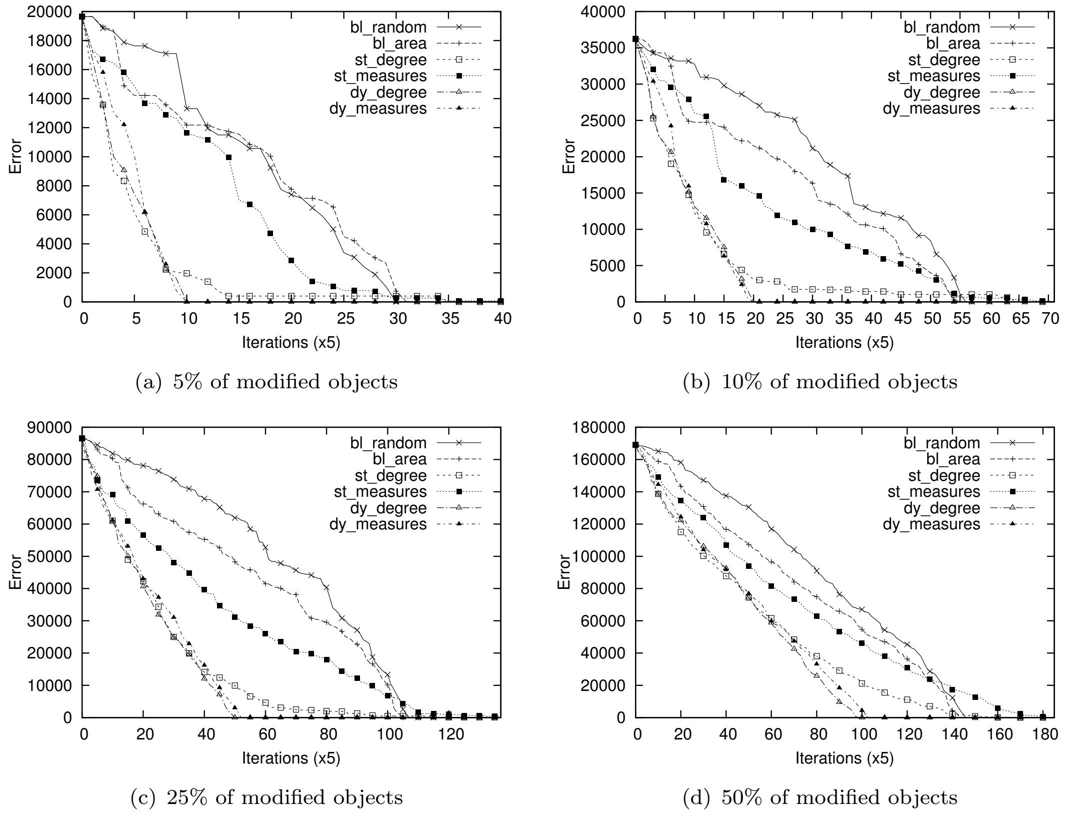
that the example in the left part of Figure 8 would take less time to be corrected than the example on the right because it needs to insert, eliminate, or translate less number of points. Figure 10 shows the results obtained from this evaluation.

These results are similar to the ones obtained in the previous evaluation. Dynamic strategies outperform all other strategies and, although there are some differences between them, these are not significant. The st_degree behaves much better than the other static alternative, and it is worse than the dynamic strategies just for the last part of the cleaning process. This means that it could be an alternative (due to its simplicity) in scenarios where the goal is to obtain an acceptable quality, but it is not necessary to repair the dataset completely. Baseline solutions are not competitive in this scenario either, although it can be observed that bl_area performs significantly better than the random approach.

5.3. *Evaluation based on difference of areas*

This evaluation is inspired in a subjective perception of quality. In the cognitive validation of the factors that affect the degree of violation of objects with respect to a topological relation (which is a sort of definition of quality) presented in Brisaboa *et al.* (2011), overlapping area was identified as one of the factors that matter for most users. The definition of quality based on the difference of areas presented above captures and adapts the same idea. Figure 11 shows the results of this experiment.

In this scenario, strategies based on the inconsistency measures presented in Brisaboa *et al.* (2014) clearly outperform those based on the degree. This result is not surprising



(a) 5% of modified objects

(b) 10% of modified objects

(c) 25% of modified objects

(d) 50% of modified objects

Figure 10. Evolution of the quality of the database. Evaluation based on incorrect points

as those inconsistency measures consider areas in their definition. A surprising result is that even the static variant outperforms those based on the degree (although dy_degree repairs the whole dataset in less iterations than st_measures). It should be noticed that dy_measures is remarkable better than any other strategy, especially in the 5% dataset. After 25 iterations, the improvement achieved by dy_measures is about 95%, compared with the 75% and the 60% achieved by st_measures and dy_degree, respectively. Therefore, dy_measures is a clear choice in applications where the quality of the dataset is defined in terms of areas (i.e. in scenarios where the human perception of quality does matter).

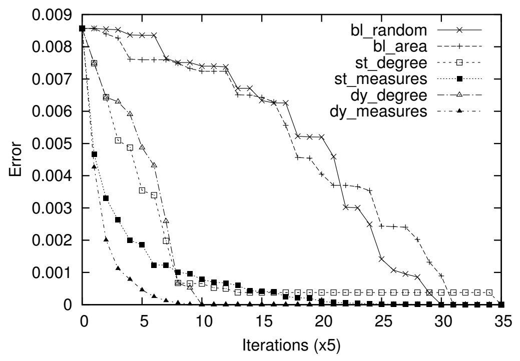
5.4. *Time analysis*

As a proof of concept, we compare the time performance of dynamic strategies. All the experiments were performed in an Intel Core 5@1.7GHz, 4GB RAM, running OS X 10.9.1. We compiled with Java version 1.6. Table 4 shows the times of the dynamic strategies using degree and inconsistency measures.

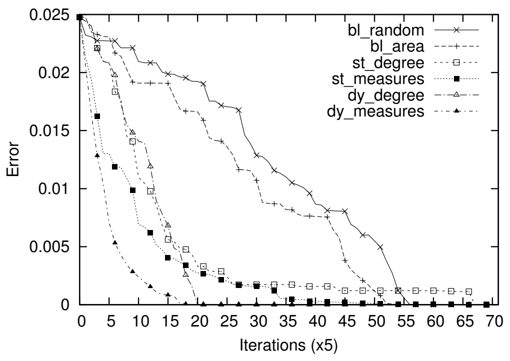
Strategy	**5%**	**25%**	**50%**
dy_degree	1989	7330	12894
dy_measures	2205	7518	13432

Table 4. Time performance of dynamic strategies (ms)

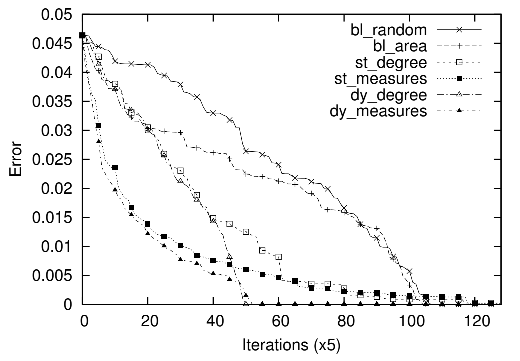
These results show that the use of inconsistency measures makes the strategy just



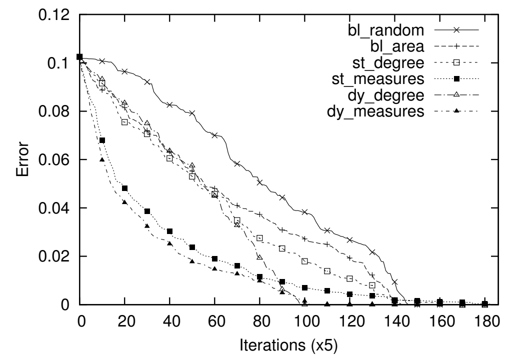
(a) 5% of modified objects



(b) 10% of modified objects



(c) 25% of modified objects



(d) 50% of modified objects

Figure 11. Evolution of the quality of the database. Evaluation based on difference of areas

slightly slower, being both alternatives in the same order of magnitude. If we compare time per iteration, dynamic strategies perform in the range [30-50] ms/iteration, which is competitive with the [10-20] ms/iteration of static strategies. Indeed, the difference would be imperceptible by humans if we integrate these strategies in a geographic information system.

6. Conclusions and future work

We have proposed and evaluated strategies that support data-cleaning of inconsistent spatial databases with respect to a set of integrity constraints that impose topological relations between spatial objects. These strategies are based on an inconsistency graph that models the violation of the integrity constraints defined in the database. The basic idea is to use information of such graph to rank geometries in a spatial dataset that should be modified in order to improve the quality of the database. We explore two kinds of weights assigned to nodes in the graph: the degree of each node and inconsistency measures as defined in Brisaboa *et al.* (2014); and compare them with a random-order baseline solution and with a naive weighting schema that uses the area of the geometry associated with the node. In addition, for each of them we compare a static and a dynamic approach.

Our experimental evaluation shows that ranked-based strategies are suitable to decrease up to 80% of revised objects to be modified in a cleaning process (i.e. only a 20% of the inconsistent objects have to be processed). In detail, the experiments show that heuristics based on the dynamic strategy clearly outperform those based on the static one. The dynamic variants based on both inconsistency measures and degrees in the inconsistency graph perform similarly. The only significant difference is when the quality of the database is evaluated in terms of differences of areas between correct and incorrect objects, in which case the dy_measures clearly outperform all the other heuristics.

The next step of our work is to evaluate the different strategies in a real production setting. In order to do that we need to spend more time on engineering the implementation of the strategies and integrating them into a geographic information system. Then, this prototype could be evaluated by specialists on the repair of spatial databases. Finally, we also leave as a future work the analysis of the data-cleaning process for topological relations between 3D spatial objects (Lee and Kwan 2005, Zlatanova *et al.* 2004).

Acknowledgement(s)

This work was partially funded by CONICYT Fondecyt *regular* 1140428 and *iniciación* 11130377 (Chile), by "Ministerio de Ciencia e Innovación" - Spain (PGE and FEDER) grant TIN2009-14560-C03-02 and by "Xunta de Galicia (FEDER)", grant GRC2013/053.

References

Abdelmoty, A.I. and Jones, C.B., 1997. Towards Maintaining Consistency of Spatial Databases. *In*: *Proceedings of the Sixth International Conference on Information and Knowledge Management (CIKM'97)* ACM, 293–300.

Borges, K.A.V., *et al.*, 2002. Integrity Constraints in Spatial Databases. *In*: *In Database Integrity: Challenges and Solutions* Ideas Group.

Bravo, L. and Rodríguez, M.A., 2009. Semantic Integrity Constraints for Spatial Databases. *In*: *Proceedings of the 3rd Alberto Mendelzon International Workshop on Foundations of Data Management (AMW)*, Vol. 450 of *CEUR Workshop Proceedings* CEUR-WS.org.

Bravo, L. and Rodríguez, M.A., 2012. Formalization and reasoning about spatial semantic integrity constraints. *Data Knowl. Eng.*, 72, 63–82.

Brisaboa, N., Rodríguez, M.L.M.A., and Seco, D., 2014. An Inconsistency Measure of Spatial Data Sets with respect to Topological Constraints. *International Journal of Geographic Information Science*, 28 (1), 56–82.

Brisaboa, N.R., Luaces, M.R., and Rodríguez, M.A., 2011. Cognitive Adequacy of Topological Consistency Measures. *In*: *ER Workshops*, Vol. 6999 of *Lecture Notes in Computer Science* Springer, 241–250.

Chaudhuri, S., Ganti, V., and Kaushik, R., 2006. A Primitive Operator for Similarity Joins in Data Cleaning. *In*: *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006* IEEE Computer Society, p. 5.

Cockcroft, S., 1997. A Taxonomy of Spatial Integrity Constraints. *GeoInformatica*, 1 (4), 327–343.

Cormen, T., *et al.*, 2009. *Introduction to Algorithms.* MIT Press.

Davis, C.A., Borges, K.A.V., and Laender, A.H.F., 1999. Spatial Integrity Constraints in Object Oriented Geographic Data Modeling. *In*: *ACM-GIS*, 1–6.

Davis, C.A., Borges, K.A.V., and Laender, A.H.F., 2005. Deriving Spatial Integrity Constraints from Geographic Application Schemas. *Encyclopedia of Database Technologies and Applications*. Idea Group, 176–183.

del Mondo, G., *et al.*, 2013. Modeling Consistency of Spatio-Temporal Graphs. *Data Knowl. Eng.*, 84 (1), 59–80.

Deng, M., bao Liu, W., and zhi Feng, X., 2005. Dealing with Inconsistency between Digital Geographic Lines from Multi-data Sources in GIS. *Journal of Remote Sensing*, 9 (4), 343–348.

Deng, M., *et al.*, 2003. Modelling Error Propagation for Spatial Consistency. *Journal of Geospatial Engineering*, 5 (2), 51–60.

Douglas, D.H. and Peucker, T.K., 1973. Alogrithms for the reudction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10 (2), 112–122.

Egenhofer, M. and Franzosa, R., 1991. Point Set Topological Relations. *International Journal of Geographical Information Science*, 5, 161–174.

Egenhofer, M. and Shariff, A., 1998. Metric Details for Natural-Language Spatial Relations. *ACM Transactions on Information Systems*, 16 (4), 295–321.

Egenhofer, M.J. and Dube, M.P., 2009. Topological relations from metric refinements. *In*: *17th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems* ACM, 158–167.

Godoy, F.A. and Rodríguez, M.A., 2004. Defining and Comparing Content Measures of Topological Relations. *GeoInformatica*, 8 (4), 347–371.

Güting, R., 1994. An Introduction to Spatial Database Systems. *VLDB Journal*, 3, 357–399.

Hadzilacos, T. and Tryfona, N., 1992. A Model for Expressing Topological Integrity Constraints in Geographic Databases. *In*: *Spatio-Temporal Reasoning*, Springer LNCS 639, 252–268.

Hunter, A. and Konieczny, S., 2005. Approaches to Measuring Inconsistent Information. *In*: *Inconsistency Tolerance*, Vol. 3300 of *Lecture Notes in Computer Science* Springer, 191–236.

ISO, 2004. *ISO 19125-1:2004 Geographic information – Simple feature access – Part 1: Common architecture.* Technical report, International Organization for Standardization.

Jin, L., Li, C., and Mehrotra, S., 2003. Efficient Record Linkage in Large Data Sets. *In*: *Eighth International Conference on Database Systems for Advanced Applications (DASFAA '03)* IEEE Computer Society, 137–.

Lee, J. and Kwan, M.P., 2005. A combinatorial data model for representing topological relations among 3D geographical features in micro-spatial environments. *International Journal of Geographical Information Science*, 19 (10), 1039–1056.

Martinez, M.V., *et al.*, 2007. How Dirty Is Your Relational Database? An Axiomatic Approach. *In*: *Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 9th European Conference, ECSQARU 2007, Hammamet, Tunisia*, 103–114.

Mäs, S., 2007. Reasoning on Spatial Semantic Integrity Constraints. *In*: *COSIT*, Springer LNCS 4736, 285–302.

Mehta, D.P. and Sahni, S., 2004. *Handbook Of Data Structures And Applications (Chapman & Hall/Crc Computer and Information Science Series.)*. Chapman & Hall/CRC.

OpenGis, 1999. *Opengis Simple Features Specification for SQL.* Technical report, Open GIS Consortium.

Ordonez, C., García-García, J., and 0002, Z.C., 2007. Measuring referential integrity in distributed databases. *In*: *Proceedings of the First Workshop on CyberInfrastructure: Information Management in eScience, CIMS 2007, Sixteenth ACM Conference on Information and Knowledge Management, CIKM 2007, Lisbon, Portugal* ACM.

Papadias, D., Mamoulis, N., and Delis, V., 1998. Algorithms for Querying Spatial Structure. *In*: *VLDB Conference*, 546–557.

Randell, D.A., Cui, Z., and Cohn, A.G., 1992. A Spatial Logic Based on Regions and Connection. *In*: B. Nebel, C. Rich and W. Swartout, eds. *KR'92. Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference* San Mateo, California: Morgan Kaufmann, 165–176.

Renz, J. and Nebel, B., 1999. On the Complexity of Qualitative Spatial Reasoning: A Maximal Tractable Fragment of the Region Connection Calculus. *Artif. Intell.*, 108 (1-2), 69–123.

Rodríguez, M.A., Bertossi, L.E., and Marileo, M.C., 2013. Consistent query answering under spatial semantic constraints. *Inf. Syst.*, 38 (2), 244–263.

Rodríguez, M.A., *et al.*, 2010. Measuring consistency with respect to topological dependency constraints. *In*: *18th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACM-GIS 2010, San Jose, CA, USA*, 182–191.

Servigne, S., *et al.*, 2000. A Methodology for Spatial Consistency Improvement of Geographic Databases. *GeoInformatica*, 4 (1), 7–34.

Veregin, H., 1999. *In*: *A framework for correcting geographical boundary inconsistency.*, 177–189 John Wiley.

Xie, Z., *et al.*, 2010. A framework for correcting geographical boundary inconsistency. *In*: *Geoinformatics: GIScience in Change, Geoinformatics 2010* IEEE, 1–5.

Zlatanova, S., Rahman, A.A., and Shi, W., 2004. Topological models and frameworks for 3D spatial objects. *Computers & Geosciences*, 30 (4), 419 – 428.

Appendix A. Theoretical framework for the cleaning process

In this appendix, we describe the cleaning process of an inconsistent spatial database as an optimization problem that minimizes the number of changes to restore its consistency. Note that this problem is slightly different from the one in our practical scenario, but its theoretical analysis inspires the practical approach proposed in this paper.

Definition A.1: Let D be a spatial database with integrity constraints Ψ of the form (1) and inconsistency graph $G_{D,\Psi} = (V, E)$. A *cleaning sequence* over G is a set of distinct nodes (i.e. tuples in D) $\mathcal{S}_G$ sorted by their weight such that each node in $\mathcal{S}_G$ is in V and, for each $(t_1, t_2) \in E$, then $t_1 \in \mathcal{S}_G$ or $t_2 \in \mathcal{S}_G$ (or both). Then, a *total cleaning process* on G is defined as the optimization problem of finding the smallest ordered set $\mathcal{S}_G$ (i.e. the smallest number of nodes) such that, by modifying geometries in tuples of the sequence, the consistency of D is restored. □

Intuitively, the *total cleaning process* of a database instance D with inconsistency graph $G_{D,\Psi}$ requires to take nodes in $\mathcal{S}_G$ and modify them such that all conflicts in D are eliminated (i.e. edges in G are eliminated). Note that this process does not say anything about how nodes (geometries) have to be modified. It assumes, however, that all necessary changes on a node are made at once, which is consistent with the orthophoto-guided process assumed in our practical approach. Even more, we will assume, as a first approach, that modifying a geometry does not introduce new conflicts with other geometries and that all conflicts are equally important. An example of such simple strategy is to eliminates one of the tuples that form an edge in the inconsistency graph. However, below we show that even this simple case leads to a hard problem.

Proposition A.2: Let $G = (V, E)$ be an undirected graph representing the inconsistent part of D under the set of integrity constraints Ψ. The decision problem of determining whether or not G has a cleaning sequence of size k is *NP*-complete. □

Proof: (a) *Membership of NP.* A non-deterministic algorithm can guess a set of k nodes in the inconsistency graph and then verify in polynomial time if the modification of the geometries in the tuples makes the database consistent. Note that the cost of modification of a geometry depends on the size of its representation which, in current systems, turns out to be equivalent to the number of points used in the representation. In our problem, let $t_1 = (\bar{u}_1, g_1) \in R$, $t_2 = (\bar{u}_2, g_2) \in P$ be two tuples such that $Confl_{D,R,P,T}(t_1, t_2)$ is true for some relational predicates R and P in D and a topological relation of constraints T in Ψ, we may modify either g_1, g_2 or both.

(b) *Completeness.* This can be proved by reduction of the vertex cover decision problem, which is NP-complete (Cormen *et al.* 2009), to our problem. A vertex cover of an undirected graph $G = (V, E)$ is a subset $V' \in V$ such that if $u, v \in E$, then $u \in V'$ or $v \in V'$ (or both). The decision problem for vertex cover is to determine if a graph has a vertex cover of a given size k.

Given a graph $G = (V, E)$, we need to construct a database instance D with a set of constraints Ψ defined over a database schema such that D has a cleaning sequence of size k if and only if G has a vertex cover of size k. Let us construct a database schema with single relation $R(id, s)$, where id is a thematic attribute and s is a spatial attribute. For each $v \in V$ we insert a tuple (v, g') in R, where g' is initially disjoint but with the same size and shape of all other geometry. For each $e = (u, v) \in E$ we then make the corresponding geometries g_u and g_v in tuples (u, g_u) and (v, g_v) of instance D Touches, and all others remain Disjoint. Then, let Ψ be:

$$\forall u, v, g_1, g_2 (R(u, g_1) \wedge R(v, g_2) \wedge u \neq v \rightarrow \mathsf{Disjoint}(g_1, g_2)).$$

Constructing this database instance can be done in polynomial time with respect to the number of tuples and the size of representation of geometries. Note that imposing the condition that only geometries belonging to tuples that are connected in the graph touch and all others are disjoint defines a set of topological conditions that is realizable (Renz and Nebel 1999) and, therefore, it is possible to have a non-empty database instance that, when checking for consistency, produces the inconsistency graph.

Intuitively each tuple represents a node in G and each edge in the graph is mapped onto an edge in the inconsistency graph. Then, G represents the inconsistent part of D, which can be restored by modifying any of the end nodes of edges in G. By definition of a cleaning sequence, the problem of determining if a subset of vertexes in G is a cleaning sequence of size k is equivalent to the decision problem of the vertex cover of size k. We have proved the proposition. □

The optimization version of this *NP*-complete decision problem is to find the total cleaning sequence, which is in *MAXSNP*-complete. Obviously, the problem becomes harder if we consider the general case in which the modification of one geometry may introduce new conflicts or when conflicts are not equally important.

Appendix B. Implementation issues

As discussed in Section 4, the key factor for the success of a data-cleaning strategy is the quality of the ranking it provides. However, there is a secondary factor represented by the running time of the computations that the strategy involves. Although this time may be negligible when compared with the time consumed by the correction of geometries, an efficient implementation will avoid annoying delays in serving the next geometry to be modified. We describe next an efficient implementation of our dynamic strategy based on two well-known data structures (see for example Mehta and Sahni (2004)): an adjacency list representation of the inconsistency graph and a priority queue.

The priority queue Q must support operations $delete\_max(Q)$ and $decrease\_key(k, id, Q)$. The former returns (and removes from the queue) the element with highest key (weight), whereas the later decreases the key of an inconsistent object id to value (weight) k. Note that id is used just for explanation purposes, the queue does not support efficient access by that field. Thus, we must previously obtain a reference to such element. A max-Heap priority queue implementation supports both these operations in $O(\log n)$ time where n is the size of the queue. In addition, each element in the priority queue must provide direct access to the corresponding node in the inconsistency graph (either a pointer or the object identifier).

On the other hand, the graph representation must support operation $direct\_neighbors(G, x)$, which returns an iterator over the neighbors of node x, which is the identifier of an object. We can assume this operation takes constant time provided direct access to node x. In addition, we assume that each node in the graph stores its weight and a reference to the corresponding element in the priority queue. Similarly, each entry in the adjacency list of a node stores the proportional weight that is due to the conflict of the object with this particular entry and a pointer to the dual relationship (i.e. entry of y in neighboring list of x stores a pointer to entry of x at list of y). We call this pointer the reverse pointer. Finally, both data structures can be efficiently constructed from the dataset and the set of integrity constraints. We do not consider this time as it is an offline process at the beginning of the cleaning process.

We describe now an iteration (lines 7-19 of Algorithm 6) of the strategy in terms of the operations over these data structures. The first step (line 7) performs the operation $delete\_max(Q)$ on the priority queue and obtains a pointer to the node with highest error, u. The third step (we skip the second step as it is manually performed by the user in charge of repair geometries), obtains the $direct\_neighbors(G, u)$ of node u and iterates over them (lines 11-19). For each direct neighbor v, it computes the inconsistency degree of v with the already modified geometry of u (it was modified in step 2). In case there is not conflict any more, v is removed from the neighboring list of u. In other case, we make use of the reverse pointer to update the new weight of u in the neighboring list of v. In both cases, the weight of node v must be updated and the corresponding element in the priority queue deposed via a $decrease\_key()$ operation.

The above explanation is for the case of weights assigned according to inconsistency measures, the cases of weights assigned by area and degree are simplifications of this more general case. For example, when using of areas, the algorithm does nothing when there is still a conflict between u and v (line 18) and the $decrease\_key()$ operation in line 19 is just executed when v does not have more conflicts and, in such case, it always set the new weight to zero (i.e. $decrease\_key(0, v, Q)$). The case of degree is binary, that is, $repinc$ takes value 1 if there is still a conflict between u and v and 0, otherwise.

Overall, an iteration takes $O(n_u \log |V|)$ time, where n_u is the number of neighbors of the node with highest priority at the beginning of the iteration. This complexity is due to the n_u *decrease_key*() operations that may be performed during an iteration.