GeoInformatica manuscript No.
(will be inserted by the editor)

Impact of Data Representation Rules on the Robustness of Topological Relation Evaluation

Alberto Belussi · Sara Migliorini · Mauro Negri · Giuseppe Pelagatti

Received: date / Accepted: date

Abstract A spatial object is characterized not only by its geometric extents, but also by the spatial relations existing with its surrounding objects. An important kind of spatial relations is represented by topological relations. Many models have been defined in literature for formalizing the semantics of topological relations between spatial objects in the Euclidean 2D and 3D space [7, 4, 3]. Nevertheless, when these relations are evaluated in available systems many robustness problems can arise, which are essentially related to the discrete representations adopted by such systems. In a Spatial Data Infrastructure (SDI) the perturbations introduced by the exchange of data between different systems can increase the robustness problems.

This paper deals with a set of rules for the representation of spatial datasets which allow to evaluate topological relations in a robust way using existing systems. These rules are well-known and described in literature and are based on a few basic assumptions on the system behavior which are fulfilled by today's systems. The main contribution of this paper is to determine in detail which rules are sufficient in order to make each topological relation robust; it turns out that the rules depend not only on the topological relation being considered, but also on the geometric types of the involved geometries and on the dimension of the space in which they are embedded, thus giving rise to a very large number of possible combinations. The paper analyses the topological relations and a significant subset of the geometric types defined in the most recent version of the Simple Feature Access (SFA) model published by OGC, considering both a 2D and a 3D space. The extension of the work to the

Alberto Belussi · Sara Migliorni
Department of Computer Science – University of Verona (Italy)
E-mail: alberto.belussi@univr.it

Mauro Negri · Giuseppe Pelagatti
Department of Electronics, Information and Bioengineering – Politecnico of Milan (Italy)

types which have been left out can be done using the same concepts and methodology.

Keywords Robustness · Topological relations · Spatial Data Infrastructure · Discrete representation · Distributed systems

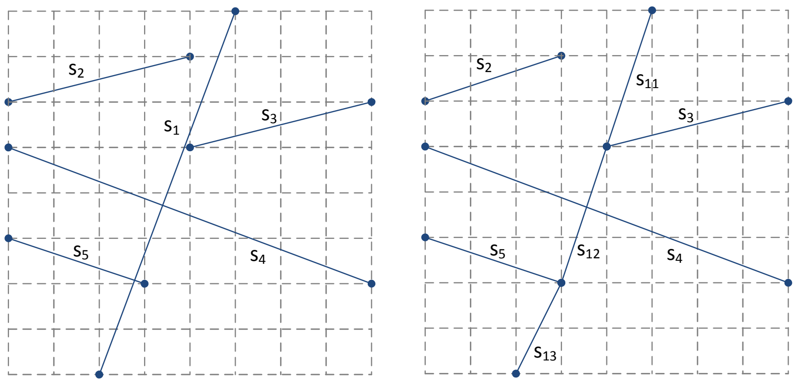
1 Introduction

Topological relations are a fundamental formal tool for describing spatial properties of data in geographical applications: this occurs for example in schema definitions, in order to define spatial integrity constraints, and also in query specification, where topological relations can be used for retrieving information of interest for the user, and in update processes where topological relations are used to specify data quality [20,16].

Although many abstract models have been studied in literature [7,4,3, 19] for defining the semantics of topological relations between geometric objects embedded in a Euclidean space, the problems arising when topological relations are evaluated on data have been much less explored. Topological relations have been defined by using the 9-intersection matrix approach [7] or other axiomatic approaches [19], while for their evaluation specific computational geometry algorithms have been implemented in systems which work on data represented as vectors in a discrete space. The evaluation of topological relations should be *robust*, i.e. two different evaluations performed by the same or by different systems on the same dataset should produce the same result; moreover, in the distributed and heterogeneous context of a Spatial Data Infrastructure (SDI), where datasets are exchanged between systems, also the evaluation of topological relations performed by different systems on data which has been exchanged through the network should remain identical. However, satisfying this requirement can be difficult for several reasons discussed in the next subsection.

1.1 Robustness Problems in the Evaluation of Topological Relations

The first difficulty in obtaining a robust evaluation of topological relations is due to the finite numerical representation of coordinates in the vector representation. 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 theoretically required, is well known [2,11]. For example, consider the 5 segments in Fig. 1.a, where the grid represents the finite 2D space. The size of the grid cells can be very small; for instance, with EPSG:32632 WGS84/UTM coordinates represented by standard FP64 numbers the grid interval is in the worst case 2^{-29} meters. In Fig. 1.a all the segment vertices are on the cell corners, but the distance between a segment and a vertex of another segment can be very small, much smaller than the cell size, as shown by the 3 vertices of segments s_2, s_3 and s_5 with respect to segment s_1. In these cases different systems can produce different evaluations of the topological relations between segments s_2, s_3, s_5 and segment s_1. In order to provide experimental evidence of this problem, we have moved one endpoint of segment s_2 to all positions of a square of 64×64 grid positions and evaluated its topological relation with segment s_1 using two spatial DBMSs: PostGIS [21] and Oracle Spatial [15]. Indeed, the two systems have produced different evaluations in many cases when the endpoint was very near to s_1.



(a) Non-robust situation (b) After application of robustness rules

Fig. 1 Topological relations between segments in a discrete space.

The problems related to the adopted finite number representation are made even worse by the data perturbation occurring during the exchange of data between different systems. Such exchanges can introduce perturbations in geometric representation as a result of the conversions between different formats and precisions. For instance, the GML language [14], 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 most current systems can introduce perturbations. Moreover, in order to reduce the size of datasets, the number of decimal positions in the decimal representation is reduced with respect to the one that would be required in order to keep the original precision. These perturbations can cause a modification of the topological relation between two segments; for example, in Fig. 1.a a small perturbation of the x coordinate of one vertex of s_3 can transform the evaluation of the relation between s_1 and s_3 from disjoint to crosses.

Finally, the robustness problem in the evaluation of topological relations is also related to the dimension of the geometric space (2D or 3D) embedding the objects. For example, in Fig. 1.a segments s_1 and s_4 have a macroscopic intersection in 2D, such that in this space every system will likely evaluate the topological relation as a crosses. Conversely, in a 3D space a small difference in the z-value of the position corresponding to the intersection of their planar projections could cause some systems to evaluate the relation as disjoint while others would still evaluate it as crosses. Therefore, in many cases a distinct analysis of the robustness of topological relations in 2D and 3D is necessary.

1.2 An Approach for Determining the Robustness Rules Required by Topological Relations

In literature several robustness rules have been proposed in order to solve the mentioned robustness problems, and they are to some extent applied by systems. These rules refer to the representation of the data, not to the algorithms adopted for evaluating the relation. 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 [6]) or repeated identically in all objects which share them. This robustness rule can solve many of the mentioned problems, but not all of them. A complementary robustness rule, which has been suggested, for instance in [23], consists in ensuring that a minimum distance is kept between all geometric primitives which are not identical. Fig. 1.b shows a possible application of these rules to the situation of Fig. 1.a: by applying the first rule, segment s_1 has been broken and is now a line constituted by segments s_{11}, s_{12} and s_{13}; by applying the second rule, one vertex of segment s_2 has been moved away from s_1 so that their relationship is interpreted as disjoint.

The primary goal of this paper is to develop an approach for identifying for each topological relation the set of robustness rules which are sufficient to make robust their evaluation; if no other kinds of rules are discovered and applied, these rules are also necessary in order to guarantee the robustness.

As a secondary goal, this paper shows the application of this approach to a wide set of topological relations, trying to cover many of the most relevant situations occurring with data compliant with current standards. However, since the rules required by each topological relation depend also on the type of geometries being considered and on the space (2D or 3D) in which they are embedded, the analysis has been restricted to the following topological relations and data types:

– All topological relations that can be expressed using the well known approach of Egenhofer et al. [7] based on the 9-intersection matrix: the topological relations of the Simple Feature Access (SFA) model published by OGC [13] are a subset of them.
– The geometric types of the most recent version of the SFA, considering both a 2D and a 3D space, but without the collection types.

The approach can be also applied to the collection types in [13], but this would require much space without producing a deeper understanding. Notice that the data types of the most recent SFA version embed the primitives in 3D space but do not include 3D objects (i.e., solids).

The approach presented in this paper is based on the following key points:

1. Definition of a set of vector predicates: a vector predicate is an elementary predicate which can be evaluated in the discrete vector model and is necessary in order to implement some topological relations (Sec. 2).
2. Definition of a set of critical vector predicates: a critical vector predicate is a vector predicate whose evaluation is non robust due to the discussed problems (Sec. 3.1).
3. Determination of which robustness rules are necessary and sufficient for making each critical vector predicate robust, taking into account a reasonable formalization of the systems' behavior in the considered SDI environment. Notice that the given rules are sufficient to guarantee robustness, because their application ensures that topological relations are evaluated in the same way by any implementation. Conversely, the given rules are necessary to guarantee robustness, because if they are not satisfied the evaluation of topological relations becomes not robust, namely the execution of different algorithm implementations may produce different results (Sec. 3.2).
4. Determination of which critical vector predicates are required for the evaluation of each topological relation on some geometrical types (Sec. 4).
5. Derivation of the robustness rules required by a topological relation between two geometries from 3 and 4 (Sec. 4).

Finally, Sec. 5 illustrates the results of some experiments performed in order to confirm the effectiveness of the proposed robustness rules.

1.3 Related Work

Geometric algorithms are typically described assuming an infinite precision that cannot be provided by the adopted computer representations. This assumption raises great difficulties in implementing robust geometric algorithms. A variety of techniques have been proposed in recent years to overcome these issues. For instance, the Exact Geometric Computation model [2] provides a method for making robust the evaluation of geometric algorithms. This can be achieved either by computing every numeric value exactly, or by using some symbolic or implicit numeric representation that allows predicate values to be computed exactly. Exact computation is theoretically possible whenever all the numeric values are algebraic, which is the case for most current problems in computational geometry. This technique has made much progress, so that for certain problems the introduced performance penalty is acceptable. However, when the computation is performed on curved objects or in 3D space the overhead is still large. For this reason, an alternative approach has been proposed which is called Controlled Perturbation (CP) [9] and belongs to the Finite-Precision Approximation Techniques. This method proceeds by perturbating 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. The algorithms of the Snap Rounding family, such as the one in [11] and [10], are examples of this approach. They require the application of rounding algorithms that convert an arbitrary-precision arrangement of segments into a fixed-precision representation. However, even if such algorithms guarantee the robustness of the result, the quality of the geometric approximation in terms of similarity with the original arrangement can be quite low and some topological relations can be modified. Conversely, the aim of this paper is to define rules that can guarantee, when they are satisfied, that a dataset is robust w.r.t. topological relation evaluation. In case of rule violations SR algorithms could be one possible mean for modifying the dataset in order to fulfill the rules.

In the geographical field, topological data models have been defined which use a representation based on topology instead of on coordinates (see for instance [6,22]). 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 original model has been defined for representing objects in a 2D space; however, several extensions to the 3D space have been defined. The Formal Data Structure (FDS) [12] has been the first data structure to consider spatial objects as an integration of geometric and thematic properties. It includes three levels: features related to thematic class, four elementary objects (point, line, surface, and body) and four primitives (node, arc, face, and edge). The model requires that elementary objects shall be disjoint and a 1 to 1 correspondence exists between objects and primitives. In order to overcome some difficulties of FDS in modeling objects with indiscernible boundary, the TEtrahedral Network (TEN) has been proposed in [17]. This model includes 4 primitives: tetrahedron, triangle, arc, and node, where the first one is a real 3D primitive. The Simplified Spatial Model (SSM) [24] has been the first topological structure that focuses on visualization aspects of queries as 3D models. It includes only two primitives: nodes and faces, while an arc can be part of two faces. Finally, the Urban Data Model (UDM) [5] represents the geometry of a body or of a surface using planar convex faces, defined as sets of nodes. In [8] the author defines the concept of geometric realm as a planar graph over a finite resolution grid. Problems of numerical robustness and topological correctness are solved below and within the realm layer so that spatial algebras defined above a realm enjoy very nice algebraic properties. Realms also interact with a database management system to enforce geometric consistency on object creation or update. All these topological representations ensure data quality and relation preservation, but they cannot be applied in a distributed context where data is transferred among different systems. On the contrary, in order to deal with a distributed context where data are exchanged among different systems and evaluated using different algorithm implementations, this paper assumes that geometries are represented with a traditional discrete vector model and defines a set of rules for making robust existing algorithms used to evaluate topological relations.

In [5] the authors face the problem of developing systematic, robust, correct and efficient implementation strategies and optimized evaluation methods for topological predicates between all combinations of the three spatial data types: point, line and polygons. In particular, they recognize four main problems in existing algorithms: even if the plane sweep algorithm is the basis of any topological relation evaluation, (1) each topological predicate usually requires an own, tailored plane sweep algorithm leading to a great number of algorithms; moreover, (2) different outputs can be required on the basis of the considered predicate, and (3) each single algorithm is an ad-hoc implementation for which it is difficult to demonstrate that it covers all cases and guarantees mutual exclusiveness among relations. Finally, (4) the kind of query (verification or determination) usually impacts the evaluation process. For solving these issues a two phases approach is proposed: in a first exploration phase the plane sweep algorithm is used for determining the given configuration between two spatial objects (e.g. their intersections), while in a subsequent evaluation phase the collected information is used to determine the existing relation.

The problem of developing correct and efficient implementation techniques of topological predicates is also treated in [18]. The authors consider all combinations of complex spatial data types including two-dimensional point, line, and region objects. The solution consists of two phases: an exploration phase, which summarizes all intersection and meeting situations in two precisely defined topological feature vectors, and an evaluation phase, which determines the kind of the topological predicate. Besides this general evaluation method, the authors present an optimized method for predicate verification and an optimized method for predicate determination.

The approach adopted in this paper is different from the one in [5,18] because it does not propose different evaluation strategies or algorithms, but it identifies a set of rules for data representation whose compliance guarantees a robust evaluation of topological relations using the existing algorithms. The reason is that in a distributed context it is convenient to guarantee robustness by modifying the geometry representation in a way that any algorithm implementation can produce the same evaluation, rather than rely on a modified implementation that cannot be available everywhere.

2 Discrete Vector Model

This section presents a discrete vector model that contains the data structures and the operations that are usually implemented in current spatial database management systems in order to deal with the evaluation of topological relations. This model is used in this paper as a formal description of an implementation of a part of the Simple Feature Access (SFA) model of OGC [13], which is an abstract specification. The SFA model contains classes describing geometries of the 2D space, but with the possibility to store also the z coordinate usually representing the height above or below sea level (such geometry representation model is often called 2.5D model). Moreover, the type *PolyhedralSurface* is available for representing surfaces in 3D space, as sets of polygon patches with some constraints. The complete type hierarchy is shown in Fig. 2. The main characteristics of these types are supposed to be known by the reader, please refer to [13] for more details.

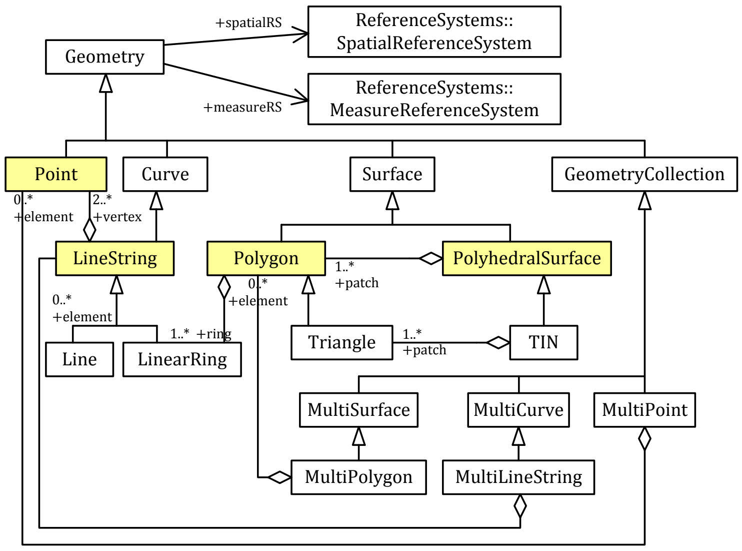


Fig. 2 Geometric type hierarchy of the Simple Feature Access (SFA) model.

As explained in the introduction, we consider only the basic types of the SFA model, since a robustness analysis for the whole set of geometric types cannot be presented due to space constraints. The considered types are: *Point*, *LineString*, *Polygon* and *PolyhedralSurface*, and we focus on the implementation of the tests that are necessary for evaluating the topological relations on geometries of these types. Moreover, we assume that: (i) in *LineString* geometries and *Polygon* rings successive collinear segments are not admitted (they can be merged in one segment); (ii) in *PolyhedralSurface* type adjacent coplanar patches are not admitted (they can be substituted by the patch obtained by merging them) and (iii) patches have no holes, this does not represent a significant limitation, since it is very difficult in a discrete space to represent collinear segments or coplanar patches, and holes in a surface can be generated by a set of patches without holes that together form a shape similar to a ring. Notice that conditions (i) and (ii) does not prevent that two different geometries can have collinear segments or patches, this situation is only avoided inside the same geometry. Finally remember that polyhedral surfaces in the SFA model are simple, i.e. they are surfaces with neither self-intersections nor self-tangency.

2.1 Discrete Vector Model Types

In the considered discrete vector model each geometry is described as a set of vertices embedded in a discrete space. A vertex is represented as a tuple of coordinates, namely by two or three real numbers encoded using a discrete approach, like the floating point model. In the sequel, these numbers are denoted as finite numbers. In the model definition we aim to identify those operations that require a computation on finite numbers, thus having a direct effect on the robustness of topological relation evaluation. The model is composed of some basic vector types that are used to implement the considered SFA types, some basic predicates and operations, and some derived predicates and operations. The following definitions formalize the model.

Definition 1 (Basic vector types)

– A *vertex* v is a tuple of finite numbers representing a 2D or 3D coordinate: $v = (x, y)$ or $v = (x, y, z)$, respectively.
– Let (v_1, v_2) be a pair of vertices, a *segment* is the linear interpolation between them.
– Let $(v_1, \ldots, v_n)$ be a list of vertices, its linear interpolation is a *ring* if and only if $v_1 = v_n$.
– A *patch* is a finite part of a plane whose boundary is defined by a ring. □

Definition 2 (SFA types) Given a geometry g belonging to basic types of the SFA model [13], its discrete representation $DR(g)$ is defined as a follows (v denotes a generic vertex):

– If $g \in Point$ then $DR(g) = v$.
– If $g \in LineString$ then $DR(g) = (v_1, \ldots, v_n)$ with $n > 1$.
The linear interpolation between any two consecutive vertices v_i, v_{i+1} is a segment s_i. Therefore, its discrete representation can be simplified as follows: $DR(g) = (s_1, \ldots, s_m)$ with $m = n - 1$ and $s_i = (v_i, v_{i+1})$.
– If $g \in Polygon$ then $DR(g) = ((v_{1,1}, \ldots, v_{1,n_1}), \ldots, (v_{k,1}, \ldots, v_{k,n_k}))$ with $n_i > 2$ and $k > 0$, where each list of vertices is a ring: the first one represents the outer boundary, while the other ones represent the inner boundaries (i.e. possible holes). Notice that since a *Polygon* can be defined only in a 2D space, all boundary rings are coplanar.
Using the ring definition, the discrete representation of $g \in Polygon$ can be simplified as follows: $DR(g) = (r_1, \ldots, r_k)$ with $k > 0$, where each $r_i = (v_{i,1}, \ldots, v_{i,n_i})$ is a ring .
– If $g \in PolyhedralSurface$ then $DR(g) = ((v_{1,1}, \ldots, v_{1,n_1}), \ldots, (v_{k,1}, \ldots, v_{k,n_k}))$ with $n_i > 2$ and $k > 0$, where each list of vertices is a ring representing a polygon without holes.
Using the patch definition, the discrete representation of a *PolyhedralSurface* can be simplified as $DR(g) = (p_1, \ldots, p_k)$ with $k > 0$ and where each p_i is a planar patch defined by the ring $(v_{i,1}, \ldots, v_{i,n_i})$. □

Def. 2 considers only the basic types of the SFA model, in accordance with what previously stated. For a formal definition of all SFA type properties and of the other SFA types deriving as specialization of the *GeometryCollection* class, please refer to [13].

In Def. 3 a set of basic operations and predicates is defined for analyzing the implementation of the topological relation evaluation in a discrete vector space. Indeed, each of them identifies a type of processing on finite numbers that is required in many cases.

Definition 3 (Basic vector predicates and operations) The signature of each operation has the syntax: $\langle \mathit{ret\ type}\rangle\langle \mathit{geom}\rangle.\langle \mathit{op\ name}\rangle(\langle \mathit{par\ type}\rangle\langle \mathit{par\ name}\rangle)$.

<u>*Operations*</u>

- $\mathit{dist} : \mathit{vertex} \times \mathit{vertex} \rightarrow \mathit{real}$, $v_1.\mathit{dist}(v_2)$ returns the Euclidean distance between two vertices v_1 and v_2.
- $\mathit{start} : \mathit{segment} \rightarrow \mathit{vertex}$, $s.\mathit{start}()$ returns the start point of the segment s.
- $\mathit{end} : \mathit{segment} \rightarrow \mathit{vertex}$, $s.\mathit{end}()$ returns the end point of the segment s.
- $\mathit{ray} : \mathit{segment} \times \mathcal{P}(\mathit{segment}) \rightarrow \mathit{integer}$, $s.\mathit{ray}(S)$ returns the number of segments of S that the semi-straight line starting from $s.\mathit{start}()$ and passing through $s.\mathit{end}()$ intersects (excluding the possible intersection at $s.\mathit{start}()$).
- $\cup : \mathit{segment} \times \cdots \times \mathit{segment} \rightarrow \mathit{segment}$, $s_1 \cup \cdots \cup s_n$ joins n overlapping (or touching) segments that lie on the same straight line, if the segments are disjoint or do not lie on the same straight line, it returns the empty geometry.[1] Given two set of segments $S_i = \{s_{i1}, \ldots, s_{in}\}$ and $S_j = \{s_{j1}, \ldots, s_{jm}\}$, the compact notation $S_i \cup S_j$ can be used to denote the join of its contained segments: $S_i \cup S_j = s_{i1} \cup \cdots \cup s_{in} \cup s_{j1} \cup s_{jm}$.
- $\mathit{bnd} : \mathit{patch} \rightarrow \mathcal{P}(\mathit{segment})$, $p.\mathit{bnd}()$ returns the set of segments defining the boundary of the patch p.

<u>*Predicates*</u>

- $\mathit{eq} : \mathit{vertex} \times \mathit{vertex} \rightarrow \mathit{boolean}$, $v.\mathit{eq}(v_0)$ tests the equality between two vertexes; two vertices are equal only if they are bitwise identical.
- $\mathit{cnt} : \mathit{segment} \times \mathit{vertex} \rightarrow \mathit{boolean}$, $s.\mathit{cnt}(v)$ tests the containment between a vertex v and the interior of a segment s: $v \subset I(s)$.
- $\mathit{int} : \mathit{segment} \times \mathit{segment} \rightarrow \mathit{boolean}$, $s_1.\mathit{int}(s_2)$ tests the intersection between the interiors of two segments: $\mathit{dim}(I(s_1) \cap I(s_2)) = 0$. If the intersection has dimension 1, it returns false.
- $\mathit{cnt} : \mathit{patch} \times \mathit{vertex} \rightarrow \mathit{boolean}$, $p.\mathit{cnt}(v)$ tests the inclusion between a vertex v and the interior of a patch p: $v \in I(p)$.
- $\mathit{int} : \mathit{patch} \times \mathit{segment} \rightarrow \mathit{boolean}$, $p.\mathit{int}(s)$ tests the intersection between the interior of a patch p and the interior of a segment s: $\mathit{dim}(I(s) \cap I(p)) = 0$. If the intersection has dimension 1, it returns false.
- $\mathit{int} : \mathit{patch} \times \mathit{patch} \rightarrow \mathit{boolean}$, $p_1.\mathit{int}(p_2)$ tests the intersection between the interior of two patches: $\mathit{dim}(I(p_1) \cap I(p_2)) = 1$. If the intersection has dimension 2, it returns false. □

[1] This operation is applied to segments produced as intermediate result of other operations and cannot be applied to segments of a *LineString* that are not collinear by definition.

Notice that, the set of basic predicates is very small and does not contain all the elementary predicates that one could expect. For example, there is no test of boundary intersection between two segments. Indeed, the boundary intersection between two segments can be obtained by comparing for equality the vertices of their boundaries using other three basic operations/predicates: $s.start()$, $s.end()$ and $v.eq(v_0)$. In a similar way, many other expected operations can be derived from these basic ones.

Since in many cases during topological relation evaluation the same expressions have to be reused, the most common repeated expressions are introduced below as derived operations and predicates. Some expressions do not identify an exact algorithm for their evaluation, but they identify the need for some basic operations and predicates, thus requiring a specific type of processing. For example, the test of segment overlapping $s.ov(s_0)$ requires to test the containment of a vertex in a segment and can avoid the test of segment intersection.

Definition 4 (Derived vector predicates and operations)

Operations and predicates on vertices
Semantics of operations and predicates is shown in Tab. 11 of App. A. In the sequel the list of operations and predicates is presented (v, v_i, $v_{i,j}$ represent a vertex). Notice that, for evaluating topological relations only the test of equality between two vertices will be required (see Sec. 2.2), since also the *belong-to* relation, the intersection computation, and the test of not empty intersection between two sets of vertices can be derived from it.

- $set : vertex \times \mathcal{P}\,(vertex) \rightarrow boolean$, $v.bel(V)$ tests if a vertex v belongs to a set of vertices $V = \{v_1, \ldots, v_n\}$.
- $\cap : \mathcal{P}\,(vertex) \times \mathcal{P}\,(vertex) \rightarrow \mathcal{P}\,(vertex)$, $\{v_{1,1}, \ldots, v_{1,n}\} \cap \{v_{2,1}, \ldots, v_{2,m}\}$ returns the common vertices between $\{v_{1,1}, \ldots, v_{1,n}\}$ and $\{v_{2,1}, \ldots, v_{2,m}\}$.

Operations and predicates on segments
Semantics of operations and predicates is shown in Tab. 12 of App. A. In the sequel the list of operations and predicates is presented (s, s_i are segments). Notice that, for evaluating topological relations only the test of equality between two vertices and the containment of a vertex in a segment will be required (see Sec. 2.2), since also the equality relation and the topological relations: *in*, *overlaps* and *disjoint* between two segments can be derived from them.

- $bnd : segment \rightarrow \mathcal{P}\,(vertex)$, $s.bnd()$ returns the boundary of a segment s.
- $eq : segment \times segment \rightarrow boolean$, $s_1.eq(s_2)$ tests the equality between two segments s_1 and s_2.
- $bel : segment \times \mathcal{P}\,(segment) \rightarrow boolean$, $s.bel(S)$ tests if the segment s belongs to the set of segments $S = \{s_1, \ldots, s_n\}$.
- $in : segment \times segment \rightarrow boolean$, $s_1.in(s_2)$ tests the inclusion between two segments: $s_1 \subset s_2$.
- $ov : segment \times segment \rightarrow boolean$, $s_1.ov(s_2)$ tests the intersection between two segments that requires they share a portion of line, but excludes inclusion or equality (i.e. $dim(I(s_1) \cap I(s_2) = 1)$.

- *dj* : *segment* $\times$ *segment* $\rightarrow$ *boolean*, $s_1.dj(s_2)$ tests of interior disjointness between two segments $(I(s_1) \cap I(s_2) = \emptyset)$.
- *diff* : *segment* $\times \mathcal{P}$ (*segment*) $\rightarrow \mathcal{P}$ (*segment*), $s.diff(S)$ computes the difference between a segment s and a set of segments S that overlap s.

Operations and predicates on patches

Semantics of operations and predicates is shown in Tab. 13 of App. A. Notice that, for evaluating topological relations the necessary tests on patches are: containment and overlapping of a segment in a patch, and relations *overlaps*, *disjoint*, *in* and *equals* between patches. Moreover, for specifying the containment of a segment in a patch in particular cases, an additional operation is introduced, called $p.bnd_{ov}(s)$, that returns the set of boundary segments of a patch p that overlap or are contained in a segment s. Finally, we remark that for evaluating the overlaps relation between a patch and a segment the predicate $p.cnt(s)$ is not necessary. In the sequel p, p_i represent a patch:

- *ver* : *patch* $\rightarrow \mathcal{P}$ (*vertex*), $p.ver()$ returns the patch vertices.
- cnt_{int} : *patch* $\times$ *segment* $\rightarrow$ *boolean*, $p.cnt_{int}(s)$ tests the inclusion between a segment s and a patch interior $(I(s) \subset I(p))$.
- *cnt* : *patch* $\times$ *segment* $\rightarrow$ *boolean*, $p.cnt(s)$ tests the inclusion between a segment s and a patch p $(s \subset p)$.
- *ov* : *patch* $\times$ *segment* $\rightarrow$ *boolean*, $p.ov(s)$ tests the overlapping between a segment interior and a patch interior $(dim(I(s) \cap I(p)) = 1 \wedge \neg(I(s) \subset I(p))$.
- *dj* : *patch* $\times$ *segment* $\rightarrow$ *boolean*, $p.dj(s)$ is a test of disjointness between a segment interior and a patch interior $(I(s) \cap I(p) = \emptyset)$.
- *eq* : *patch* $\times$ *patch* $\rightarrow$ *boolean*, $p_1.eq(p_2)$ tests the equality between two patches p_1 and p_2.
- int_2 : *patch* $\times$ *patch* $\rightarrow$ *boolean*, $p_1.int_2(p_2)$ is a test of interior intersection of dimension 2 between two patches interior $(dim(I(p_1) \cap I(p_2)) = 2)$.
- *in* : *patch* $\times$ *patch* $\rightarrow$ *boolean*, $p_1.in(p_2)$ tests the inclusion between two patches $(p_1 \subset p_2)$.
- *dj* : *patch* $\times$ *patch* $\rightarrow$ *boolean*, $p_1.dj(p_2)$ is a test of interior disjointness between two patches $(I(p1) \cap I(p2) = \emptyset)$.
- *ov* : *patch* $\times$ *patch* $\rightarrow$ *boolean*, $p_1.ov(p_2)$ is a test of interior overlapping between two patches $(dim(I(p_1) \cap I(p_2)) = 2 \ \wedge \ \neg(I(p_1) \subset I(p_2)) \ \wedge \ \neg(I(p_2) \subset I(p_1)))$. □

In order to simplify as much as possible the specification of the evaluation tests for topological relations, which are illustrated in Sec. 2.2, some additional derived operations and predicates are presented in Tab. 14 of App. A, that apply to geometries of types: *LineString*, *Polygon* and *PolyhedralSurface*.

2.2 Testing Topological Relations in the Discrete Vector Model

This paper considers all topological relations that can be expressed using the well known approach of Egenhofer et al. [7] based the 9-intersection matrix. The topological relations of the SFA model are a subset of them.

The 9-intersection matrix is defined by using the concepts of interior (internal part), boundary and exterior (external part) of a geometric object. Given a geometric object a of the abstract type *Geometry* and the operations: $a.PS()$ returning the point set represented by a, $a.boundary()$, returning the geometric object representing the boundary of a (or the *emptyGeometry* if a has an empty boundary), the following point sets are defined:

1. *interior* of a, denoted as $I(a)$: it is the point set $a.PS() \setminus a.boundary().PS()$. Namely, it is the set of object points that do not belong to its boundary.
2. *boundary* of a, denoted as $B(a)$: it is the point set $a.boundary().PS()$.
3. *exterior* of a, denoted as $E(a)$: it is the point set $a.space() \setminus a.PS()$. Namely, it is the set of points from the space embedding a that do not belong to the object itself.

Definition 5 (Topological relation) Given two geometric objects a and b of any geometric type, the definition of a topological relation is given, using the combinatorial topology approach and the Dimensionally Extended 9-Intersection Model (DE-9IM) [4], by assigning the following matrix:

$$R(a,b) = \begin{bmatrix} dim(I(a) \cap I(b)) & dim(I(a) \cap B(b)) & dim(I(a) \cap E(b)) \\ dim(B(a) \cap I(b)) & dim(B(a) \cap B(b)) & dim(B(a) \cap E(b)) \\ dim(E(a) \cap I(b)) & dim(E(a) \cap B(b)) & dim(E(a) \cap E(b)) \end{bmatrix} \quad (1)$$

where the possible values are $\{F, 0, 1, 2, T, *\}$ with the meaning F: empty set, 0: point, 1: curve, 2: surface, *: any, T: 0,1,2. □

In the sequel, we show how each cell of the 9-intersection matrix can be evaluated in the discrete vector model presented above. Given the discrete representation $DR(g_1)$, $DR(g_2)$ of two geometries g_1, g_2, the content of each cell can be computed by evaluating a given logical expression which contains some of the vector operations and predicates previously presented and has $DR(g_1)$, $DR(g_2)$ as parameters. The logical expressions are obtained by considering for each cell all the possible combinations of geometry types that are admissible. Since this matrix can be used for the definition of any topological relation, the obtained set of expressions is sufficient for evaluating any topological relation defined by means of a 9-intersection matrix. This approach is similar to the one applied in [18], but is extended to the 3D space types of the SFA specification.

Proposition 1 (Evaluation of matrix cells using vector predicates) *Given two geometries a and b, the following tables show for each combination of types the cells of the matrix that have to be evaluated: one table regards the 2D space, the other one the 3D space. In the table the following symbols are used: T (F) indicates that the cell for the considered combination of types ($type(a)/type(b)$) is always true (false), while $Tab(\boldsymbol{x})$ means that Tab. $\boldsymbol{x}$ of App. B shows the logical expression that implements in the discrete model the test required for evaluating the cell. Finally, if the cell can be evaluated considering the test specified for another cell $c_{i,j}$ of the matrix M_{t_1,t_2}, this cell is* *specified by the symbol* $c_{i,j}^{t_1,t_2}$.

2D	*Point (**PT**)*	*LineString (**LN**)*	*Polygon (**PL**)*
PT	M_{PP} $\begin{bmatrix} Tab(15) & F & \neg c_{1,1} \\ F & F & F \\ \neg c_{1,1} & F & T \end{bmatrix}$	M_{PL} $\begin{bmatrix} Tab(15) & Tab(19) & Tab(23) \\ F & F & F \\ T & e_1(*) & T \end{bmatrix}$	M_{PG} $\begin{bmatrix} Tab(15) & c_{1,1}^{PL} & Tab(23) \\ F & F & F \\ T & T & T \end{bmatrix}$
LN	$M_{LP} = M_{PL}^T$	M_{LL} $\begin{bmatrix} Tab(18) & Tab(19) & Tab(24) \\ Tab(19) & c_{1,1}^{PP} & e_2(*) \\ Tab(24) & e_2(*) & T \end{bmatrix}$	M_{LG} $\begin{bmatrix} Tab(16) & c_{1,1}^{LL} & Tab(24) \\ c_{1,1}^{PG} & c_{1,1}^{PL} & e_3(*) \\ T & c_{1,3}^{LL} & T \end{bmatrix}$
PL	$M_{GP} = M_{PG}^T$	$M_{GL} = M_{LG}^T$	M_{GG} $\begin{bmatrix} Tab(21) & c_{1,1}^{GL} & Tab(25) \\ c_{1,1}^{LG} & c_{1,1}^{LL} & c_{1,3}^{LG} \\ Tab(25) & c_{1,3}^{GL} & T \end{bmatrix}$

(*) e_1*: if ln.bnd()* $\neq \emptyset$ *then T else F;* e_2*: if ln.bnd()* $= \emptyset$ *then F else* $c_{1,3}^{PL}$*;*
e_3*: if ln.bnd()* $= \emptyset$ *then F else* $c_{1,3}^{PG}$.

3D	*Point (**PT**)*	*LineString (**LN**)*	*PolyhedralSurface (PS)*
PS	M_{SP} $\begin{bmatrix} Tab(15) & F & T \\ c_{1,1}^{LP} & F & T \\ Tab(23) & F & T \end{bmatrix}$	M_{SL} $\begin{bmatrix} Tab(20) & c_{1,1}^{SP} & T \\ c_{1,1}^{LL} & c_{1,1}^{LP} & c_{1,3}^{LL} \\ Tab(24) & e_1(*) & T \end{bmatrix}$	M_{SL} $\begin{bmatrix} Tab(22) & c_{1,1}^{SL} & Tab(25) \\ c_{1,1}^{SL} & c_{1,1}^{LL} & c_{3,1}^{SL} \\ Tab(25) & c_{3,1}^{SL} & T \end{bmatrix}$

(*) e_1*: if ln.bnd()* $= \emptyset$ *then F else* $c_{3,1}^{SP}$.

Proof The completeness proof of the above tables and of the cases illustrated in App. B is presented in [1]. □

3 Robustness of Vector Predicates

The discrete representation of geometries may produce many unexpected problems during query evaluation and quality tests, due to the two main problems highlighted in the introduction: numerical weakness of algorithms implementation and data perturbation induced by transfer. Sec. 3.1 formalizes such problems and defines a set of assumptions about the considered environment, then Sec. 3.2 analyses the robustness of the vector predicates used in the evaluation of topological relations.

3.1 Assumptions on Systems and Implementations

Let us consider two systems, denoted here as S (source) and D (destination), which exchange spatial data and evaluate topological relations on such exchanged data. During the transfer a transformation may occur on each vertex; let v_S be a vertex in S and v_D the same vertex in D, the transformation from v_S to v_D is captured by a mapping $f()$, such that $v_D = f(v_S)$.

The behavior of current systems displays a set of problems with respect to robustness. In the sequel these problems are listed and for each one some minimal assumptions on the system behavior are stated. They are considered as necessary preconditions in order to build robustness rules.

Problem 1 (Numerical weakness in algorithms implementation) An aspect that introduces ambiguity in the evaluation of topological relations is the necessity of implementing a system of linear equations in order to evaluate some *elementary geometric tests*. These tests are used in all implementations of the basic vector predicates. They are ambiguous due to the finite precision of the discrete geometric representation and of the algorithms implementation in current GIS systems. □

In particular, the following tests might lead to ambiguous results, when performed on different systems.

Problem 1.1 (Vertex-Vertex equality/disjunction) If the two vertices are close to each other, different systems may return different results. □

Problem 1.2 (Vertex-Segment relative position) If the distance between a vertex v and a segment s is very small, then different systems may return different results The relative position between an (oriented) segment and a close vertex in a 2D/3D space can be: (i) the segment contains the vertex; (ii) the segment is on the left of the vertex; (iii) the segment is on the right of the vertex. Left and right refers to the unique plane that contains both s and v. For example, s can contain v and $f(s)$ might be on the left of $f(v)$ or s can be on the right of v and $f(s)$ on the left of $f(v)$. □

Problem 1.3 (Segment-Segment intersection/disjunction in 3D) In some cases different systems return different results if the two segments are close to each other. □

Segment-segment intersection in 2D is not an independent elementary problem, since it can be reduced to Prob 1.2, as it will be shown in Sec. 3.2.

Problem 1.4 (Vertex-Patch relative position) If the distance between a vertex v and a patch p is very small, then different systems can return different results. The relative position between a vertex and an (oriented) patch in the 3D space can be: (i) the patch contains the vertex; (ii) the patch is above the vertex; (iii) the patch is below the vertex. □

Problem 1.5 (Vertex-Vertex distance computation) Computing the distance between two vertices in different systems can return different results. □

In order to achieve the robustness in these elementary tests, the following assumptions are introduced. In particular, Ass. 1 is important to preserve vertex identity, and Ass. 3 is important to preserve the disjunction of geometries.

Assumption 1 (Equality preservation in data exchange) The mapping $f()$ representing the exchange of geometries between a source system S and a destination system D is a function, namely: $\forall v_1, v_2 \in \mathcal{V}(v_1.eq(v_2) \implies f(v_1).eq(f(v_2)))$ where $\mathcal{V}$ is the set of vertices in the source system S. □

Assumption 2 (Bounded error in distance computation) Given two vertices transferred from system S to system D, the evaluation of their relative distance in the two systems can be not equal, but the difference is less then a threshold T: $\forall v_1, v_2 \in \mathcal{V}(\mid v_1 \mathit{dist}(v_2) - f(v_1).\mathit{dist}(f(v_2)) \mid < T)$. □

Assumption 3 (Minimum distance for preserving relative positions) In the transfer of geometries from system S to system D, it is possible to determine a threshold distance TD ($TD >> T$) such that the following conditions hold, where $\mathcal{V}$ is the set of vertices, $\mathcal{S}$ is the set of segments and $\mathcal{P}$ is the set of patches in the source system S.

Assumption 3.1 (Vertex-Vertex minimum distance) $\forall v \in \mathcal{V}$ $(\forall v_1 \in \mathcal{V}$ $(v.\mathit{dist}(v_1) > TD + T \Longrightarrow (\neg v.\mathit{eq}(v_1) \Longrightarrow \neg f(v).\mathit{eq}(f(v_1)))$. □

Assumption 3.2 (Vertex-Segment minimum distance) $\forall v \in \mathcal{V}$ $(\forall s \in \mathcal{S}$ $(v.\mathit{dist}(s) > TD + T \Longrightarrow$ ((s is on the left of v) $\wedge$ ($f(s)$ is on the left of $f(v)$) $\vee$ (s is on the right of v) $\wedge$ ($f(v)$ is on the right of $f(v)$))))) □

Assumption 3.3 (Segment-Segment 3D minimum distance) $\forall s \in \mathcal{S}_{3D}$ $(\forall s_1 \in \mathcal{S}_{3D}(s.\mathit{dist}(s_1) > TD + T \Longrightarrow (\neg s.\mathit{int}(s_1) \Longrightarrow \neg f(s).\mathit{int}(f(s_1)))$. □

Assumption 3.4 (Vertex-Patch minimum distance) $\forall v \in \mathcal{V}$ $(\forall p \in \mathcal{P}$ $(v.\mathit{dist}(p) > TD + T \Longrightarrow$ ((p is above v) $\wedge$ ($f(p)$ is above $f(v)$) $\vee$ (p is below v) $\wedge$ ($f(p)$ is below $f(v)$))))). □

The requirement $TD >> T$ is necessary because the uncertainty in the distance computation T would make Ass. 3 useless. Notice that with the precision levels provided by current systems, TD is significantly smaller than the average error in the coordinate representation with respect to real positions of points (spatial accuracy): $TD <<$ *Absolute-Positional-Error*. Moreover, the value for TD can be chosen by applying the following considerations: (i) TD only depends on the precision in coordinate representation; (ii) this precision in the floating point representation depends on the magnitude of the numbers to be represented: the higher is the magnitude, the lower is the precision, the higher will be TD. For example, considering the worse case in EPSG:32632 WGS84/UTM coordinates which implies a precision around 10^{-9} meters, the value of TD should be significantly greater than 10^{-9}, so that also the numerical weakness of distance evaluation can be adsorbed. Finally, notice that as shown in Sec. 1.1 in real datasets it is not correct to assume that TD is the minimum distance among the represented vertices, segments and patches, if this property has not been explicitly implemented.

The second problem that can impact the robustness of elementary tests is due to the perturbation that a dataset can suffer in a transfer process from one system to another one.

Problem 2 (Perturbation in data exchange) The data transfer between two systems S and D may cause a perturbation that slightly modifies the discrete representation of geometries. □

In order to deal with this problem, the following minimal assumption is stated.

Assumption 4 (Boundedness of perturbation in data exchange) The perturbation introduced in a vertex representation by the mapping $f()$ has an upper bound, called PB (Perturbation upper bound). In other words, given a vertex v, the distance between v and $f(v)$ is always less than PB: $\forall v \in \mathcal{V}\ (v.dist(f(v)) < PB)$. □

If a sequence of n data transfers is considered instead of a single one, then it is also assumed that the combined effect of all perturbations cannot diverge: $\forall v \in \mathcal{V}_S(v.dist(f_n(\ldots(f_1(v)))) < PB)$. Otherwise, after each perturbation the robustness rules that will be presented in next section will guarantee the preservation of the elementary tests, but the perturbed dataset will possibly not satisfy the robustness rules any more, thus requiring a procedure for restoring robustness before another perturbation is applied.

3.2 Robustness Evaluation of Vector Predicates

The implementation of topological relations on the discrete vector model makes use of a set of predicates, which include in their definition the elementary operations described in Sec. 2. For the reasons highlighted above, some of these predicates can return different results in different systems, and are called here critical vector predicates. This section introduces the set of critical vector predicates that are used in the evaluation of topological relations and defines a set of rules for making them robust.

Given the problems on systems and implementations described in Sec. 3.1, the vector predicates, introduced by Def. 3 of Sec. 2, can be proved to be critical with respect to robustness, since their evaluation on the same data can change when data are transferred from one system to another one.

Proposition 2 (Lack of robustness in basic predicates and operations) *Given the systems and implementation problems described in Sec. 3.1, the predicates of Def. 3 are all not robust, and hence are called* ***critical vector predicates****. Among the operations presented in Def. 3, only the following two operations are not robust: $v.dist(v_0)$ and $s.ray(S)$.*

Proof The lack of robustness of $v.eq(v_0)$, $s.cnt(v)$ and $p.cnt(v)$ is a direct consequence of Prob. 1.1, 1.2 and 1.4, respectively. Relatively to $s.int(s_0)$, in 3D it is a direct consequence of Prob. 1.3; while in 2D, it is not robust since it can be reduced to a combination of elementary operations which are not robust. In particular, if two segments s_1 and s_2 intersect each other in 2D then the relative positions of the end points of s_1 w.r.t. s_2 are opposite (i.e., one is on the left of s_2, the other one is on the right). These relative positions are subject to Prob. 1.2 and thus $s.int(s_0)$ is not robust also in 2D. Similarly, $p.int(s)$ is not robust since it can be reduced to a combination of elementary operations that are not robust. In particular, since $p.int(s)$ requires that the

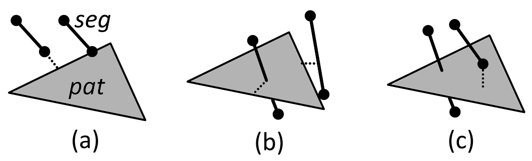


Fig. 3 Critical situations for vector predicates.

intersection dimension is zero, then p and s have to be not coplanar and the possible relative positions of s w.r.t. p are those ones shown in Fig. 3.a, Fig. 3.b and Fig. 3.c, which are subject to Prob. 1.2, 1.3 and 1.4, respectively. A similar reasoning is true also for $p.int(p_0)$ by considering the boundary segments of one patch w.r.t. the other one. The lack of robustness of the operation $v.dist(v_0)$ is a direct consequence of Prob. 1.5. Finally the operation $s.ray(S)$ is not robust due to the Prob. 1.3, because the variation in segment intersection detection produces a different intersection count and thus a different result of the $s.ray(S)$ operation. □

3.2.1 Robustness Rules for Basic Critical Predicates

This section introduces a set of rules that, applied on the datasets in the source system S, make non ambiguous the evaluation of the critical vector predicates identified in Def. 3. These rules are classified into two categories: Identity Rules (IR) and minimum Distance Rules (DR). The following IR are used to ensure the robustness of topological relation evaluation and are based on Ass. 1.

Rule 1 (IR1: Vertex-Segment) For each vertex v that has to lie in a segment $s = (v_1, v_2)$, a new vertex v_h bitwise identical to v has to be introduced in the s representation splitting it into two new segments $s_1 = (v_1, v_h)$ and $s_2 = (v_h, v_2)$ [8]. Therefore, after the rule IR1 has been applied, the following condition is always true: $\forall v \in S(\forall s \in S(\neg s.cnt(v)))$. □

Notice that, in a discrete vector model after splitting a segment s the resulting segments s_1 and s_2 are usually not collinear due to the involved approximations, except for some statistically rare cases. Since the presence of collinear segments complicates some predicate definitions and in general requires to transform all tests between two segments into tests between a segment and a set of segments without really affecting the contribution, in this paper the vector model does not consider the presence of collinear segments in the representation of a geometry.

Rule 2 (IR2: Segment-Segment) All intersections between two segments s_1 and s_2 have to be represented by splitting them through the insertion of a new common vertex, which has to be represented only once or by means of several identical instances, namely instances that have bitwise identical coordinates. Therefore, after the rule IR2 has been applied, the following condition is true: $\forall s_1 \in S(\forall s_2 \in S(\neg s_1.eq(s_2) \implies \neg s_1.int(s_2)))$. □

Rule 3 (IR3: Vertex-Patch) All vertex-patch intersections must be represented by a vertex v contained in the patch definition. Therefore, the patch representation has to be split in two and the vertex v has to be inserted as a new start/end point of a new segment composing one of the patch boundaries. Therefore, after the rule IR3 has been applied, the following condition is true: $\forall v \in S(\forall p \in S(\neg p.cnt(v)))$. □

These rules are not sufficient alone to guarantee the robustness for all critical predicates. In particular, in order to solve the situation highlighted in Prob. 1.1-1.4 and given Ass. 4 and 3, the following other rules are introduced.

Rule 4 (DR0: Vertex-Vertex) For all pairs of vertices v_1 and v_2 of S, the distance between them is either zero or is greater than a Minimum Granted Distance (MGD): $\forall v_1 \in S(\forall v_2 \in S(v_1.dist(v_2) = 0 \vee v_1.dist(v_2) > MGD))$. □

Rule 5 (DR1: Vertex-Segment) For all vertices v and for all segments s of S, the distance between them is either zero or is greater than MGD: $\forall v \in S(\forall s \in S(s.dist(v) = 0 \ \vee \ s.dist(v) > MGD))$. □

Rule 6 (DR2: Segment-Segment) For all pairs of distinct segments s_1 and s_2 of S, the distance between them is either zero or is greater than MGD: $\forall s_1 \in S(\forall s_2 \in S(s_1.dist(s_2) = 0 \ \vee \ s_2.dist(s_2) > MGD))$. □

Rule 7 (DR3: Vertex-Patch) For all vertices v (representing isolated points or segment end points) and for all patches p of S, the distance between them is either zero or is greater than MGD: $\forall v \in S(\forall p \in S(p.dist(v) = 0 \ \vee \ p.dist(v) > MGD))$. □

Considering Ass. 3, 2 and 4, MGD should be always be always greater than $2 \times PB + TD + T$, since we need to guarantee a distance $TD + T$ between two geometries even if each of them moves towards the other by PB.

Notice that in 2D space DR2 is implied by DR1 and that the minimum granted distance between patches and segments is implied by DR2 and DR3.

Theorem 1 (Robustness of critical vector predicates) *Given Ass. 1, 3 and 4 the necessary and sufficient rules to be applied on dastasets in the source system (S) in order to guarantee the robustness evaluation of the critical vector predicates in the destination system (D) are those shown in Tab. 1. Notice that for the mentioned assumptions, when IR1, IR2 and IR3 are applied on S, predicates $s.cnt(v)$, $s_1.int(s_2)$ and $p.cnt(v)$ are always false.*

Proof $v_1.eq(v_2)$ Considering the first row, DR0 is a sufficient condition for the robustness of $v_1.eq(v_2)$. If $v_1.eq(v_2)$ is true in S, then Ass. 1 preserves the truth of the predicate; while, if $v_1.eq(v_2)$ is false, then given Ass. 1 and Ass. 3.1, DR0 guarantees that two distinct vertices remain distinct. Regarding the necessity of DR0, observe that without it a false $v_1.eq(v_2)$ can become true after a perturbation due to data transfer.

$s.cnt(v)$ Considering the second row, IR1+DR1 are sufficient conditions for the robustness of $s.cnt(v)$. If $s.cnt(v)$ is true, IR1 has not been applied

Table 1 Robustness conditions for critical vector predicates

Predicate	**in 2D space**	**in 3D space**
$v_1.eq(v_2)$	DR0	DR0
$s.cnt(v)$	IR1 + DR1	IR1 + DR1
$s_1.int(s_2)$	IR1 + DR1	IR1 + IR2 + DR2
$p.cnt(v)$	IR1 + DR1	IR1 + IR3 + DR3
$p.int(s)$	NA	IR1 + IR2 + IR3 + DR2 + DR3
$p_1.int(p_2)$	NA	IR1 + IR2 + IR3 + DR2 + DR3

correctly, hence after its application this case cannot occur. Conversely, if $s.cnt(v)$ is false, then either v is an end point of s or v is disjoint from s. In the first case, given Ass. 1, IR1 is sufficient for preserving in data transfer the identity between a vertex and a segment end point, which preserves the falsity of $f(s).cnt(f(v))$ avoiding the vertex to move into the segment interior. In the second case, given Ass. 4 and Ass. 3.2, DR1 is sufficient for preserving the spatial disjunction among a vertex and a segment interior, which preserves the falsity of $f(s).cnt(f(v))$. Regarding the necessity of IR1 (DR1), observe that without IR1 (DR1) the first (second) case described above cannot be robust.

$s_1.int(s_2)$ Considering the third row, in 2D DR1 is sufficient to preserve $s_1.int(s_2)$, if two segments intersect, then the relative position of one segment end point with respect to the other one is opposite (i.e., one end point is on the left, and the other one on the right of the other segment); hence, given Ass. 4 and Ass. 3.2, after a data transfer DR1 preserves the relative position of vertices with respect to segments and $s_1.int(s_2)$ is preserved. The same reasoning applies for two disjoint but very close segments. In addition, IR1 is applied so that, given Ass. 1, it avoids that a $s_1.cnt(s_2.start())$ (or any of the other possible containment between s_1 interior and s_2 end points) becomes a $f(s_1).int(f(s_2))$. In 3D IR2 is also introduced in order to avoid that two crossing segments become disjoint, and DR2 is introduced in place of DR1 since, given Ass. 3.3, it preserves the spatial disjunction of segments, avoiding that two disjoint segments become interior crossing segments. Regarding the necessity of DR1 in 2D, observe that without it, due to Prob. 1.2, the relative positions of one segment end points with respect to the other segment can change converting two crossing segments in two disjoint segments or vice versa; finally, without IR1 a $s_1.cnt(s_2.start())$ (or other similar situations) can be transformed after data transfer into a $f(s_1).int(f(s_2))$.

$p.cnt(v)$ Considering the fourth row, in 2D if $p.cnt(v)$ is true then the relative position of v with respect to a segment of p boundary has to be preserved; DR1 is sufficient to preserve this condition, since, given Ass. 4 and Ass. 3.2, after a data transfer DR1 preserves the relative position of vertices with respect to segments. In addition, IR1 is applied so that, given Ass. 1, it avoids that, given a segment s of p boundary such that $v.eq(s.start())$ (or $v.eq(s.end())$) becomes a $f(p).cnt(f(v))$. Finally, if $p.dj(v)$, then given Ass. 4 and Ass. 3.2, DR1 is sufficient for preserving the relative position between a vertex and the segment of p boundary. Regarding the necessity of DR1 in 2D we can observe that without DR1, due to Prob. 1.2, the relative positions of a vertex with respect to a segment of a patch boundary can change moving the vertex inside or outside the patch; finally, without IR1 a $v.eq(s.start())$ (or $v.eq(s.end()))$ can be transformed after data transfer into a $f(p).cnt(f(v))$.

In 3D, IR1+IR3+DR3 are sufficient conditions for the robustness of $p.cnt(v)$. In particular, if $p.cnt(v)$ is true, then IR3 has not been applied correctly, hence after its application this case cannot occur. Conversely, if $p.cnt(v)$ is false then either v is an end point of some segments of p boundary (indeed, IR1 excludes that v lies in the interior of some boundary segments) or v is disjoint from p. In the first case, given Ass. 1, IR3 is sufficient for preserving in data transfer the identity between a vertex and a segment end point, which preserves the falsity of $f(p).cnt(f(v))$ avoiding the vertex to move into the patch interior. In the second case, given Ass. 4 and Ass. 3.3, DR3 is sufficient for preserving the spatial disjunction among a vertex and a patch interior, which preserves the falsity of $f(p).cnt(f(v))$. In addition, IR1 is introduced so that, it avoids a vertex v to be contained in one segment of the patch boundary; moreover, when v is equal to an internal vertex of the patch, given Ass. 1, it avoids that $f(p).cnt(f(v))$ becomes true. Regarding the necessity of IR3, observe that without it the first case described above cannot be robust. The necessity of DR3 is justified by the second case; finally IR1 is necessary in order to avoid that, due to Prob. 1.2, an internal vertex of the patch moves inside it.

$p.int(s)$ Considering the fifth row, in 3D IR1+IR2+IR3+DR2+DR3 are sufficient conditions for the robustness of $p.int(s)$. If $p.int(s)$ is true, then the relative position of the end points of s with respect to the patch p is opposite (i.e., one end point is above, and the other one below the patch); hence, given Ass. 4 and Ass. 3.4, after a data exchange DR3 preserves the relative position of vertices with respect to patches and $p.int(s)$ is thus preserved. Conversely, if $p.int(s)$ is false, then the following cases are possible: (i) s is disjoint from p, or (ii) s interacts with segments of p boundary (s_p). In the first case, given Ass. 3.3 and Ass. 3.4, IR3+DR3+DR2 preserve disjointness, avoiding that an end point v of s, such that $p.cnt(v)$, crosses the p border (IR3); or an end point of s changes its relative position w.r.t. p (DR3), or s changes its relative position w.r.t. a segment of p boundary (DR2). In the second case, if $s.int(s_p)$ is true then, given Prob. 1.3, a perturbation might transform it into a $f(p).int(f(s))$; IR2 avoids the possibility of having this case. Finally, IR1 avoids the possibility of having that $s_p.cnt(s.start())$ (or $s_p.cnt(s.end()))$ is true, thus the last possible case is that $s_p.start().equal(s.start())$ (or any other combination of start/end points) and this is preserved by IR1 and Ass. 1. Regarding the necessity of the conditions, observe that without each of the listed rules, one of the above described cases cannot be robust.

$p_1.int(p_2)$ The proof for the sixth row is very similar to the previous one, since the preservation of patch intersections requires the preservation of intersection between patch boundary segments and patch interiors. The only case that has to be considered in addition is the situation in which two patches are intersecting without requiring the intersection of any patch boundary segments with other patch interior (see Fig. 4), in this case the preservation of segments intersection is sufficient, and hence IR2 is required. □

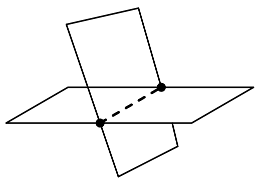


Fig. 4 Example of intersecting patches without any interior and boundary intersections.

Before stating robustness rules for critical vector operations, let us remark that in this context the required robustness for the operation $v.dist(v_0)$ is not that the distance between two vertices is kept unmodified, but it is sufficient that: given two vertices v_1 and v_2 such that $v_1.dist(v_2) > 0$, then $f(v_1).dist(f(v_2)) > 0$.

Theorem 2 (Robustness of critical vector operations) *Given Ass. 1-4, the necessary and sufficient rules to be applied on datasets in the source system S in order to guarantee the robustness of the critical vector operations evaluation on target system D are the following ones: (i) for $v.dist(v_0)$ DR0 is required (ii) for $s.ray(S_0)$ DR1 is required.*

Proof $v.dist(v_0)$ If DR0 is satisfied, then, given Ass. 3, a distance $v_1.dist(v_2) > 0$ cannot become equal to zero in D. Vice-versa, if the distance $v_1.dist(v_2) > 0$ becomes zero in D then DR0 is violated, thus DR0 is also a necessary condition.

$s.ray(S)$ Consider that this predicate is used to verify if a point ($s.start()$) is inside or outside a ring (S_0), thus the alteration of the number of intersections between the ray and the segments of S_0 far from $s.start()$ can only occur when two segments of S_0 produce a shape similar to a cusp so that in S system the cusp intersects the ray and in D system it does not (or vice versa). Hence, the count is decreased (or increased) of two, which does not change the inside/outside evaluation. Therefore, the only problem regards what happens very near to $s.start()$, however, if DR1 guarantees the robustness of the relative position of a vertex and a segment, then DR1 is sufficient to ensure the robustness of $s.ray(S_0)$. The need comes directly from the fact that the only alteration of the $s.ray(S_0)$ result can occur when $s.start()$ crosses one segment of S_0, which violates DR1. □

3.2.2 Robustness Rules for Derived Critical Predicates

A derived critical predicate is a predicate which can be expressed in terms of other predicates, at least one of which is critical. A derived critical predicate becomes robust if and only if its constituent predicates become robust. Among all possible derived predicates, here are considered the ones that are useful for the implementation of the topological relation evaluations, which have been presented in Def. 4. In this section the conditions for guaranteeing their robust evaluation are shown.

Table 2 Robustness conditions for derived critical vector predicates. Terms DR1^0 (DR2^0) means that DR0 is implied by DR1 (DR2), and DR2^1 that DR1 is implied by DR2.

Derived Predicate	**in 2D space**	**in 3D space**	**Basic Predicates**
$v.bel(V)$, $V_1 \cap V_2$ $V_1 \cap V_2 \neq \emptyset, s.eq(s_0)$, $s.bel(S), ln_1.eq(ln_2)$	DR0	DR0	$v.eq(v_0)$
$pg_1.eq(pg_2)$	DR0	–	$v.eq(v_0)$
$p.eq(p_0), ps_1.eq(ps_2)$	–	DR0	$v.eq(v_0)$
$s.ov(s_0)$	IR1 + DR1	IR1 + DR1	$s.cnt(v)$
$s.in(s_0)$	IR1 + DR1^0	IR1 + DR1^0	$v.eq(v_0), s.cnt(v)$
$s.dj(s_0)$	IR1 + DR1^0	IR1 + IR2 + DR2^{0,1}	$v.eq(v_0), s.cnt(v)$, $s.int(s_0)$
$p.ov(s)$	IR1 + DR1^0	IR1 + IR2 + IR3 + DR2^{0,1} + DR3	$v.eq(v_0), s.cnt(v)$, $s.int(s_0), p.cnt(v)$,
$p.cnt(s), p.ov(p_0)$, $p.in(p_0), p.int_2(p_0)$	IR1 + DR1^0	IR1 + IR2 + IR3 + DR2^{0,1} + DR3	$v.eq(v_0), s.cnt(v)$, $s.int(s_0), s.ray(S)$, $p.cnt(v)$
$p.dj(s)$	IR1 + DR1^0	IR1 + IR2 + IR3 + DR2^{0,1} + DR3	$v.eq(v_0), s.cnt(v)$, $s.int(s_0), s.ray(S)$, $p.cnt(v), p.int(s)$
$p_1.dj(p_2)$	IR1 + DR1^0	IR1 + IR2 + IR3 + DR2^{0,1} + DR3	$v.eq(v_0), s.cnt(v)$, $s.int(s_0), s.ray(S)$ $p.cnt(v), p.int(p_0)$

Lemma 1 (Robustness of derived critical vector predicates) *Under the hypothesis of Thm. 1 and Thm. 2, the derived vector predicates of Def. 4 become robust with the application of the rules shown in Tab. 2 (second and third column for 2D and 3D space, respectively).*

Proof The derived vector predicates listed in the first column are defined in terms of basic predicates listed in the fourth column (see Def. 3); hence according to Thm. 1 and Thm. 2, the sufficient and necessary rules for guaranteeing their robustness derive from the union of the rules that guarantee the robustness of the basic predicates that define them. □

4 Robustness of Topological Relations Evaluation

The previous section discussed the robustness of vector predicates and operations. The obtained results are applied in this section for deriving the robustness of topological relations. Remember that, if no rules are introduced on discrete geometries representation, vector predicates are all not robust.

The results are presented by showing, for each possible pair of geometric types, a *robustness matrix* which specifies in each cell the required rules for guaranteeing a robust evaluation. In particular, the necessary robustness rules are identified by means of a set of predefined robustness levels, that are defined below. Basically, greater level number means an increasing number of robustness rules to be applied.

Definition 6 (Robustness levels) Considering the results shown in Thm. 1 and Lemma 1, the following robustness levels are introduced *level R*: no additional rules are necessary to guarantee robustness; *level 0*: DR0 is required; *level 1*: IR1 and DR1 are required; *level 2*: IR1, IR2 and DR2 are required; *level 3*: IR1, IR3 and DR3 are required; *level 4*: IR1, IR2, IR3, DR2 and DR3 are required. □

The following theorems present the *robustness matrix* for each combination of geometric types in 2D and 3D space, respectively. Notice that different cells and different combinations of types require different robustness rules. Moreover, the embedding space has also an impact on this analysis. In the tables each matrix cell contains a 4-tuple $(L_0 L_1 L_2 L_T)$ representing the robustness level that is required to implement the test $dim(f(A) \cap f(B)) = x$, where $x \in \{0, 1, 2, T\}$. In the 4-tuples the symbol F in position i is used to indicate that the intersection producing the dimension i is always false.

Theorem 3 (2D Robustness matrices) *Tab. 3 shows the robustness matrices for all the possible combinations of geometric types in 2D space.*

Proof The presented robustness matrices have been obtained by considering the expressions presented in Prop. 1 for intersection evaluation in each matrix cell and the results of Thm. 1, Thm. 2 and Lemma 1. In particular, for each combination of types a matrix has been computed, and for each matrix cell the corresponding vector expression has been analyzed and the critical basic or derived vector predicates contained in it have been identified, then according to Thm. 1, Thm. 2 and Lemma 1 the maximum level of robustness required by the identified critical predicates has been computed. □

Theorem 4 (3D Robustness Matrices) *Tab. 4 shows the robustness matrices for all the possible combinations of geometric types in 3D space.*

Proof See proof of Thm. 3. □

As an example of the application of these results to some topological relations of SFA model we consider here the disjoint and touch relations, which can be defined using the 9-IM matrix as shown in the following definition.

Table 3 Robustness matrices in 2D space.

2D	*Point* (**PT**)	*LineString* (**LN**)	*Polygon* (**PL**)
PT	(pt, pt) $\begin{bmatrix} (0FF0) & F & (0FF0) \\ F & F & F \\ (0FF0) & F & T \end{bmatrix}$	(pt, ln) $\begin{bmatrix} (1FF1) & (0FF0) & (1FF1) \\ F & F & F \\ T & (RFFR) & T \end{bmatrix}$	(pt, pg) $\begin{bmatrix} (1FF1) & (1FF1) & (1FF1) \\ F & F & F \\ T & T & T \end{bmatrix}$
LN	$(ln, pt) = (pt, ln)^T$ $\begin{bmatrix} (1FF1) & F & T \\ (0FF0) & F & (RFFR) \\ (1FF1) & F & T \end{bmatrix}$	(ln, ln) $\begin{bmatrix} (11F1) & (1FF1) & (F1F1) \\ (1FF1) & (0FF0) & (1FF1) \\ (F1F1) & (1FF1) & T \end{bmatrix}$	(ln, pg) $\begin{bmatrix} (F1F1) & (11F1) & (F1F1) \\ (1FF1) & (1FF1) & (F1F1) \\ T & (RFFR) & T \end{bmatrix}$
PL	$(pg, pt) = (pt, pg)^T$ $\begin{bmatrix} (1FF1) & F & T \\ (1FF1) & F & T \\ (1FF1) & F & T \end{bmatrix}$	$(pg, ln) = (ln, pg)^T$ $\begin{bmatrix} (F1F1) & (1FF1) & T \\ (11F1) & (1FF1) & (F1F1) \\ (F1F1) & (F1F1) & T \end{bmatrix}$	(pg, pg) $\begin{bmatrix} (FF11) & (F1F1) & (FF11) \\ (F1F1) & (11F1) & (F1F1) \\ (F1F1) & (F1F1) & T \end{bmatrix}$

Table 4 Robusness matrices in 3D space.

3D	*Point* (**PT**)	*LineString* (**LN**)	*PolyhedralSurface* (**PS**)
PT	(pt, pt) $\begin{bmatrix} (0FF0) & F & (0FF0) \\ F & F & F \\ (0FF0) & F & T \end{bmatrix}$	(pt, ln) $\begin{bmatrix} (1FF1) & (0FF0) & (1FF1) \\ F & F & F \\ T & (RFFR) & T \end{bmatrix}$	(pt, ps) $\begin{bmatrix} (3FF3) & (1FF1) & (3FF3) \\ F & F & F \\ T & (RFFR) & T \end{bmatrix}$
LN	$(ln, pt) = (pt, ln)^T$ $\begin{bmatrix} (1FF1) & F & T \\ (0FF0) & F & (RFFR) \\ (1FF1) & F & T \end{bmatrix}$	(ln, ln) $\begin{bmatrix} (21F2) & (1FF1) & (F2F2) \\ (1FF1) & (0FF0) & (1FF1) \\ (F2F2) & (1FF1) & T \end{bmatrix}$	(ln, ps) $\begin{bmatrix} (44F4) & (21F2) & (F4F4) \\ (3FF3) & (1FF1) & (3FF3) \\ T & (F2F2) & T \end{bmatrix}$
PS	$(ps, pt) = (pt, ps)^T$ $\begin{bmatrix} (3FF3) & F & T \\ (1FF1) & F & (RFFR) \\ (3FF3) & F & T \end{bmatrix}$	$(ps, ln) = (ln, ps)^T$ $\begin{bmatrix} (44F4) & (3FF3) & T \\ (21F1) & (1FF1) & (F2F2) \\ (F4F4) & (3FF3) & T \end{bmatrix}$	(ps, ps) $\begin{bmatrix} (4444) & (44F4) & (FF44) \\ (44F4) & (21F2) & (F4F4) \\ (FF44) & (F4F4) & T \end{bmatrix}$

Definition 7 (Disjoint and touches relations of SFA) The formal definition of each relation is presented below together with the specification of the corresponding set of 9-intersection matrices. Notice that the set of matrices can change with respect to the considered geometric types. In particular, the following notation is used *pt* denotes a point, *c* denotes a curve, *s* denotes a surface, while *a*, *b* are generic geometries.

- **DJ**: $a.disjoint(b) \equiv_{def} a.PS() \cap b.PS() = \emptyset$
  $R_{dj}(pt, pt) = [FFT\ FFF\ TFT]$
  $R_{dj}(pt, c/s) = [FFT\ FFF\ T*T]$
  $R_{dj}(c/s, pt) = R_{dj}(pt, c/s)^T$
  $R_{dj}(c/s, c/s) = [FFT\ FF*\ T*T]$
- **TC**: $a.touch(b) \equiv_{def} (I(a) \cap I(b) = \emptyset) \wedge (a.PS() \cap b.PS() \neq \emptyset)$
  $R_{tc}(pt, c/s) = [FTF\ FFF\ TTT]$
  $R_{tc}(c/s, pt) = R_{tc}(pt, c/s)^T$
  $R_{tc}(c, s) = [F**\ *T*\ T*T] \cup [FT*\ ***\ T*T] \cup [F**\ T**\ T*T]$[2]
  $R_{tc}(s, c) = R_{tc}(c, s)^T$

[2] only in 3D spaces

$$R_{tc}(c,c) = [F*T\ *T*\ T*T] \cup [F*T\ T**\ T*T] \cup [FTT\ ***\ T*T]$$
$$R_{tc}(s,s) = [F*T\ *T*\ T*T] \cup [F*T\ T**\ T*T]^1 \cup [FTT\ ***\ T*T]^2 \quad \square$$

Tab. 5 and Tab. 6 show the minimum and maximum level of robustness for all the intersection tests required by *disjoint* in 2D and 3D, respectively. Notice that, as expected, the required robustness level changes according to the considered combination of geometric types, e.g. $a.DJ(b)$ has robustness level $(0,0)$ for the combination (pt, ln) in 2D, while has $(1,1)$ for the combination (ln, pg) in the same space; and also according to the embedding space, e.g. $ln.DJ(ln)$ has robustness level $(0,1)$ in 2D, while has $(1,2)$ in 3D. Each table

Table 5 $a.DJ(b)$ in 2D. The notation $f(c_{x,y})$ means that the cell is a derived cell and its test can be avoided since it is implied by the test of $c_{x,y}$.

2D	*Point* (**PT**)	*LineString* (**LN**)	*Polygon* (**PL**)
PT	$R_{dj}(pt,pt)=$ $[\underline{F}\mathbf{F}T\ \mathbf{FFT}\ T\mathbf{FT}]$ $\begin{bmatrix} 0 & F & f(c_{1,1}) \\ F & F & F \\ f(c_{1,1}) & F & T \end{bmatrix}$ Robust. level = [0,0]	$R_{dj}(pt,ln)=$ $[\underline{FF}T\ \mathbf{FFF}\ \mathbf{T*T}]$ $\begin{bmatrix} 1 & 0 & f(c_{1,1},c_{1,2}) \\ F & F & F \\ T & * & T \end{bmatrix}$ Robust. level = [0,1]	$R_{dj}(pt,pg)=$ $[\underline{FF}T\ \mathbf{FFF}\ \mathbf{TTT}]$ $\begin{bmatrix} 1 & 1 & f(c_{1,1},c_{1,2}) \\ F & F & F \\ T & T & T \end{bmatrix}$ Robust. level = [1,1]
LN	$R_{dj}(ln,pt) = R_{dj}(pt,ln)^T$ Robust. level = [0,1]	$R_{dj}(ln,ln) =$ $[\underline{FF}T\ \underline{FF}*\ T*\mathbf{T}]$ $\begin{bmatrix} 1 & 1 & f(c_{1,1}) \\ 1 & 0 & * \\ f(c_{1,1}) & * & T \end{bmatrix}$ Robust. level = [0,1]	$R_{dj}(ln,pg) =$ $[\underline{FF}T\ \underline{FF}*\ \mathbf{T}T\mathbf{T}]$ $\begin{bmatrix} 1 & 1 & f(c_{1,1},c_{1,2}) \\ 1 & 1 & * \\ T & f(c_{1,2}) & T \end{bmatrix}$ Robust. level = [1,1]
PL	$R_{dj}(pg,pt) = R_{dj}(pt,pg)^T$ Robust. level = [1,1]	$R_{dj}(pg,ln) = R_{dj}(ln,pg)^T$ Robust. level = [1,1]	$R_{dj}(pg,pg) =$ $[\underline{F}FT\ F\underline{F}T\ TT\mathbf{T}]$ $\begin{bmatrix} 1 & f(c_{1,1}) & f(c_{1,1}) \\ f(c_{1,1}) & 1 & f(c_{1,1}) \\ f(c_{1,1}) & f(c_{1,1}) & T \end{bmatrix}$ Robust. level = [1,1]

cell contains for a pair of geometric types t_1, t_2:

- The 9-intersection matrix of the relation using the following notation: (i) the matrix cells that must be considered for testing the relation (*necessary cells*) are underlined; (ii) the cells in bold are not to be considered, since their value is fixed for the considered geometric types; (iii) finally, the cells in italics are functions of the necessary ones (*derived cells*).
- The 9-intersection matrix of the relation showing in each necessary cell its robustness level, and for each derived cell the cells from which they depend.
- The robustness level of the relation is described by specifying the minimum and maximum level of robustness of the tests regarding necessary cells.

As a further example, App. C considers the robustness behavior the relation *Touches*. In particular, Tab. 26 and Tab. 27 show the minimum and maximum level of robustness required for all intersection tests.

Table 6 $a.DJ(b)$ in 3D. The notation $f(c_{x,y})$ means that the cell is a derived cell and its test can be avoided since it is implied by the test of $c_{x,y}$.

3D	*Point* (**PT**)	*LineString* (**LN**)	*PolyhedralSurface* (**PS**)
PT	Same as 2D	Same as 2D	$R_{dj}(pt, ps)=$ $[\underline{FF}T\ \mathbf{FFF}\ \mathbf{T}{*}\mathbf{T}]$ $\begin{bmatrix} 3 & 1 & f(c_{1,1}, c_{1,2}) \\ F & F & F \\ T & * & T \end{bmatrix}$
	Robust. level = [0,0]	Robust. level = [0,1]	Robust. level = [1,3]
LN	Same as 2D	$R_{dj}(ln, ln) =$ $[\underline{FF}T\ \underline{FF}{*}\ T{*}\mathbf{T}]$ $\begin{bmatrix} 2 & 1 & f(c_{1,1}) \\ 1 & 0 & * \\ f(c_{1,1}) & * & T \end{bmatrix}$	$R_{dj}(ln, ps) =$ $[\underline{FF}T\ \underline{FF}{*}\ \mathbf{T}T\mathbf{T}]$ $\begin{bmatrix} 4 & 2 & f(c_{1,1}) \\ 3 & 1 & * \\ T & * & T \end{bmatrix}$
	Robust. level = [0,1]	Robust. level = [1,2]	Robust. level = [1,4]
PS	$R_{dj}(ps, pt) = R_{dj}(pt, ps)^T$	$R_{dj}(ps, ln) = R_{dj}(ln, ps)^T$	$R_{dj}(ps, ps) =$ $[\underline{FF}T\ \underline{FF}{*}\ T{*}\mathbf{T}]$ $\begin{bmatrix} 4 & 4 & f(c_{1,1}) \\ 4 & 2 & * \\ f(c_{1,1}) & * & T \end{bmatrix}$
	Robust. level = [1,3]	Robust. level = [1,4]	Robust. level = [2,4]

In the following section topological relations *in* (IN), *contains* (CT), *equals* (EQ), *overlap* (OV), and *cross* (CR) are used in addition to the DJ and TC, in order to discuss the presented experiments. The assumed semantics for these relations is the one presented in [13].

5 Experimental Results

This section describes the results of some experiments performed with the aim to confirm the effectiveness of the robustness rules defined in Sec. 3.2. In particular, we want to show that different rules are required according to: (i) the considered topological relation, (ii) the types of the geometries and (iii) the space in which geometries are embedded. Each experiment evaluates the topological relations existing between the geometries of two datasets having EPSG:32632 WGS84/UTM Zone 32N coordinates; the considered cases are: (i) *Points* and *LineStrings* in 2D; (ii) *Polygons* and *Polygons* in 2D; (iii) *LineStrings* and *LineStrings* in 3D; (iv) *Points* and *PolyhedralSurfaces* in 3D.

In all experiments, first the topological relations among each pair of datasets have been computed without any manipulation. Then, the coordinates of the geometries have been rounded using a grid of different granularities ranging from 10^{-6} to 10^{-1}, in order to simulate a data perturbation, and the relations have been evaluated again. Only the results regarding *DJ* and *TC* are reported since only these two relations have been analyzed in the previous section.

The experiment results are reported in tables which show the number of relation changes and the number of violations of the robustness rules which are required according to the theorems of the previous sections. Remember that the DRx violations are always computed on the original geometries with an increasing value of MGD. In all tables the number of rules violations are always greater than the number of relation changes; this is in accordance with the fact that a relation change always requires a rule violation and not vice versa, since a lack in robustness does not necessarily imply a relation change. When available, we have used in the experiments real datasets produced by different providers, since the production processes might have had an impact on dataset robustness.

Points and LineStrings in 2D

The following four datasets containing *Points* and *LineStrings* in 2D space, produced by different subjects, have been considered:

- Road Elements (RE_{north}) and Road Junctions (RJ_{north}) of a town of Northen Italy: containing 3851 and 2856 geometries, respectively.
- Road Elements (RE_{cent}) and Road Junctions (RJ_{cent}) of a village in Central Italy: containing 7458 and 2401 geometries, respectively.

The computed relations are: DJ, TC and IN. Remember that the DJ relation on types (*pt*,*ln*) in 2D space has a *robustness level 1* (see Tab. 5), which requires that IR1 and DR1 rules are satisfied by robust datasets; the TC relation in the same case has a *robustness level 0* (see Tab. 26 in App. C), thus only DR0 has to be satisfied for obtaining robustness.

The results of the experiments are shown in Tab. 7, where the number of changes for TC and DJ relations after data perturbations of different sizes are reported (we consider both the gained and the lost TC/DJ relations). The results regard both the *north* and *cent* datasets and also the violations of the DR0 and DR1 rules in both datasets are shown in the last columns. There are no violations of IR1 in both datasets.

Table 7 Number of relation changes between pairs of geometries computed by the query $RE_x \bowtie_Q RJ_x$ (where $Q = RE.MBR \cap RJ.MBR \neq \emptyset$). The cardinality of the result is 8651 (6596) pairs for x=north (x=cent). First column contains the simulated perturbation PB, in the next columns the number of changes w.r.t. no rounded datasets regarding relation TC and relation DJ are reported and the last two columns report violations of DR0 and DR1 considering $MGD = 2 \times PB$. For DR0 in brackets the number of violations between a junction and a point of a road element boundary are reported.

	# TC changes		# DJ changes				# violations of rules			
	$IN \rightarrow TC$		$DJ \rightarrow TC$		$DJ \rightarrow IN$		DR0 (DR0 boundary)		DR1	
PB	north	cent	north	cent	north	cent	north	cent	north	cent
10^{-6}	0	0	16	3	0	0	42(42)	3(3)	54	3
10^{-5}	0	0	48	3	1	0	55(55)	3(3)	112	3
10^{-4}	0	0	76	3	0	0	330(330)	5(5)	693	5
10^{-3}	0	0	2255	5	2	0	2297(2291)	5(5)	2339	6
10^{-2}	0	1	2089	8	2	0	2301(2293)	14(14)	2344	16
10^{-1}	0	2	2282	15	4	0	2346(2312)	37(35)	2382	40

The results of this experiment are consistent with the assumptions and the robustness rules presented in the previous section, since:

- no TC relations are lost after perturbations: this is consistent with Ass. 1 that preserving coordinate equality preserves existing TC relations between *Points* and *LineStrings*;
- some new TC relations have been produced in this experiment for perturbations of size 10^{-2} and 10^{-1} and some DR0 violations are detected: this is consistent with Thm. 3 and Def. 7, since DR0 is required by the robustness rule of TC to avoid that a different relation becomes a TC relation;
- no IR1 violations are observed in both datasets, no new DJ relations are generated by perturbations, while many of them are lost due to DR0 and DR1 violations: this is consistent with Thm. 3 and Def. 7.

Further observations on this experiment are: (i) more relation changes are observed on the *north* than on the *cent* datasets, and accordingly the number of violations of DR0 and DR1 is higher for *north* than *cent* datasets. (ii) Since this experiment regards points and vertices of *LineStrings*, each violation of DR0 always produces at least one violation of DR1. Indeed, the number of observed DR1 violations is higher than the number of DR0 violations.

Table 8 Number of relation changes between pairs of geometries computed by the query $RA_x \bowtie_Q VA_x$ (where $Q = RA.MBR \cap VA.MBR \neq \emptyset$). The cardinality of the result is 20667 (5885) pairs for x=north (x=cent). Columns are organized as in Tab. 7. The last two columns report violations of DR1 considering $MGD = 2 \times PB$.

	# TC changes						# DJ changes				# violations	
	$TC \to OV$		$OV \to TC$ $CT \to TC$		$DJ \to TC$		$DJ \to OV$		$OV \to DJ$		of $DR1$	
PB	north	cent	north	cent	north	cent	north	cent	north	cent	north	cent
10^{-6}	139	0	4361	0	2346	0	81	0	32	0	358200	0
10^{-5}	111	0	5484	0	2639	0	32	0	4	0	359066	0
10^{-4}	114	0	5552	0	2663	0	22	0	5	0	359844	8
10^{-3}	119	0	5555	0	2658	0	25	0	2	0	363126	16
10^{-2}	101	0	5529	1	2634	0	32	0	6	0	366124	2340
10^{-1}	99	1	5545	2	2632	0	29	0	1	0	391114	12518

Polygons in 2D

The following four datasets containing polygons have been considered:

- Road Areas (RA_{north}) and Vehicular Areas (VA_{north}) of a town of Northen Italy: containing 2955 and 5860 geometries respectively.
- Road Areas (RA_{cent}) and Vehicular Areas (VA_{cent}) of a village in Central Italy: containing 436 and 1769 geometries, respectively.

The same process described before has been performed also on these datasets. The computed relations are: DJ, TC, OV, CT and IN. Remember that we focus on the DJ and TC relations, which on types (*pg*, *pg*) in 2D space have both a *robustness level 1* (see Tab. 5 and Tab. 26 in App. C), which requires that IR1 and DR1 have to be satisfied by robust datasets. The results of this experiment are reported in Tab. 8, where columns are organized as described in the previous experiment, but in this case only the violations of DR1 are reported. Moreover, there are 68 violations of IR1 in the *north* datasets and no violation in the *cent* datasets, which are not shown in the table since identity rules do not depend on the considered MGD.

The results are consistent with the system assumptions and robustness rules presented in the previous sections, in particular:

- The changes of TC into OV relations in the *north* datasets indicate a lack of robustness that has been detected by the violations of IR1 and DR1 rules, that were required by TC relation according to Thm. 3 and Def. 7.
- The great number of changes of DJ into TC or OV relations in the *north* datasets again indicates a lack of robustness that is certified by the high number of DR1 violations, indeed DR1 is required for DJ robustness according to Thm. 3 and Def. 7.
- The significant number of DR1 violations in *cent* datasets for a perturbation of 10^{-1} does not produce changes in DJ relations, which means that although *cent* datasets are not robust no change in TC or DJ relations occurs (in the experiments we registered some changes of OV relations).
- The only case of TC change in *cent* dataset for a perturbation of 10^{-1} is due to a violation of DR1, that after the perturbation has not modified the touching segments but has produced a cross of a polygon boundary by another vertex, thus transforming the TC in an OV relation.
- As in the previous experiment, the DJ relation changes more frequently than the TC relation and changes are more frequent in the *north* than in the *cent* datasets. Accordingly to this behavior, the number of DR1 violations is higher in *north* than *cent* datasets.

Table 9 Number of relation changes between pairs of geometries computed by the query $RE_x \bowtie_Q RE_x$ (where $Q = RE.MBR \cap RE.MBR \neq \emptyset$). The cardinality of the result is 9612 (12632) pairs for $x = north$ ($x = cent$). Columns are organized as in Tab. 7. The last two columns report violations of DR2 considering $MGD = 2 \times PB$.

	# TC changes				# DJ changes				# violations of	
	$TC \to DJ$		$TC \to CR$		$DJ \to TC$		$DJ \to CR$		DR2	
PB	north	cent	north	cent	north	cent	north	cent	north	cent
10^{-6}	0	0	0	0	0	0	0	0	0	4
10^{-5}	0	0	0	0	0	0	0	0	0	6
10^{-4}	0	0	0	0	0	4	0	0	0	26
10^{-3}	0	0	0	0	0	10	0	0	376	28
10^{-2}	0	0	0	0	442	13	0	0	1322	38
10^{-1}	0	0	0	0	725	13	0	0	2202	74

LineStrings in 3D

In order to test the rules also in the 3D space, we have considered the datasets of road elements, which contain 3D *LineStrings*, and we have computed the relations among them in 3D. The computed relations are: DJ, TC and CR. The DJ and TC relations on types (*ln*, *ln*) in 3D space have both a *robustness level 2* (see Tab. 6 and Tab. 27 in App. C), thus IR1, IR2 and DR2 have to be satisfied for obtaining robustness.

Results are reported in Tab. 9, where columns are organized as described in the previous experiment, but in this case only the violations of DR2 are reported. There are no violations of IR1 and IR2 in both datasets.

The results are consistent with the system assumptions and robustness rules presented in the previous sections, in particular:

– In 3D space both DJ and TC relations change only with perturbations greater than 10^{-4} for *cent* and 10^{-3} for *north* datasets. As highlighted in previous experiments, also in this case we observe a greater robustness of existing TC relations w.r.t. existing DJ relation in both datasets.
– Also the number of DR2 violations is consistent, according to Thm. 4 and Def. 7, with the observed relation changes and shows that *north* and *cent* datasets are robust for DJ and TC relations in case of perturbations of 10^{-4} and 10^{-3}, respectively.

Table 10 Number of relation changes between pairs of geometries computed by the query $Points_x \bowtie_Q PSurface_x$ (where $Q = Point.geometry \cap PSurface.MBR \neq \emptyset$) and x represents the dataset cardinality ($x \in \{5000, 10000\}$). The cardinality of the result is 6826741 (27497550) pairs for $x = 5000$ ($x = 10000$). Columns are organized as in Tab. 7. The last two columns report violations of DR3 (DR1) considering $MGD = 2 \times PB$.

	# TC changes				# DJ changes						# violations of DR3 (DR1)	
	TC → *DJ*		*TC* → *IN* *IN* → *TC*		*DJ* → *TC*		*DJ* → *IN*		*IN* → *DJ*			
PB	5K	10K	5K	10K	5K	10K	5K	10K	5K	10K	5K	10K
10^{-6}	342	725	5	23	280	521	16	45	235	520	0(302)	1(608)
10^{-5}	371	714	6	20	279	478	24	32	230	509	0(302)	1(608)
10^{-4}	313	703	2	22	273	466	10	43	238	518	3(302)	8(608)
10^{-3}	384	676	7	9	265	505	19	35	237	522	15(302)	72(608)
10^{-2}	357	692	6	16	286	490	11	38	238	519	182(302)	753(608)
10^{-1}	339	754	11	17	260	495	17	38	231	518	1708(302)	7141(608)

Points and Polyhedral Surfaces in 3D

Finally, a test on some synthetic datasets has been performed in order to test the robustness rules between points and polyhedral surfaces in the 3D space. Datasets of two cardinalities have been considered: 5000 and 10000. The computed relations are: DJ, TC and IN. The DJ and TC relations on types (*pt*, *ps*) in 3D space have a *robustness level 3* (see Tab. 6) and *robustness level 1* (see Tab. 27 in App. C), respectively; thus IR1 and DR1 have to be satisfied for obtaining robustness of TC relation and IR1, IR3 and DR3 for robustness of DJ relation. Results are reported in Tab. 10, where columns are organized as described in the previous experiment, but in this case the results regard both the $5K$ and the $10K$ datasets and also the violations of DR1 and

DR3 in both datasets are shown in the last columns. There are 531 (1108) violations of IR1 and 246 (548) violation of IR3 in $5K$ $(10K)$ datasets.

The results of this experiment are consistent with the system assumptions and robustness rules presented in the previous sections, since:

- some changes of existing TC relations occurred and, according to Thm. 4 and Def. 7, in this case they are consequences of the detected IR1 violations; indeed, a higher number of IR1 violations w.r.t the number of TC relation changes was produced;
- for similar reasons, the new DJ relations produced by a change of IN relations are consequences of the detected IR3 violations and all changes of existing DJ relations are due to DR1 and DR3 violations.

As final test we considered five pairs of datasets containing randomly generated points and polyhedral surfaces in the 3D space (each dataset having 1000 geometries). Such datasets had been prepared so that IR1 and IR3 rules were always satisfied and DR1 and DR3 rules were satisfied up to a perturbation of 10^{-1}; we computed the relations between points and surfaces, resulting in 1667 TC relations and 1058 DJ relations (only the pairs with intersecting MBR were considered in the queries). Then, we simulated perturbations from 10^{-6} to 10^{-1} and recomputed the relations between points and surfaces on rounded geometries. No relation change was observed after any perturbation in any pair of datasets.

Overall, the experiment results highlight that different datasets have different behaviours with respect to robustness in topological relation evaluation. In particular, datasets $\langle RE_{north}, RJ_{north}\rangle$ and $\langle RA_{north}, VA_{north}\rangle$ showed a bad behaviour, i.e. many relations changed after perturbation, while $\langle RE_{cent}, RJ_{cent}\rangle$ and $\langle RA_{cent}, VA_{cent}\rangle$ showed a more robust behaviour, indeed even after heavy perturbations only a few relations changed. Notice that the higher robustness of *cent* datasets with respect to *north* ones is not visible during the common display and navigation operations. Finally, we observe that in the 3D space the synthetic datasets violate robustness rules even if they are randomly generated (only IR1 violations were somehow injected in the datasets), showing that the proposed rules are not trivially satisfied by a spatial dataset.

6 Conclusion and Future Work

Evaluating topological relations in distributed environments with many datasets and many systems that interoperate, such as an SDI, can be a tough task to perform. One reason of this situation regards the quality of datasets, indeed when the quality of the processed data is low, different results can be obtained in different systems regarding the computation of topological relations among geometries. This issue is usually called robustness and the goal of this paper is to guarantee the robust evaluation of topological relations among different systems. In particular, the considered context has been illustrated by means of (i) a reference vector model for representing geometries of the Simple Feature

Access model extended to the 3D space (i.e. including polyhedral surfaces); (ii) a set of basic predicates on vector geometries, that can be critical with respect in system evaluations; (iii) a set of problem definitions and assumptions on systems behaviour.

Given such context, the following results have been formally derived: (i) a set of expressions for testing the conditions of the 9-intersection matrix cells [7] in terms of vector predicates; (ii) a set of rules that guarantee the robustness of evaluation for the basic critical predicates; (iii) a set of rules that guarantees the robustness evaluation of topological relation defined with the 9-intersection matrix, with respect to the embedding space (2D or 3D) and involved geometric types. Finally, the results of some experiments on real datasets have also been presented, in order to show the effectiveness of the proposed approach in characterizing the robustness of spatial datasets with respect to topological relation evaluations.

Future work includes the definition of algorithms for rectifying geometries in spatial datasets in order to (i) preserve as much as possible the existing topological relations and (ii) satisfy the proposed rules for guaranteeing robustness in topological relations evaluation. Moreover, the study will be extended to 3D volume type.

References

1. Belussi, A., Migliorini, S., Negri, M., Pelagatti, G.: Evaluation of Topological Relations in a Discrete Vector Model. Tech. Rep. RR 91/2013, Department of Computer Science, University of Verona (2013)
2. Chen, L.: Exact Geometric Computation: Theory and Applications. Ph.D. thesis, New York University, Department of Computer Science (2001)
3. Clementini, E., Di Felice, P.: A Comparison of Methods for Representing Topological Relationships. Inf. Sci. Appl. **3**(3), 149–178 (1995)
4. Clementini, E., Felice, P.D., Oosterom, P.v.: A Small Set of Formal Topological Relationships Suitable for End-User Interaction. In: Proceedings of the Third International Symposium on Advances in Spatial Databases, pp. 277–295. Springer-Verlag (1993)
5. Coors, V.: 3d-gis in networking environments. Computers, Environment and Urban Systems **27**(4), 345–357 (2003)
6. Egenhofer, M.J., Frank, A.U., Jackson, J.P.: A topological data model for spatial databases. In: Proceedings of the 1st Symposium on Design and Implementation of Large Spatial Databases (SSD '90), pp. 271–286 (1990)
7. Egenhofer, M.J., Franzosa, R.: Point-set topological spatial relations. International Journal of Geographical Information Systems **5**(2), 161–174 (1991)
8. Güting, R.H., Schneider, M.: Realms: A Foundation for Spatial Data Types in Database Systems. In: Int. Symp. on Advances in Spatial Databases, vol. 692, pp. 14–35 (1993)
9. Halperin, D.: Controlled Perturbation for Certified Geometric Computing with Fixed-Precision Arithmetic. In: Proceedings of the Third International Congress Conference on Mathematical Software, ICMS'10, pp. 92–95. Springer-Verlag (2010)
10. Halperin, D., Packer, E.: Iterated snap rounding. Comput. Geom. Theory Appl. **23**(2), 209–225 (2002)
11. Hobby, J.: Practical segment intersection with finite precision output. Comp. Geometry Theory and App **13**, Comp. Geometry Th. and App. (1999)
12. Molenaar, M.: A formal data structure for 3D vector maps. In: Proceedings of EGIS90, p. 770781 (1990)
13. OGC: OpenGIS Implementation Standard for Geographic Information – Simple Feature Access – Part 1: Common Architecture (2011). Version 1.2.1.

14. Open Geospatial Consortium: OGC Geography Markup Language (GML) – Extended Schema and Encoding Rules, version 3.3.0 (2012). https://portal.opengeospatial.org/files/?artifact_id=46568
15. Oracle: Oracle Spatial User's Guide and Reference 10g Release 2 (10.2) (2006). http://docs.oracle.com/cd/B19306_01/appdev.102/b14255.pdf
16. Pelagatti, G., Negri, M., Belussi, A., Migliorini, S.: From the Conceptual Design of Spatial Constraints to their Implementation in Real Systems. In: Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 448–451. ACM, New York, NY, USA (2009)
17. Pilouk, M.: Integrated modelling for 3D GIS. Ph.D. thesis, ITC, The Netherlands (1996)
18. Praing, R., Schneider, M.: Efficient Implementation Techniques for Topological Predicates on Complex Spatial Objects. Geoinformatica **12**(3), 313–356 (2008)
19. Randell, D.A., Cui, Z., Cohn, A.: A Spatial Logic Based on Regions and Connection. In: Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR'92), pp. 165–176. Morgan Kaufmann (1992)
20. Rodríguez, M.A., Brisaboa, N., Meza, J., Luaces, M.R.: Measuring Consistency with Respect to Topological Dependency Constraints. In: Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS '10, pp. 182–191. ACM, New York, NY, USA (2010)
21. The Open Source Geospatial Foundation: PostGIS 2.1 Manual (2013). http://postgis.net/stuff/postgis-2.1.pdf
22. Theobald, D.M.: Topology Revisited: Representing Spatial Relations. International Journal of Geographical Information Science **15**(8), 689–705 (2001)
23. Thompson, R.J., van Oosterom, P.: Interchange of Spatial Data-Inhibiting Factors. In: Proceeding of the 9th AGILE International Conference on Geographic Information Science (2006)
24. Zlatanova, S.: 3D GIS for Urban Development. Ph.D. thesis, ITC – Faculty of Geo-Information Science and Earth Observation, The Netherlands (2000)

A Derived Vector Predicates and Operations

This section presents the semantics of derived vector predicates and derived operations introduced in Sec. 2.1.

Table 11 Expressions for derived operations and predicates regarding vertices. In the table symbols v, $v_1, \ldots, v_n$ denote a vertex, while V, V_1, V_2 are sets of vertices.

Signature	**Derivation expression**	**Dependency**
boolean $v.bel(V)$	$v.eq(v_1) \lor \cdots \lor v.eq(v_n)$	$v.eq(v_0)$
set(vertex) $V_1 \cap V_2$	$\{v \mid \exists v_1 \in V_1 : v.eq(v_1) \land \exists v_2 \in V_2 : v.eq(v_2)\}$	$v.eq(v_0)$
boolean $V_1 \cap V_2 \neq \emptyset$	$\exists v_1 \in V_1(\exists v_2 \in V_2(v_1.eq(v_2)))$	$v.eq(v_0)$

Table 12 Expressions for derived operations and predicates regarding segments. In the table symbols s, $s_0, s_1, \ldots, s_n$ denote a segment, while S is a set of segments.

Signature	**Derivation expression**	**Dependency**
set(vertex) $s.bnd()$	$\{s.start(), s.end()\}$	$s.start(), s.end()$
boolean $s_1.eq(s_2)$	$(s_1.start().eq(s_2.start()) \land s_1.end().eq(s_2.end())) \lor$	$v.eq(v_0)$
	$(s_1.start().eq(s_2.end()) \land s_1.end().eq(s_2.start()))$	
boolean $s.bel(S)$	$\exists s_i \in S(s.eq(s_i))$	$v.eq(v_0)$
boolean $s_1.in(s_2)$	$(s_1.start().eq(s_2.start()) \land s_2.cnt(s_1.end())) \lor$	$v.eq(v_0), s.cnt(v)$
	$(s_1.start().eq(s_2.end()) \land s_2.cnt(s_1.end())) \lor$	$s.start(), s.end()$
	$(s_1.end().eq(s_2.start()) \land s_2.cnt(s_1.start())) \lor$	
	$(s_1.start().eq(s_2.end()) \land s_2.cnt(s_1.end())) \lor$	
	$(s_2.cnt(s_1.start()) \land s_2.cnt(s_1.end()))$	
boolean $s_1.ov(s_2)$	$(s_1.cnt(s_2.start()) \land s_2.cnt(s_1.end())) \lor$	$s.cnt(v)$
	$(s_1.cnt(s_2.start()) \land s_2.cnt(s_1.start())) \lor$	$s.start(), s.end()$
	$(s_1.cnt(s_2.end()) \land s_2.cnt(s_1.end())) \lor$	
	$(s_1.cnt(s_2.end()) \land s_2.cnt(s_1.start()))$	
boolean $s_1.dj(s_2)$	$\neg s_1.int(s_2) \land \neg s_1.ov(s_2) \land$	$v.eq(v_0), s.cnt(v)$
	$\neg s_1.in(s_2) \land \neg s_1.eq(s_2) \land \neg s_2.in(s_1)$	$s.int(s_0)$
set(segments)	$s.diff(S) = \{s_i \mid s_i.in(s) \land \forall s_j \in S(s_i.dj(s_j))\} \land$	$v.eq(v_0), s.cnt(v)$
$s.diff(S)$	$S \cup s.diff(S) = S \cup \{s\}$	$s.int(s)$

Notice that the operation $s.diff(S)$ produces a result only if the set S contains some segments that overlap s, otherwise the result is always the empty geometry, since the second condition is not satisfied.

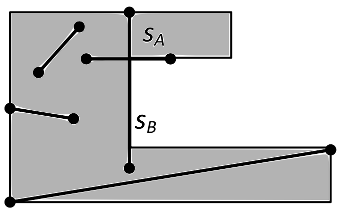


Fig. 5 Examples of possible cases that make true the predicate $p.cnt(s)$. Notice that segments s_A and s_B do not satisfy the predicate $p.cnt_{int}()$, while all the other segments do.

Table 13 Expressions for derived operations and predicates regarding patches. In the table symbols p, p_i denote patches, s, s_i denote a segment, S is a set of segments, while v, v_i denote a vertex.

Signature	**Derivation expression**	**Dependency**
set(vertex) $p.ver()$	$\bigcup_{s \in p.bnd()} s.bnd()$	$p.bnd(), s.start(),$ $s.end()$
boolean $p.cnt_{int}(s)$	$\forall v \in s.bnd()(p.cnt(v) \vee$ $\exists s_i \in p.bnd()(s_i.cnt(v) \vee$ $v.bel(s_i.bnd()))) \wedge$ $\forall s_p \in p.bnd()(\neg s.int(s_p) \wedge \neg s.ov(s_p)) \wedge$ $(s.ray(p.bnd()) \bmod 2) = 1$	$s.start(), s.end()$ $s.cnt(v), p.cnt(v)$ $s.int(s_0), s.ray(S)$ $v.eq(v_0)$
set(segments) $p.bnd_{ov}(s)$	$\{s_p \mid s_p \in p.bnd() \wedge (s_p.ov(s) \vee s_p.in(s))\}$	$p.bnd(), s.start(),$ $s.end(), s.cnt(v),$ $v.eq(v_0)$
boolean $p.cnt(s)$	$p.cnt_{int}(s) \vee \exists s_p \in p.bnd()(s_p.ov(s)) \wedge$ $\forall s_j \in (s.diff(p.bnd_{ov}(s)))(p.cnt_{int}(s_j))$	$p.bnd(), p.cnt(v),$ $s.start(), s.end(),$ $s.cnt(v), s.int(s_0),$ $s.ray(S), v.eq(v_0)$
boolean $p.ov(s)$	$(\exists v \in s.bnd()(p.cnt(v) \vee v.bel(p.ver()) \vee$ $\exists s_p \in p.bnd()(s_p.cnt(v))) \wedge$ $(\exists s_1 \in p.bnd()(s_1.int(s)) \vee$ $\exists v_i \in p.ver()(s.cnt(v_i) \wedge$ $\neg\exists s_2 \in p.bnd()(v_i.bel(s_2.bnd()) \wedge$ $(s_2.ov(s) \vee s_2.in(s))))))$ $\vee (\exists s_1 \in p.bnd()(s_1.int(s)) \wedge$ $\exists s_2 \in p.bnd()(\neg s_2.eq(s_1) \wedge s_2.int(s)))$ $\vee (\exists v_i \in p.ver()(s.cnt(v_i) \wedge$ $\neg\exists s_2 \in p.bnd()(v_i.bel(s_2.bnd()) \wedge$ $(s_2.ov(s) \vee s_2.in(s))) \wedge$ $\exists v_j \in p.ver()(\neg v_i.eq(v_j) \wedge s.cnt(v_j) \wedge$ $\neg\exists s_2 \in p.bnd()(v_j.bel(s_2.bnd()) \wedge$ $(s_2.ov(s) \wedge s_2.in(s))))))$	$p.bnd(), p.cnt(v),$ $s.start(), s.end(),$ $s.int(s_0), s.cnt(v),$ $v.eq(v_0)$
boolean $p.dj(s)$	$\neg p.int(s) \wedge \neg p.cnt(s) \wedge \neg p.ov(s)$	$p.int(s), p.bnd()$ $p.cnt(v), s.ray(S),$ $s.start(), s.end(),$ $s.int(s_0), s.cnt(v),$ $v.eq(v_0)$
boolean $p_1.eq(p_2)$	$\forall s_1 \in p_1.bnd()(\exists s_2 \in p_2.bnd()(s_1.eq(s_2))) \wedge$ $\forall s_2 \in p_2.bnd()(\exists s_1 \in p_1.bnd()(s_2.eq(s_1)))$	$p.bnd(), v.eq(v_0)$
boolean $p_1.int_2(p_2)$	$(\exists s_i \in p_1.bnd()(p_2.cnt(s_i) \vee p_2.ov(s_i)) \wedge$ $\exists s_j \in p_1.bnd()(\neg s_j.eq(s_i) \wedge$ $(p_2.cnt(s_j) \vee p_2.ov(s_j))))$ $\vee (\exists s_i \in p_2.bnd()(p_1.cnt(s_i) \vee p_1.ov(s_i)) \wedge$ $\exists s_j \in p_2.bnd()(\neg s_j.eq(s_i) \wedge$ $(p_1.cnt(s_j) \vee p_1.ov(s_j))))$	the same as $p.cnt(s)$
boolean $p_1.in(p_2)$	$(\forall s_1 \in p_1.bnd()(p_2.cnt(s_1) \vee$ $\exists s_2 \in p_2.bnd()(s_1.eq(s_2) \vee s_1.in(s_2))))$ $\wedge \neg p_1.eq(p_2)$	the same as $p.cnt(s)$
boolean $p_1.dj(p_2)$	$\neg p_1.int(p_2) \wedge \neg p_1.int_2(p_2) \wedge \neg p_1.eq(p_2)$	the same as $p.cnt(s)+$ $p_1.int(p_2)$
boolean $p_1.ov(p_2)$	$p_1.int_2(p_2) \wedge \neg p_1.in(p_2) \wedge \neg p_2.in(p_1)$ $\wedge \neg p_1.eq(p_2)$	the same as $p.cnt(s)$

Table 14 Expressions for derived operations and predicates regarding linestrings (*line*), polygons (*poly*) and polyhedral surfaces (*psur*). In the table the following notation is used: $DR(line) = ln = \{s_1, \ldots, s_k\}$ $(k > 0)$, $DR(poly) = pg = \{ring_1, \ldots, ring_l\} = \{pat_1, \ldots, pat_l\}$ $(l > 1)$ and $DR(psur) = ps = \{p_1, \ldots, p_m\}$ $(m > 0)$, where s_i is a segment, pat_i is a patch, and p_i is a polygon.

Signature	**Derivation expression**
vertex $ln.start()$	if $\neg s_1.start.eq(s_k.end())$ then $s_1.start()$ else $\emptyset$
vertex $ln.end()$	if $\neg s_1.start.eq(s_k.end())$ then $s_k.end()$ else $\emptyset$
set(vertex) $ln.bnd()$	if $\neg s_1.start.eq(s_k.end())$ then $\{s_1.start(), s_k.end()\}$ else $\emptyset$
set(vertex) $ln.intVer()$	$\{s_i.start() \mid s_i \in ln\} \setminus ln.bnd()$
boolean $ln_1.eq(ln_2)$	$\forall s_1 \in ln_1(\exists s_2 \in ln_2(s_1.eq(s_2))) \wedge$ $\forall s_2 \in ln_2(\exists s_1 \in ln_1(s_2.eq(s_1)))$
linestring $pg.extBnd()$	$ring_1$
set(linestring) $pg.intBnd()$	if $\lvert pg \rvert > 1$ then $\{ring_2, \ldots, ring_l\}$ else $\emptyset$
set(linestring) $pg.bnd()$	$\{pg.extBnd()\} \cup pg.intBnd()$
patch $pg.extPat()$	pat_1
set(patch) $pg.intPat()$	$\{pat_2, \ldots, pat_l\}$
boolean $pg_1.eq(pg_2)$	$pg_1.extBnd().eq(pg_2.extBnd()) \wedge$ $\forall ln_1 \in pg_1.intBnd()(\exists ln_2 \in pg_2.intBnd()(ln_1.eq(ln2))) \wedge$ $\forall ln_2 \in pg_2.intBnd()(\exists ln_1 \in pg_1.intBnd()(ln_1.eq(ln_2)))$
set(segment) $ps.bnd()$	$\{s \mid \exists! p \in ps(s.bel(p.bnd()))\}$
set(segment) $ps.intSeg()$	$\{s \mid \exists p \in ps(s.bel(p.bnd()) \ \wedge \ \neg s.bel(ps.bnd()))\}$
set(segment) $ps.intVer()$	$\{v \mid \exists p \in ps(\exists s \in p.bnd()(v.bel(s.bnd()))) \wedge$ $\neg\exists s' \in ps.bnd()(v.bel(s'.bnd())))\}$
boolean $ps_1.in(ps_2)$	$\forall p_i \in ps_1(\exists p_j \in ps_2(p_i.eq(p_j))$
boolean $ps_1.eq(ps_2)$	$ps_1.in(ps_2) \ \wedge \ ps_2.in(ps_1)$

B Proof Tables

This section reports the proof tables for Prop. 1. The following notation is used inside such tables: *pn*, *ln*, *pg*, and *ps* represent the discrete representation in the vector model of point, linestring, polygon and polyhedral surface, respectively. Similarly, *v*, *s*, and *p* denotes a vertex, segment, and patch, respectively.

Table 15 Proof *Interior-Interior Intersection* $(pt/*)$

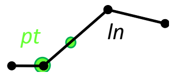
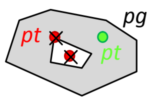
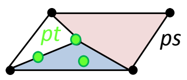
Case	**Testing conditions**	**Scene**
(pt_1/pt_2) dim=0/T	$pt_1.eq(pt_2)$	
(pt/ln) dim=0/T	$\exists s \in ln(s.cnt(pt)) \vee \ pt.bel(ln.intVer())$	
(pt/pg) dim=0/T	$pg.extPat().cnt(pt) \wedge$ $\neg\exists p \in pt.intPat()(p.cnt(pt) \ \vee \ pt.bel(p.ver())$ $\vee \ \exists s \in p.bnd()(s.cnt(pt)))$	
(pt/ps) dim=0/T	$\exists p \in ps(p.cnt(pt) \vee$ $pt.bel(ps.intVer()) \vee$ $\exists s \in ps.intSeg()(s.cnt(pt))))$	

Table 16 Proof *Interior-Interior intersection* (ln/pg)

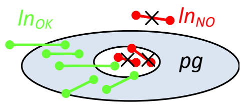
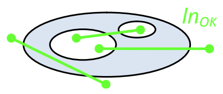
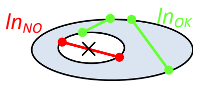
Case	Testing conditions	Scene
ln/pg dim=1/T	$\exists s \in ln(\exists v \in s.bnd()(pg.extPat().cnt(v) \wedge \neg\exists p \in pg.intPat()(p.cnt(v) \vee \exists s_1 \in p.bnd(s_1.cnt(v)) \vee v.bel(p.ver())$	
	$\exists s \in ln(\exists ln_1 \in pg.bnd()(\exists s_1 \in ln_1(s.int(s_1))))$	
	$\exists s \in ln(pg.extPat().cnt(s) \neg\exists p \in pg.intPat()(p.cnt(s)))$	

Table 17 Proof *Interior-Interior intersection* (ps/ps) dim=T – (only in 3D space).

Case	Necessary conditions	Scene
ps_1/ps_2 dim=T	$\exists p_1 \in ps_1(\exists p_2 \in ps_2(p_1.eq(p_2) \vee p_1.int_2(p_2)))$	same last row of Table 22
	$\exists s \in ps_1.intSeg()(\exists p \in ps_2(p.ov(s) \vee p.cnt(s)))\vee$ $\exists s \in ps_2.intSeg()(\exists p \in ps_1(p.ov(s) \vee p.cnt(s)))$	see Table 22
	$\exists s_1 \in ps_1.intSeg()(\exists s_2 \in ps_2.intSeg()(s_1.ov(s_2) \vee s_1.int(s_2) \vee s_2.in(s_1) \vee s_1.eq(s_2)\vee s_1.int(s_2)))$	see Table 22
	$\exists v \in ps_1.intVer()(\exists p \in ps_2(p.cnt(v)\vee \exists s \in p.bnd()(s.cnt(v))))\vee$ $\exists v \in ps_2.intVer()(\exists p \in ps_1(p.cnt(v)\vee \exists s \in p.bnd()(s.cnt(v))))$	see Table 22
	$\exists v_1 \in ps_1.intVer()(\exists v_2 \in ps_2.intVer()(v_1.eq(v_2)))$	see Table 22

Table 18 Proof *Interior-Interior Intersection* (*ln*/*ln*) Notice that testing conditions are presented on several rows: each row describes a disjunct which is a conjuction of a necessary condition and an additional one (usually the last one is used to take into account the dimension requirement)

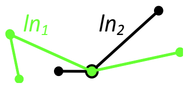
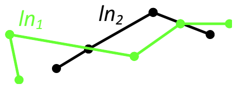
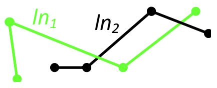
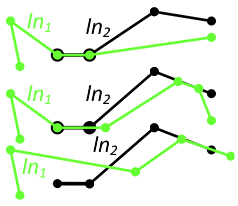
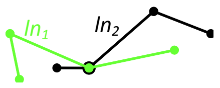
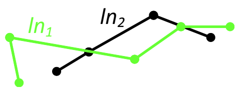
Case	Necessary conditions	Additional conditions	Scene
ln_1/ln_2 $dim{=}0$	$ln_1.intVer() \cap$ $ln_2.intVer() \neq \emptyset$	$\forall s_1 \in ln_1(\forall s_2 \in ln_2($ $s_1.int(s_2) \vee s_1.dj(s_2)))$	
	$\exists v \in ln_1.intVer()($ $\exists s \in ln_2(s.cnt(v))) \vee$ $\exists v \in ln_2.intVer()($ $\exists s \in ln_1(s.cnt(v)))$	$\forall s_1 \in ln_1(\forall s_2 \in ln_2($ $s_1.int(s_2) \vee s_1.dj(s_2)))$	
	$\exists s_1 \in ln_1(\exists s_2 \in ln_2$ $(s_1.in(s_2)))$	$\forall s_1 \in ln_1(\forall s_2 \in ln_2($ $s_1.int(s_2) \vee s_1.dj(s_2)))$	
ln_1/ln_2 $dim{=}1$	$\exists s_1 \in ln_1(\exists s_2 \in ln_2$ $(s_1.eq(s_2) \vee s_1.in(s_2) \vee$ $s_2.in(s_1) \vee s_1.ov(s_2))$		
ln_1/ln_2 $dim{=}T$	$ln_1.IntVer() \cap$ $ln_2.IntVer() \neq \emptyset$		
	$\exists v \in ln_1.intVer()($ $\exists s \in ln_2(s.cnt(v))) \vee$ $\exists v \in ln_2.intVer()($ $\exists s \in ln_1(s.cnt(v)))$		
	$\exists s_1 \in ln_1(\exists s_2 \in ln_2$ $(s_1.eq(s_2) \vee s_1.in(s_2) \vee$ $s_2.in(s_1) \vee s_1.int(s_2) \vee$ $s_1.ov(s_2)))$		the same scenes of the 3rd and 4th row.

Table 19 Proof *Interior-Boundary intersection* (*pt*/*ln*) and (*ln*/*ln*) –

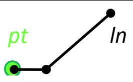
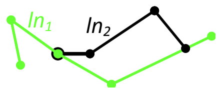
Case	Testing conditions	Scene
pt/ln $dim{=}0/T$	$pt.bel(ln.bnd())$	
ln_1/ln_2 $dim{=}0/T$	$\exists v \in ln_2.bnd()(v.bel(ln_1.intVer()) \vee$ $\exists s \in ln_1(s.cnt(v)))$	

Table 20 Proof *Interior-Interior Intersection* (ln/ps) – (only in 3D space)

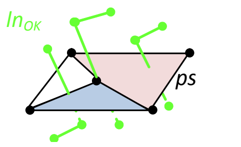
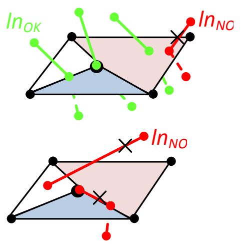
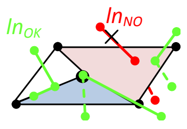
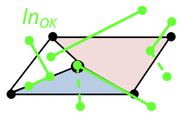
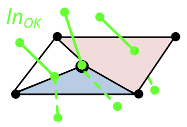
Case	**Necessary conditions**	**Additional conditions**	**Scene**
ln/ps $dim = 0$	$\exists s \in ln(\exists p \in ps($ $p.int(s))\vee$ $\exists s_1 \in ps.intSeg()($ $s.int(s_1))\vee$ $\exists v_1 \in ps.intVer()($ $s.cnt(v_1)))$	$\forall s \in ln(\forall p \in ps$ $(p.int(s) \vee p.dj(s)))\wedge$ $\forall s \in ln(\forall s_1 \in ps.intSeg()$ $(s.dj(s_1) \vee s.int(s_1)))$	
	$\exists v \in ln.intVer()($ $\exists p \in ps(p.cnt(v))\vee$ $\exists s_1 \in ps.intSeg()($ $s_1.cnt(v))\vee$ $\exists v_1 \in ps.intVer()($ $v.eq(v_1)))$	$\forall s \in ln(\forall p \in ps$ $(p.int(s) \vee p.dj(s)))\wedge$ $\forall s \in ln(\forall s_1 \in ps.intSeg()$ $(s.dj(s_1) \vee s.int(s_1)))\wedge$	
ln/ps $dim = 1$	$\exists s \in ln(\exists p \in ps($ $p.cnt(s) \vee p.ov(s))\vee$ $\exists s_1 \in ps.intSeg()($ $s.ov(s_1) \vee s_1.in(s)\vee$ $s.eq(s_1)))$		
ln/ps dim=T	$\exists s \in ln(\exists p \in ps(\neg p.dj(s))\vee$ $\exists s_1 \in ps.intSeg()($ $s.ov(s_1) \vee s.in(s_1)\vee$ $s1.in(s) \vee s.eq(s_1) \vee s.int(s_1))\vee$ $\exists v_1 \in ps.intVer()(s.cnt(v_1))$		
	$\exists v \in ln.intVer()($ $\exists p \in ps(p.cnt(v))\vee$ $\exists s_1 \in ps.intSeg()($ $s_1.cnt(v))\vee$ $\exists v_1 \in ps.intVer()($ $v.eq(v_1)))$		

Table 21 Proof *Interior-Interior Intersection* $(pg_1/pg_2\ )$ – (only in 2D space)

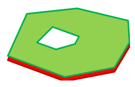
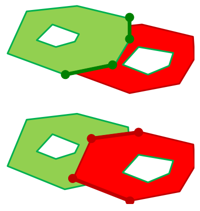
Case	**Testing conditions**	**Scene**
pg_1/pg_2 dim=2/T	$pg_1.extPat().eq(pg_2.exPat())$	
	$\exists s \in pg_1.bnd()((pg_2.extPat().ov(s)\vee$ $pg_2.extPat().cnt(s))\wedge$ $\neg\exists p \in pg_2.intPat()(p.cnt(s)))\vee$ $\exists s \in pg_2.bnd()((pg_1.extPat().ov(s)\vee$ $pg_1.extPat().cnt(s))\wedge$ $\neg\exists p \in pg_1.intPat()(p.cnt(s)))$	

Table 22 Proof *Interior-Interior intersection* (ps_1/ps_2) dim=0/1/2 – (only in 3D space).

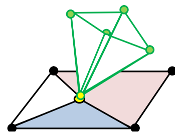
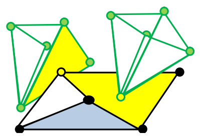
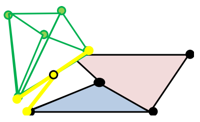
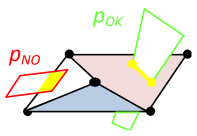
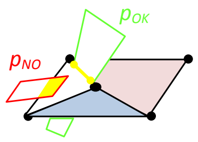
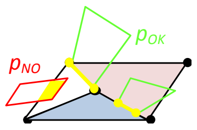
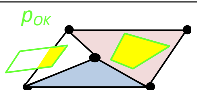
Case	Necessary conditions	Additional conditions	Scene
ps_1/ps_2 *dim*=0	$\exists v \in ps_1.\mathit{intVer}()(v.\mathit{bel}(ps_2.\mathit{intVer}()))$	$\forall p_1 \in ps_1(\forall p_2 \in ps_2(p_1.\mathit{dj}(p_2)))$	
	$\exists v \in ps_1.\mathit{intVer}()(\exists p \in ps_2(p.\mathit{cnt}(v)\vee \exists s \in p.\mathit{bnd}()(s.\mathit{cnt}(v))))\vee \exists v \in ps_2.\mathit{intVer}()(\exists p \in ps_1(p.\mathit{cnt}(v)\vee \exists s \in p.\mathit{bnd}()(s.\mathit{cnt}(v))))$	$\forall p_1 \in ps_1(\forall p_2 \in ps_2(p_1.\mathit{dj}(p_2)))$	
	$\exists s_1 \in ps_1.\mathit{intSeg}()(\exists s_2 \in ps_2.\mathit{intSeg}()(s_1.\mathit{int}(s_2)))$	$\forall p_1 \in ps_1(\forall p_2 \in ps_2(p_1.\mathit{dj}(p_2)))$	
ps_1/ps_2 *dim*=1	$\exists p_1 \in ps_1(\exists p_2 \in ps_2(p_1.\mathit{int}(p_2)))$	$\forall p_1 \in ps_1(\forall p_2 \in ps_2(p_1.\mathit{dj}(p_2) \vee p_1.\mathit{int}(p_2)$	
	$\exists s \in ps_1.\mathit{intSeg}()(\exists p \in ps_2(p.\mathit{cnt}(s)\vee p.\mathit{ov}(s)))\vee \exists s \in ps_2.\mathit{intSeg}()(\exists p \in ps_1(p.\mathit{cnt}(s)\vee p.\mathit{ov}(s)))$	$\forall p_1 \in ps_1(\forall p_2 \in ps_2(p_1.\mathit{dj}(p_2) \vee p_1.\mathit{int}(p_2)$	
	$\exists s_1 \in ps_1.\mathit{intSeg}()(\exists s_2 \in ps_2.\mathit{intSeg}()(s_1.\mathit{ov}(s_2) \vee s_1.\mathit{in}(s_2)\vee s_1.\mathit{eq}(s_2)))$	$\forall p_1 \in ps_1(\forall p_2 \in ps_2(p_1.\mathit{dj}(p_2) \vee p_1.\mathit{int}(p_2)$	
ps_1/ps_2 *dim*=2	$\exists p_1 \in ps_1(\exists p_2 \in ps_2(p_1.\mathit{eq}(p_2) \vee p_1.\mathit{int}_2(p_2)))$		

Table 23 Proof *Interior-Exterior intersection* (pt/ln), (pt/pg) and (pt/ps)

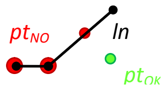
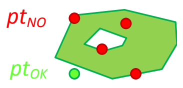
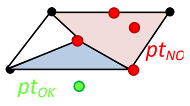
Case	Testing conditions	Scene
pt/ln $dim{=}0/\mathrm{T}$	$\forall v \in ln.bnd()(\neg pt.eq(v))\wedge$ $\forall v_i \in ln.intVer()(\neg pt.eq(v_i))$ $\forall s \in ln(\neg s.cnt(v)))$	
pt/pg $dim{=}0/\mathrm{T}$	$(\neg pg.extPat().cnt(pt)\wedge$ $\exists p \in pg.intPat()(p.cnt(pt)))\wedge$ $\forall ln \in pg.bnd()(\forall s \in ln(\neg s.cnt(pt)\wedge$ $\forall v \in s.bnd()(\neg pt.eq(v)))$	
pt/ps $dim{=}0/\mathrm{T}$	$\forall p \in ps(\neg p.cnt(pt))\wedge$ $\forall s \in ps.bnd()(\neg s.cnt(pt)\wedge$ $\forall v \in s.bnd()(\neg pt.eq(v)))$	

Table 24 Proof *Interior-Exterior intersection* (ln/ln), (ln/pg) and (ln/ps)

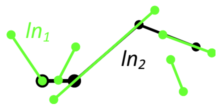
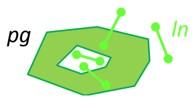
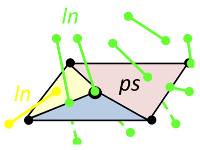
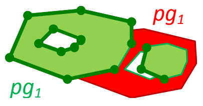
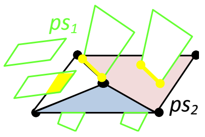
Case	Additional conditions	Scene
ln_1/ln_2 $dim{=}1/\mathrm{T}$	$\exists s_1 \in ln_1(\forall s_2 \in ln_2(s_1.dj(s_2)\vee$ $s_1.int(s_2) \vee s_1.ov(s_2) \vee s_2.in(s_1)))$	
ln/pg $dim{=}1/\mathrm{T}$	$\exists s \in ln(\neg pg.extPat().cnt(s))\vee$ $\exists s \in ln(\exists p \in pg.intPat()(p.cnt(s)))\vee$ $\exists ln \in pg.bnd()(\exists s_1 \in ln(s.int(s_1)))$	
ln/ps $dim{=}1/\mathrm{T}$	$\exists s \in ln(\forall p \in ps($ $p.int(s) \vee p.dj(s) \vee p.ov(s))\wedge$ $\neg\exists s_2 \in ps.intSeg()(s.in(s_2) \vee s.eq(s_2)))$	

Table 25 Proof *Interior-Exterior intersection* (pg/pg) and (ps/ps)– Case C.3.11 and C.3.16.

Case	**Additional conditions**	**Scene**
pg_1/pg_2 dim=2/T	$\exists s \in pg_1.extPat().bnd()($ $(\neg pg_2.extPat().cnt(s) \vee$ $(pg_2.extPat().cnt(s) \wedge$ $\exists p \in pg_2.intPat()(p.ov(s) \vee p.cnt(s)))))$	
ps_1/ps_2 dim=2/T	$\exists p_1 \in ps_1(\forall p_2 \in ps_2(p_1.dj(p_2) \vee$ $p_1.ov(p_2) \vee p1.int(p_2)))$	

C Robustness Levels of the Relation *Touches*

This section analyses the robustness behaviour of the relation *Touches*. Tables 26 and 27 shows the minimum and maximum level of robustness required for all intersection tests. The notation $f(c_{xy})$ means that the cell is a derived cell and its test can be avoided since it is implied by the test of c_{xy}. The term RobLev stands for robustness level.

Table 26 Analysis of the robustness behavior of $a.TC(b)$ in 2D.

2D	Point (PT)	LineString (LS)	Polygon (PL)
PT	NA	$R_{tc}(pt, ln)$= $[F\underline{T}F\ \mathbf{FFF}\ \mathbf{T}T\mathbf{T}]$ $\begin{bmatrix} f(c_{1,2}) & 0 & f(c_{1,2}) \\ F & F & F \\ T & f(c_{1,2}) & T \end{bmatrix}$ RobLev = [0,0]	$R_{tc}(pt, pg)$= $[F\underline{T}F\ \mathbf{FFF}\ \mathbf{TTT}]$ $\begin{bmatrix} f(c_{1,2}) & 1 & f(c_{1,2}) \\ F & F & F \\ T & T & T \end{bmatrix}$ RobLev = [1,1]
LS	$R_{tc}(ln, pt) =$ $R_{tc}(pt, ln)^T$ RobLev = [0,0]	$R_{tc}(ln, ln) = [\underline{F}{*}T\ {*}\underline{T}{*}\ T{*}\mathbf{T}]$ $\begin{bmatrix} 1 & * & f(c_{1,1}) \\ * & 0 & * \\ f(c_{1,1}) & * & T \end{bmatrix}$ $R_{tc}(ln, ln) =[\underline{FT}T\ {*}{*}{*}\ T{*}\mathbf{T}]$ $\begin{bmatrix} 1 & 1 & f(c_{1,1}) \\ * & * & * \\ f(c_{1,1}) & * & T \end{bmatrix}$ RobLev = [0,1]	$R_{tc}(ln, pg) = [\underline{F}{*}{*}\ F\underline{T}{*}\ \mathbf{T}{*}\mathbf{T}]$ $\begin{bmatrix} 1 & * & * \\ f(c_{1,1}) & 1 & * \\ T & * & T \end{bmatrix}$ $R_{tc}(ln, pg) =[\underline{FT}{*}\ F{*}{*}\ \mathbf{T}{*}\mathbf{T}]$ $\begin{bmatrix} 1 & 1 & * \\ f(c_{1,1}) & * & * \\ T & * & T \end{bmatrix}$ RobLev = [1,1]
PL	$R_{tc}(pg, pt) =$ $R_{tc}(pt, pg)^T$ RobLev = [1,1]	$R_{tc}(pg, ln) = R_{tc}(ln, pg)^T$ RobLev = [1,1]	$R_{tc}(pg, pg) =[\underline{F}FT\ F\underline{T}T\ TT\mathbf{T}]$ $\begin{bmatrix} 1 & f(c_{1,1}) & f(c_{1,1}) \\ f(c_{1,1}) & 1 & f(c_{1,1}) \\ f(c_{1,1}) & f(c_{1,1}) & T \end{bmatrix}$ RobLev = [1,1]

Table 27 Analysis of the robustness behavior of $a.TC(b)$ in 3D.

3D	Point (PT)	LineString (LS)	Polyhedral Surface (PS)
PT	NA	Same as 2D RobLev = [0,0]	Same as 2D RobLev = [1,1]
LS	Same as 2D RobLev = [0,0]	$R_{tc}(ln, ln) = [\underline{\mathrm{F}}{*}T\ {*}\underline{\mathrm{T}}{*}\ T{*}\mathbf{T}]$ $\begin{bmatrix} 2 & * & f(c_{1,1}) \\ * & 0 & * \\ f(c_{1,1}) & * & T \end{bmatrix}$ $R_{tc}(ln, ln) = [\underline{\mathrm{FT}}T\ {*}{*}{*}\ T{*}\mathbf{T}]$ $\begin{bmatrix} 2 & 1 & f(c_{1,1}) \\ * & * & * \\ f(c_{1,1}) & * & T \end{bmatrix}$ RobLev = [0,2]	$R_{tc}(ln, ps) = [\underline{\mathrm{F}}{*}{*}\ {*}\underline{\mathrm{T}}{*}\ \mathbf{T}{*}\mathbf{T}]$ $\begin{bmatrix} 4 & * & * \\ * & 1 & * \\ T & * & T \end{bmatrix}$ $R_{tc}(ln, ps) = [\underline{\mathrm{FT}}\ {*}{*}{*}\ \mathbf{T}{*}\mathbf{T}]$ $\begin{bmatrix} 4 & 2 & * \\ * & * & * \\ T & * & T \end{bmatrix}$ $R_{tc}(ln, ps) = [\underline{\mathrm{F}}{*}{*}\ \underline{\mathrm{T}}{*}{*}\ \mathbf{T}{*}\mathbf{T}]$ $\begin{bmatrix} 4 & * & * \\ 3 & * & * \\ T & * & T \end{bmatrix}$ RobLev = [1,4]
PS	$R_{tc}(ps, pt) = R_{tc}(pt, ps)^T$ RobLev = [1,1]	$R_{tc}(ps, ln) = R_{dj}(ln, ps)^T$ RobLev = [2,4]	$R_{tc}(ps, ps) = [\underline{\mathrm{F}}{*}T\ {*}\underline{\mathrm{T}}{*}\ T{*}\mathbf{T}]$ $\begin{bmatrix} 4 & * & f(c_{1,1}) \\ * & 2 & * \\ f(c_{1,1}) & * & T \end{bmatrix}$ $R_{tc}(ps, ps) = [\underline{\mathrm{F}}{*}T\ \underline{\mathrm{T}}{*}{*}\ T{*}\mathbf{T}]$ $\begin{bmatrix} 4 & * & f(c_{1,1}) \\ 4 & * & * \\ f(c_{1,1}) & * & T \end{bmatrix}$ $R_{tc}(ps, ps) = [\underline{\mathrm{FT}}T\ {*}{*}{*}\ T{*}\mathbf{T}]$ $\begin{bmatrix} 4 & 4 & f(c_{1,1}) \\ * & * & * \\ f(c_{1,1}) & * & T \end{bmatrix}$ RobLev = [2,4]