Evaluation of Topological Relations in a Discrete Vector Model

Alberto Belussi
Sara Migliorini
Mauro Negri
Giuseppe Pelagatti

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 [6, 3, 2]. 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. Moreover, in a Spatial Data Infrastructure (SDI) the perturbations introduced by the exchange of data between different systems can increase the robustness problems. The aim of this report is to define an implementation of topological relations with reference to a discrete vector model commonly adopted by today's systems.

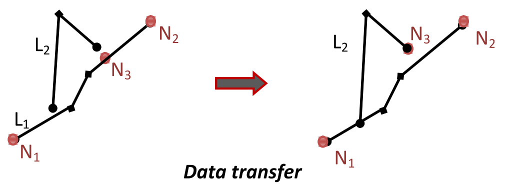
Keywords: Robustness
Topological relations
Spatial Data Infrastructure
Discrete representation
Distributed systems

1 Introduction

Topological relationships are a fundamental formal tool for describing spatial properties of data in geographical applications: this occurs for example in schema definitions, where spatial integrity constraints have to be defined, and also in query specification, where spatial filters have to be used for retrieving information of interest for the user, and in update processes where topological relations are used to specify data quality [17, 13].

Although many abstract models have been studied in literature [6, 3, 2] for defining the semantics of topological relationships between geometric objects embedded in an Euclidean space, the problems arising when topological relationships are evaluated on real data have been much less explored. In particular, topological relations have been defined by using the 9-intersection matrix approach [6] or other axiomatic approaches [16], while for their evaluation specific computational geometry algorithms have been implemented in systems which work on real data represented as vectors in a discrete space.

A consequence of this fact is that the evaluation of topological relations can be non robust, i.e. it can produce different results on the same data in different contexts. 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 Euclidean space, instead of the real numbers theoretically required, is well known [1, 10]. This report considers the distributed and heterogeneous context of a Spatial Data Infrastructure (SDI) in which the problems related to the adopted finite number representation are made even worse by the data perturbation occurring during the exchange between different systems. Such exchanges can introduce perturbations in geometric representation as a result of the conversions between different formats and precisions.



System S_1 evaluation:	System S_2 evaluation:
N_1 *Touches* L_1,	N_1 *Touches* L_1,
N_2 *Touches* L_1,	N_2 *Disjoint* L_1,
N_3 *In* L_1,	N_3 *Disjoint* L_1,
N_3 *Disjoint* L_2,	N_3 *Touches* L_2,
L_2 *Disjoint* L_1	L_2 *Cross* L_1,

Figure 1: Non robust evaluation of topological relations.

For example, consider a spatial database containing some instances of two feature types *Road Link* and *Road Node*, which are transferred from a system S_1 to a system S_2. Fig. 1 shows that different topological relations can be computed by the two systems between the same pairs of object instances. For the sake of this example the intuitive meaning of the topological relations *In*, *Touches* and *Disjoint* is referred to; their exact meaning will be defined later. Fig. 1 assumes that node N_3 and one endpoint of link L_2 have been slightly perturbed by the data transfer, hence giving rise to the different evaluations of the relations between N_3 and L_2, N_3 and L_1, and L_1 and L_2, while the different evaluation of the relation between N_2 and L_1 is due to different results in computation.

The robustness problem in the evaluation of topological relations is also related to the dimension of the geometric space of the objects (2D or 3D). For example, consider two curves which have a clear intersection in 2D: in this space every system will likely evaluate the topological relation as a *Crosses*, but in a 3D space a small difference in the z-value could cause some system to evaluate the relation as *Disjoint* while others would evaluate it as *Crosses.* Therefore, in many cases a distinct analysis of the robustness of topological relations in 2D and 3D is necessary.

In literature several robustness rules have been proposed in order to solve the mentioned robustness problems, and they are to some extent applied by real systems. The most important one is based on the identification of common geometric primitives between different objects. These common primitives can be either stored once and referred to by the objects (topological structures [5]) 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 [19], consists in ensuring that a minimum distance is kept between all geometric primitives which are not identical.

The goal of this report is to define an implementation of topological relations using a discrete vector model with is commonly adopted by available systems. The remainder of the report is organized as follows: Sec. 2 presents some related results contained in literature. Sec. 3 introduces the geometric model and the topological relations considered in the following. Sec. 4 formalizes the reference discrete vector model, which implements the presented geometric model, together with a set of elementary predicates, called critical vector predicates. A *critical vector predicate* has the following characteristics: it is an elementary predicate which can be evaluated in the discrete vector model, it is necessary in order to implement some topological relation and it is critical because its evaluation can be not robust in the defined reference model. Using the definitions of the previous sections, Sec. 5 and Sec. 6 shows how each cell of the 9-intersection matrix [6, 3], which describes a topological relation among two geometries, can be computed by evaluating a given logical expression containing some vector predicates. Finally, Sec. 7 discusses conclusions and future work.

2 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 [1] 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) [8] 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 [10] and [9], 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 [5, 18]). 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) [11] 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 [14]. This model includes 4 primitives: tetrahedron, triangle, arc, and node, where the first one is a real 3D primitive. The Simplified Spatial Model (SSM) [20] 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) [4] represents the geometry of a body or of a surface using planar convex faces, defined as sets of nodes. In [7] the author defines the 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 [4] 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 [15]. The authors considers 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 [4, 15] 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.

3 Geometric Model and Topological Relations

The geometric model considered in this report is compliant with the last Simple Feature Access (SFA) model of OGC [12]. It contains classes describing geometries of the 2D space, but with the possibility to store also the z coordinate usually by representing the height above or below sea level. Moreover, the type *PolyhedralSurface* is available for representing 3D surfaces, as sets of polygon patches with some constraints. The complete type hierarchy is shown in Fig. 3.

The main characteristics of these types are supposed to be known by the reader, please refer to [12] for more details. Anyway, since polyhedral surfaces are the most complex geometries of the model, their main characteristics are briefly described in Sec. 3.1. Finally, notice that while *Point* and *LineString* geometries can be embedded in 2D or 3D spaces, *Polygon* geometries can be embedded only in 2D space and *PolyhedralSurface* only in 3D space. Another constraint regards *LineString* geometries for which successive collinear segments are not admitted. In the same model a reference set of topological relations is proposed which is presented in Sec. 3.2.

3.1 Polyhedral Surface

A geometric object described by a polyhedral surface is a three-dimensional surface embedded in 3D space. A surface of this type is connected and is defined by a set of surface patches, which satisfies some properties.

A *surface patch* is an elementary and planar three-dimensional surface defined in 3D space by assigning a planar ring; this ring, called *fe*, represents the external boundary of the surface patch. Notice that a patch can also be a triangle (producing TIN), but it does not have to. Moreover, without loss

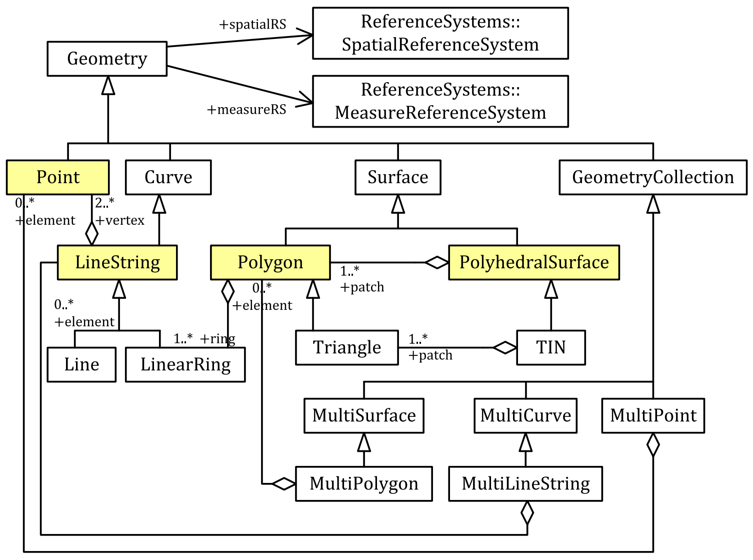


Figure 2: Geometric type hierarchy of the Simple Feature Access (SFA) model of OGC [12].

of generality, the following simplification is introduced w.r.t. the standard: no holes are admitted in a patch.

The boundary of a polyhedral surface is the set of *LineString* that are boundary of one and only one patch of the surface. If the boundary is the empty set, the surface is a cycle and implicitly defines a solid ($phSurface.isCycle()?\ phSurface.boundary().isEmpty()$). Finally, by definition a polyhedral surface is always simple ($phSurface.isSimple() = true$). An example of polyhedral surface is shown in Fig. 3.

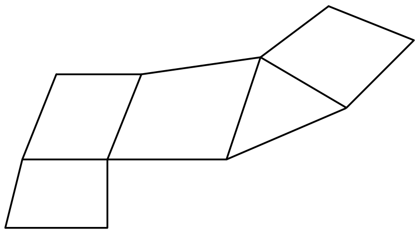


Figure 3: An example of polyhedral surface.

The following assumptions are also considered through the report: adjacent coplanar patches are not admitted, since they can be substituted by the patch obtained by merging them, and a polyhedral surface is always a 2-manifold.

3.2 Topological Relation

In order to describe the spatial relations existing between two geometries, it is necessary to use a reference model. This paper considers the well known approach of Egenhofer et al. [6] based on the 9-intersection matrix. Topological relations of the SFA architecture are defined on the basis of this approach, but the framework proposed in this paper can be applied to any mutually exclusive set of topological relations, that can be defined by means of sets of 9-intersection matrices and operates on the set of geometric types of the SFA architecture.

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 1 (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) [3], by assigning the following matrix:

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

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 order to have a set of topological relations for presenting the examples in this report, the set REL_{tp} is considered which is made up of the relations: *Disjoint* (DJ), *Touches* (TC), *In* (IN), *Contains* (CT), *Equals* (EQ), *Overlaps* (OV) and *Crosses* (CR) as defined in [3]. This set has the following characteristics:

- Its constituent relations are mutually exclusive: if the relation $r \in REL_{tp}$ is valid between two geometric objects, no other relation of that set is satisfied.

- The set is complete: given two geometric objects in a certain spatial scene, the set will always have a relationship that is true in that scene.
- Relations apply to objects of the same type or of different types.
- Relations can be applied only between objects defined in the same space (2D or 3D); comparison between objects defined in different spaces is not supported.

Definition 2 (Reference set of topological relations REL_{tp})**.** The formal definition of each relation in REL_{tp} 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 surface, while *a*, *b* are generic geometries.

- **DJ**: $a.DJ(b) \triangleq a.PS() \cap b.PS() = \emptyset$
  $R_{dj}(pt,pt) = [FFT\ FFF\ TFT]$
  $R_{dj}(pt,c/s) = [FFT\ FFF\ T*T]$
  $R_{dj}(c/s,pt) = R_{dj}(pt,c/s)^T$
  $R_{dj}(c/s,c/s) = [FFT\ FF*\ T*T]$
- **TC**: $a.TC(b) \triangleq (I(a) \cap I(b) = \emptyset) \wedge (a.PS() \cap b.PS() \neq \emptyset)$
  $R_{tc}(pt,c/s) = [FTF\ FFF\ TTT]$
  $R_{tc}(c/s,pt) = R_{tc}(pt,c/s)^T$
  $R_{tc}(c,s) = [F{*}{*}\ {*}T{*}\ T{*}T] \cup [FT{*}\ {*}{*}{*}\ T{*}T] \cup [F{*}{*}\ T{*}{*}\ T{*}T]^1$
  $R_{tc}(s,c) = R_{tc}(c,s)^T$
  $R_{tc}(c,c) = [F{*}T\ {*}T{*}\ T{*}T] \cup [F{*}T\ T{*}{*}\ T{*}T] \cup [FTT\ {*}{*}{*}\ T{*}T]$
  $R_{tc}(s,s) = [F{*}T\ {*}T{*}\ T{*}T] \cup [F{*}T\ T{*}{*}\ T{*}T]^1 \cup [FTT\ {*}{*}{*}\ T{*}T]^1$
- **IN**: $a.IN(b) \triangleq (a.PS() \cup b.PS() = a.PS()) \wedge (I(a) \cap I(b) \neq \emptyset) \wedge$
  $(a.PS() \cap b.PS() \neq b.PS())$
  $R_{in}(pt,c/s) = [TFF\ FFF\ T*T]$
  $R_{in}(c,c/s) = [T*F\ \ {*}{*}F\ T*T]$
  $R_{in}(s,s) = [TFF\ T*F\ T*T]$
- **CT**: $a.CT(b) \triangleq b.IN(a)$
- **EQ**: $a.EQ(b) \triangleq (a.PS() \cap b.PS() = a.PS()) \wedge (a.PS() \cap b.PS() = b.PS())$
  $R_{eq}(a,b) = [TFF\ FTF\ FFT]$
- **OV**: $a.OV(b) \triangleq (I(a) \cap I(b) \neq \emptyset) \wedge$
  $(dim(a) = dim(b) = dim(I(a) \cap I(b))) \wedge$
  $(a.PS() \cap b.PS() \neq a.PS()) \wedge (a.PS() \cap b.PS() \neq$
  $b.PS()$

[1]only in 3D spaces

$R_{ov}(s,s) = [2*T \;\; *** \;\; T*T]$
$R_{ov}(c,c) = [1*T \;\; *** \;\; T*T]$

- **CR**: $a.CR(b) \triangleq (I(a) \cap I(b) \neq \emptyset) \wedge$
  $(dim(a) \cap dim(b) < \max(dim(a), dim(b))) \wedge$
  $(a.PS() \cap b.PS() \neq a.PS()) \wedge (a.PS() \cap b.PS() \neq b.PS()$
  $R_{cr}(c,s) = [T*T \;\; *** \;\; T*T]$
  $R_{cr}(s,c) = R_{cr}(c,s)^T$
  $R_{cr}(c,c) = [0*T \;\; *** \;\; T*T]$
  $R_{cr}(s,s) = [1*T \;\; *** \;\; T*T] \cup [0*T \;\; *** \;\; T*T]$ □

4 Discrete Vector Model

This section presents a discrete vector model containing both 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 has to be considered as a formal description of a possible implementation of a part of the SFA model, which indeed is an abstract specification. In particular, we consider the SFA model reduces to *Point*, *LineString*, *Polygon* and *PolyhedralSurface* types and we focus on the implementation of the tests that are necessary for evaluating the topological relations described by the 9-intersection matrix presented in Def. 1.

In a 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 requires a computation on finite numbers thus having a direct effect on the robustness of topological relation evaluation.

The model is composed of some vector types that are used to implement the considered SFA types, some basic predicates and operations and some derived predicate an operations. The following definitions formalize the vector types.

Definition 3 (Vector types)**.** The discrete vector model considered in this report is characterized by the following 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 $v_1 = v_n$, namely it is a cycle.
- A *patch* is a planar polygon whose boundary is defined by a ring, i.e $(v_1, \ldots, v_n)$ with $v_1 = v_n$. $\square$

The discrete representation of a *Point* corresponds to a single vertex, while *LineString*, *Polygon* and *PolyhedralSurface* are described as sets of vertices where a certain interpolation method is applied between consecutive vertices. In particular, the linear interpolation is the most frequently used.

Definition 4 (Discrete geometry representation)**.** Given a geometry g, 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*$(g) = (r_1, , r_k)$ 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. Indeed, without loss of generality this assumes that no holes are admitted inside a patch.
  Using the patch definition, the discrete representation of a *Polyhedral-Surface* (*Polygon*) 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 $(v_{i,1}, \ldots, v_{i,n_i})$. $\square$

Considering that we are interesting in identifying the operations that are necessary for performing the processing required to test topological relations on the considered SFA types, since we need to study their robustness, we list hereby the set of operations and predicates that we consider as the set of *basic* tools for obtaining this goal. Indeed, each of them identifies a type of processing on finite numbers that is required in many cases during topological relation evaluation and that has to be analyzed for accessing its robustness.

Definition 5 (Basic vector operations)**.** The signature of each operation and predicate follows the syntax $\langle$*ret type*$\rangle$ $\langle$*geom*$\rangle$.$\langle$*op name*$\rangle$ ($\langle$*par type*$\rangle$ $\langle$*par name*$\rangle$), where *ret type* is the return type, *geom* is the geometric object, *op name* is the operation name, *par type* is the parameter type and *par name* is the parameter name:

Operations

- *real* $v.dist$(*vertex* v_0): it returns the Euclidean distance between a vertex v and another vertex v_0.
- *vertex* $s.start$(): it returns the start point of the segment s.
- *vertex* $s.end$(): it returns the end point of the segment s.
- *vertex* $s.midpnt$(): it returns the midpoint of the segment s.
- *segment* $s_1 \cup \cdots \cup s_n$: it joins two 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.
- *set(segment)* $p.bnd$(): it returns the set of segments defining the boundary of the patch p.

Predicates

- *boolean* $v.eq$(*vertex* v_0): it is a test of equality between two vertexes. In particular, two vertices are equal if they are bitwise identical.
- *boolean* $s.cnt$(*vertex* v): it is a test of containment between a vertex and a segment interior: $v \subset I(s)$.
- *boolean* $s.int$(*segment* s_0): it is a test of intersection between the interiors of two segments: $dim(I(s) \cap I(s_0)) = 0$. If the intersection has dimension 1, it returns false.
- *boolean* $p.cnt$(*vertex* v): it is a test of inclusion between a vertex v and the interior of a patch p: $v \subset I(p) \neq \emptyset$.
- *boolean* $p.int$(*segment* s): it is a test of intersection between the interior of a patch p and the interior of a segment s: $dim(I(s) \cap I(p)) = 0$. If the intersection has dimension 1, it returns false.
- *boolean* $p.int$(*patch* p_0): it is a test of intersection between the interior of two patches: $dim(I(p) \cap I(p0)) = 1$. If the intersection has dimension 2, it returns false. $\square$

In order to discuss the robustness of topological relation evaluations in the discrete vector model it is necessary to express the tests required by each cell of of the 9-intersection matrix (see Def. 1) in terms of the basic predicates introduced above. Since in many cases the same expressions have to be reused, the common repeated expressions are introduced hereby as derived operations and derived predicates.

Definition 6 (Derived operations on vector types)**.** The following derived vector operations are defined on the introduced discrete vector model:

Operations and predicates on vertices

Semantics of operations and predicates is shown in Table 1. 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 subsection 5), since also the belonging to relation, the test of not empty intersection between two sets of vertices and the intersection can be derived from it.

- *boolean* $v.bel(set(vertex) V)$: it tests if v belongs to $V = \{v_1, \dots, v_n\}$.
- *set(vertex)* $\{v_{1,1}, \dots, v_{1,n}\} \cap \{v_{2,1}, \dots, v_{2,m}\}$: it returns the common vertexes between $V_1 = \{v_{1,1}, \dots, v_{1,n}\}$ and $V_2 = \{v_{2,1}, \dots, v_{2,m}\}$.
- *boolean* $\{v_{1,1}, \dots, v_{1,n}\} \cap \{v_{1,2}, \dots, v_{2,m}\} \neq \emptyset$: it returns *true* if at least one vertex exists that belongs to both sets $\{v_{1,1}, \dots, v_{1,n}\}$ and $\{v_{1,2}, \dots, v_{2,m}\}$.

Table 1: Expressions for derived operations and predicates regarding vertices. Letters v, v_1, v_2 denotes vertices, while V, V_1 and V_2 denotes set of vertices.

Signature	**Derivation expression**	**Dep.**
boolean $v.bel(V)$	$v.eq(v_1) \vee \dots \vee v.eq(v_n)$	$v.eq(v_0)$
set$(vertex)$ $V_1 \cap V_2$	$\{v \mid \exists v_1 \in V_1 : v.eq(v_1) \wedge \exists v_2 \in V_2 : v.eq(v_2)\}$	$v.eq(v_0)$
boolean $V_1 \cap V_2 \neq \emptyset$	$\exists v_1 \in V_1(\exists v_2 \in V_2(v_1.eq(v_2)))$	$v.eq(v_0)$

Operations and predicates on segments

Semantics of operations and predicates is shown in Table 2. In the sequel the list of operations and predicates is presented (s, s_i represent a segment). 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 subsection 5), since also the equality relation and the topological relations: in, overlaps and disjoint between two segments can be derived from them.

- *set(vertex) s.bnd*(): it returns the boundary of the segments.
- *boolean* s_1.*eq*(*segment* s_2): it tests the equality between two discrete segments.
- *boolean s.bel*(*set*(*segments*) S): it tests if s belongs to $S = \{s_1, \ldots, s_n\}$.
- *boolean* s_1.*in*(*segment* s_2): it is a test of inclusion between two segments $s_1 \subset s_2$.
- *boolean* s_1.*ov*(*segments*$_2$): it is a test of intersection between two segments that requires they share a portion of line, but excludes inclusion or equality (i.e. $dim(I(s_1) \cap I(s_2) = 1)$.
- *boolean* s_1.*dj*(*segment* s_2): it is a test of interior disjointness between two segments $(I(s_1) \cap I(s_2) = \emptyset)$.
- *set(segments) s.diff*(*set*(*segments*)S): it 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. 3 and Tab. 4. 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:

- *set*(*vertex*) *p.ver*(): it returns the patch vertexes.
- *boolean* $p.cnt_{int}$(*segment s*): it is a test of inclusion between a segment and a patch interior $(I(s) \subset I(p))$.
- *boolean p.cnt*(*segment s*): it is a test of inclusion between a segment and a patch $(s \subset p)$.
- *boolean p.ov*(*segments*): it is a test of overlapping between a segment interior and a patch interior $(dim(I(s) \cap I(p)) = 1 \wedge \neg(I(s) \subset I(p))$.
- *boolean p.dj*(*segments*): it is a test of disjointness between a segment interior and a patch interior $(I(s) \cap I(p) = \emptyset)$.
- *boolean* p_1.*eq*(*patch* p_2): it is a test of equality between p_1 and p_2.

Table 2: Expressions for derived operations and predicates regarding segments. Letter s denotes a segment, while S is a set of segments

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

- *boolean* $p_1.\mathit{int}_2$(*patch* p_2): it is a test of interior intersection of dimension 2 between two patches interior $(\mathit{dim}(I(p_1) \cap I(p_2)) = 2)$.

- *boolean* $p_1.\mathit{in}$(*patch* p_2): it is a test of inclusion between two patches $(p_1 \subset p_2)$.

- *boolean* $p_1.\mathit{dj}$(*patch* p_2): it is a test of interior disjointness between two patches $(I(p1) \cap I(p2) = \emptyset)$.

- *boolean* $p_1.\mathit{ov}(p_2)$: it is a test of interior overlapping between two patches $(\mathit{dim}(I(p_1) \cap I(p_2)) = 2 \ \wedge\ \neg(I(p_1) \subset I(p_2)) \ \wedge\ \neg(I(p_2) \subset I(p_1)))$. □

Table 3: Expressions for derived operations and predicates regarding patches. Letters p, p_i denote patches, s, s_i denote segments, v, v_i are vertices.

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 $p_1.eq(p_2)$	$\forall s_1 \in p_1.bnd()(\exists s_2 \in p_2.bnd()(s_1.eq(s_2))) \wedge$ $\forall s_2 \in p_2.bnd()(\exists s_1 \in p_1.bnd()(s_2.eq(s_1)))$	$p.bnd(), v.eq(v_0)$

Table 4: Expressions for derived operations and predicates regarding patches. Letters p, p_i denote patches, s, s_i denote segments, v, v_i are vertices. (Cont.)

Signature	**Derivation expression**	**Dependency**
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 $p.cnt(s)$

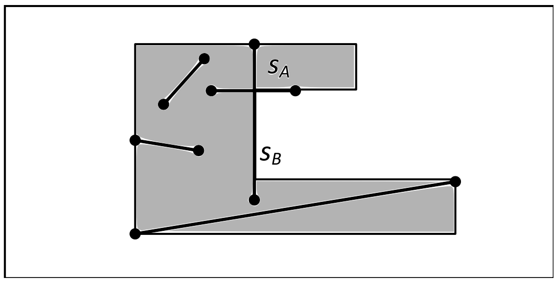


Figure 4: Examples of possible cases that make true the predicate $p.cnt(s)$. Notice that segments s_A and s_B do not satisfy the predicate the $p.cnt_{int}()$, while all the other segments do.

In order to simplify as much as possible the specification of the evaluation tests for topological relations, which are illustrated in the next subsection we introduce some derived operations and predicates that apply to geometries of types: *Linestring*, *Polygon* and *Polyhedral Surface*. These operations are presented in Table 5.

5 Testing Topological Relation in the Discrete Vector Model

This section shows how each cell of the 9-intersection matrix, that describes the relation among the discrete representation $DR(g_1)$, $DR(g_2)$ of two geometries g_1, g_2, can be computed by evaluating a given logical expression

Table 5: Expressions for derived operations and predicates regarding linestrings (*line*), polygons (*poly*) and polyhedral surfaces (*psur*). Moreover, $DR(line) = ln = \{s_1, ..., s_k\}$ $(k > 0)$, $DR(poly) = pg = \{ring_1, ..., ring_l\} = \{pat_1, ..., pat_l\}$ $(l > 1)$ and $DR(psur) = ps = \{p_1, ..., p_m\}$ $(m > 0)$.

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 $\emptyset$
$set(vertex)\ ln.intVer()$	$\{s_i.start() \mid s_i \in ln\} \setminus ln.bnd()$
$boolean\ ln_1.eq(ln_2)$	$\forall s_1 \in ln_1(\exists s_2 \in ln_2(s_1.eq(s_2))) \wedge \forall s_2 \in ln_2(\exists s_1 \in ln_1(s_2.eq(s_1)))$
$linestring\ pg.extBnd()$	$ring_1$
$linestring\ pg.intBnd()$	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
$set(patch)\ pg.intPat()$	$\{pat_2, \ldots, pat_l\}$
$boolean\ pg_1.eq(pg_2)$	$pg_1.extBnd().eq(pg_2.extBnd()) \wedge \forall ln_1 \in pg_1.intBnd()(\exists ln_2 \in pg_2.intBnd()(ln_1.eq(ln2))) \wedge \forall ln_2 \in pg_2.intBnd()(\exists ln_1 \in pg_1.intBnd()(ln_1.eq(ln_2)))$
$set(segment)\ ps.bnd()$	$\{s \mid \exists! p \in ps(s.bel(p.bnd()))\}$
$set(segment)\ ps.intSeg()$	$\{s \mid \exists p \in ps(s.bel(p.bnd())\ \wedge\ \neg s.bel(ps.bnd()))\}$
$set(segment)\ ps.intVer()$	$\{v \mid \exists p \in ps(\exists s \in p.bnd()(v.bel(s.bnd()))) \wedge \neg\exists s' \in ps.bnd()(v.bel(s'.bnd()))\}$
$boolean\ ps_1.in(ps_2)$	$\forall p_i \in ps_1(\exists p_j \in ps_2(p_i.eq(p_j))$
$boolean\ ps_1.eq(ps_2)$	$ps_1.in(ps_2)\ \wedge\ ps_2.in(ps_1)$

containing some of the vector predicates previously presented and having $DR(g_1)$, $DR(g_2)$ as parameters. This is obtained by considering for each matrix 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 which is defined by means of a 9-intersection matrix.

This approach is similar to the one applied in [15], but is extended to the 3D object of the SFA specification. In the following propositions, this notation is used: *pt*, *ln*, *pg* and *ps* represent the discrete representation in the vector model of a point, a linestring, a polygon and a polyhedral surface, respectively. While v, s and p are a vertex, a segment and a patch of the vector model, respectively. Moreover, since every test on the matrix diagonal is symmetric, as the intersection test is symmetric, only one versus is presented for the cells on the diagonal.

Proposition 1 (cell(1,1): Interior-Interior intersection)**.** Given two geometries a and b, the following table shows the cases that might occur in the evaluation of the predicate $dim(I(a) \cap I(b))$, i.e. the cell $(1,1)$ of the 9-intersection matrix, according to the possible geometric types of a and b ($type(a)/type(b)$).

	pt	*ln*	*pg*	*ps*
pt	**C.1.1** see Table 6	**C.1.2** see Table 6	**C.1.3** see Table 6	**C.1.4** see Table 6
ln	**C.1.5** same as C.1.2	**C.1.6** see Table 15	**C.1.7** (only in 2D space) $dim = 0 \Rightarrow$ always F other cases in Table 16	**C.1.8** see Table 7
pg	**C.1.9** same as C.1.3	**C.1.10** same as C.1.7	**C.1.11** (only in 2D space) see Table 9	**C.1.12** NA
ps	**C.1.13** same as C.1.4	**C.1.14** same as C.1.8	**C.1.15** NA	**C.1.16** (only in 3D space) see Table 8

Proof. **C.1.12** and **C.1.15** are not applicable (NA), since as mentioned in Sec. 3, polygon geometries can be embedded only in 2D space while polyhedral surfaces only in 3D space, and topological relations can be evaluated only inside the same space. For all the other cases, Tables 6, 16, 15, 7, 9, 8 and 10 in Appendix 6 shows for each pair of geometric types a sequence of scenes that represent the possible cases of Interior-Interior intersection. The completeness of the set of considered scenes is discussed hereby for each case.

C.1.1 – dim=0/T: two points can only intersect each other if they are equal. No other cases exist.

C.1.2 – dim=0/T: a point *pt* can intersect the interior of a linestring *ln* if there exists a segment s_i of *ln* that contains *pt* in its interior or if *pt* belongs to the set of internal vertices of *ln*. No other cases exist.

C.1.3 – dim=0/T: a point *pt* can intersect the interior of a polygon *pg* if it is contained in the interior of its external patch and it does not intersect any internal patch of *pg*. No other cases exist.

C.1.4 – dim=0/T: a point *pt* can intersect the interior of a polyhedral surface *ps* if it is contained in the interior of one of its patches or it is contained in one of its internal segments or it is equal to one of its internal vertices. No other cases exist.

C.1.6 – dim=0: given two linestrings ln_1 and ln_2, represented as sets of segments, they can produce an interior-interior intersection with dimension 0 if their pointset intersection is composed of one or more isolated points. This can be obtained starting from the intersection of the constituent segments, called s_i and s_j respectively, in the following ways: (i) s_i and s_j share one endpoint which is not part of the ln_1 boundary or ln_2 boundary (i.e. ln_1

and ln_2 have at least one internal vertex in common); (ii) s_i and s_j intersect each other in one point of their interiors; (iii) finally, one endpoint of s_i which is not part of the ln_1 boundary belongs to s_j interior or vice versa; (iv) the other possible scenes, in which the predicate is false, are s_i disjoint s_j, i.e. no intersection, or s_i overlap/in/contains/equal s_j, i.e. intersection of dimension 1.

C.1.6 – dim=1: two linestring ln_1 and ln_2 can produce an interior-interior intersection with dimension 1 if their pointset intersection is composed of one or more segments. This can be obtained starting from the intersection of the component segments, called s_i and s_j respectively, when s_i overlap/in/contains/ equal s_j.

C.1.6 – dim=T: it is obtained by combining the previous two cases.

C.1.7 – dim=1/T: a linestrings *ln* and a polygon *pg* can produce an interior-interior intersection with dimension 1 in the following cases: (i) at least one component segment of *ln* has one end point intersecting the external patch of *pg* and not intersecting any internal patches of *pg*; (ii) at least one component segment of *ln* crosses one segment of the *pg* boundary; (iii) at least one component segment of *ln* in contained in the external patch of *pg* but not in any of its internal patches.

C.1.8 – dim=0: a linestring *ln* and a polyhedral surface *ps* can produce an interior-interior intersection with dimension 0 in the following cases: (i) at least one component segment of *ln* has an intersection of dimension 0 with a patch of *ps*, or with an internal segment of *ps*, or with an internal vertex of *ps*; (ii) at least one internal vertex of *ln* intersects a patch of *ps*, or an internal segment of *ps*, or an internal vertex of *ps*; additionally, in both cases no component segment of *ln* is coplanar and has an intersection with any patch of *ps*.

C.1.8 – dim=1: a linestring *ln* and a polyhedral surface *ps* can produce an interior-interior intersection with dimension 1 in the following cases: (i) at least one component segment of *ln* is coplanar and has an interserction with a patch of *ps*; (ii) at least one component segment of *ln* is equal to/contained or contains/overlaps an internal segment of *ps*.

C.1.8 – dim=T: a linestring *ln* and a polyhedral surface *ps* can produce an interior-interior intersection in the cases of the two previous paragraphs.

C.1.11 – dim=2/T: two polygons pg_1, pg_2 can produce an interior-interior intersection in the following cases: (i) they have the same external path; (ii) there exists a segment s of the boundary of pg_1 that intersects the external patch of pg_2 and there is no internal patch of pg_2 that contains s, or vice versa.

C.1.16 – dim=0: two polyhedral surfaces ps_1, ps_2 can produce an interior-interior intersection of dimension 0 in the following cases: (i) ps_1 and ps_2 share a common internal vertex, (ii) an internal vertex of ps_1 intersects a patch, an internal segment or an internal vertex of ps_2 (or vice versa); (iii) the interior of an internal segment of ps_1 has an intersection of dimension 0 with the interior of an internal segment of ps_2; in all cases an additional condition is required, i.e. any patch of ps_1 is interior disjoint from any patch of ps_2.

C.1.16 – dim=1: two polyhedral surfaces ps_1, ps_2 can produce an interior-interior intersection of dimension 1 in the following cases: (i) at least one patch of ps_1 has an intersection of dimension 1 with at least a patch of ps_2; (ii) at least one patch of ps_1 has an intersection of dimension 1 with at least one internal segment of ps_2; (iii) at least one internal segment of ps_1 is equals to/contains/is contained/overlaps with at least one internal segment of ps_1; in all cases an additional condition is required, i.e. no patch of ps_1 has an intersection of dimension 2 (coplanarity) with a patch of ps_2.

C.1.16 – dim=2: two polyhedral surfaces ps_1, ps_2 can produce an interior-interior intersection of dimension 2, i.e. at least one patch of ps_1 equals to/contains/is contained/ovelaps with at least one patch of ps_2.

C.1.16 – dim=T: two polyhedral surfaces ps_1, ps_2 can produce an interior-interior intersection if one of the cases described in the previous three points occurs, without the specified additional conditions. □

Proposition 2 (cell(1,2): Interior-Boundary intersection)**.** The following table shows the possible cases that might occur in the evaluation of the predicate $dim(I(a) \cap B(b))$, i.e. the cell $(1, 2)$ of the 9-intersection matrix, according to the possible geometric types of a and b ($type(a)/type(b)$).

	pt	*ln*	*pg*	*ps*
pt	**C.2.1** always F	**C.2.2** see Table 11	**C.2.3** same as C.1.2	**C.2.4** same as C.1.2
ln	**C.2.5** always F	**C.2.6** see Table 11	**C.2.7** (only 2D space) same as C.1.6	**C.2.8** (only 3D space) same as C.1.6
pg	**C.2.9** always F	**C.2.10** (only 2D space) same as C.1.3	**C.2.11** (only 2D space) same as C.1.7	**C.2.12** NA
ps	**C.2.13** always F	**C.2.14** (only 3D space) same as C.1.4	**C.2.15** NA	**C.2.16** (only 3D space) same as C.1.7

Proof. The following cases can be reduced to situations in the previous proposition. In case **C.2.3**, *pg.bnd*() is a set of 2D rings: the test C.1.2 has to be applied to each pair (pt, r_i), s.t. $r_i \in pg.bnd()$, and at least one pair must be true. In **C.2.4** *ps.bnd*() is a set of 3D rings: the test C.1.2 has to be applied to each pair (pt, r_i), s.t. $r_i \in ps.bnd()$, and at least one pair must be true. In **C.2.7** *pg.bnd*() is a set of 2D rings: the test C.1.6 has to be applied to each pair (ln, r_i), s.t. $r_i \in pg.bnd()$, and at least for one pair it must be true. In **C.2.8** *ps.bnd*() is a set of 3D rings: the test C.1.6 has to be applied to each pair (ln, r_i), s.t. $r_i \in ps.bnd()$, at least for one pair it must be true. In **C.2.10** $ln.bnd()$ is a set of points: the test C.1.3 has to be applied to each pair (v_i, ln), s.t. $v_i \in ln.bnd()$: at least for one pair it must be true. In **C.2.11** $pg_2.bnd()$ is a set of 2D rings: the test C.1.7 has to be applied to each pair (r_i, pg_1), s.t. $r_i \in pg_2.bnd()$, and at least for one pair it must be true. In **C.2.14** $ln.bnd()$ is a set of points: the test C.1.4 has to be applied to each pair (v_i, ps), s.t. $v_i \in ln.bnd()$, and at least for one pair it must be true. In **C.2.16**: $ps_2.bnd()$ is a set of 3D rings: the test C.1.7 has to be applied to each pair (r_i, ps_1), s.t. $r_i \in ps_2.bnd()$, and at least for one pair it must be true.

C.2.1, **C.2.5**, **C.2.9**, **C.2.13**: are always false, since *pt* has empty boundary, i.e. $B(b) = \emptyset$ if $type(b) = pt$.

C.2.12 and **C.2.15** are not applicable (NA), because polygons and polyhedral surfaces are embedded in two different spaces.

The only cases that remain to be discussed are **C.2.2** and **C.2.6**. Tab. 11 in Sec. 6 shows a sequence of scenes that represent the possible cases of Interior-Boundary intersection among points and linestrings. The fact that the set of considered scenes is complete is discussed hereby for each case.

C.2.2 - dim=0/T: a point *pt* intersects the boundary of a linestring *ln* if there exists an endpoint of *ln* that is equal to *pt*. No other cases exist.

C.2.6 - dim=0/T: the interior of a linestring ln_1 intersects the boundary of another linestring ln_2 in the following cases: (i) there exists an endpoint of ln_2 that intersects an internal vertex of ln_1; (ii) there exists an endpoint of ln_2 that intersects the interior of a component segment of ln_1. No other cases exist. □

Proposition 3 (cell(1,3): Interior-Exterior intersection)**.** The following table shows the cases that might occur in the evaluation of the predicate $dim(I(a) \cap E(b))$, i.e. the cell $(1, 3)$ of the 9-intersection matrix, according to the possible geometric types of a and b ($type(a)/type(b)$).

	pt	*ln*	*pg*	*ps*
pt	**C.3.1** equivalent to ¬ C.1.1	**C.3.2** see Table 12	**C.3.3** see Table 12	**C.3.4** see Table 12
ln	**C.3.5** always T	**C.3.6** see Table 13	**C.3.7** (only 2D space) see Table 13	**C.3.8** (only 3D space) see Table 13
pg	**C.3.9** always T	**C.3.10** always T	**C.3.11** (only 2D space) see Table 14	**C.3.12** NA
ps	**C.3.13** always T	**C.3.14** always T	**C.3.15** NA	**C.3.16** (only 3D space) see Table 14

□

Proof. **C.3.1** is equivalent to $\neg C.1.1$ since the exterior of a point intersects another point if they are disjoint, i.e. they are not equal. **C.3.5**, **C.3.9** and **C.3.13** are always true: the exterior of a point *pt* always intersects any given line, polygon or polyhedral surface respectively, since a line/polygon/surface cannot be contained in *pt*. **C.3.10** and **C.3.14** are always true: the exterior of a line *ln* always intersects any given polygon or polyhedral surface respectively, since a polygon/surface cannot be contained in *ln*. **C.3.12** and **C.3.15** are not applicable (NA), since polygons and polyhedral surfaces are embedded in two different spaces. Tables 12, 13 and 14 in Appendix 6 show a sequence of scenes that represent the possible cases of Interior-Exterior intersection in the remaining cases. The completeness of the set of considered scenes is discussed hereby for each case.

C.3.2 - dim=0/T: a point *pt* intersects the exterior of a linestring *ln*, if it is not poinset contained in *ln*, which can be tested by applying the following condition: *pt* is not equal to the endpoints of *ln* and it is not equal to one of the internal vertices of *ln* and it is not contained in the interior of one of its component segments.

C.3.3 - dim=0/T: a point *pt* intersects the exterior of a polygon *pg*, if it is not poinset contained in *pg*, which can be tested by applying the following condition: *pt* is not contained in the interior of the external patch of *pg* or it is contained in the interior of one internal patch of *pg* and it is neither contained in the interior of one component segment s of *pg* boundary nor it is equal to one of the endpoints of s.

C.3.4 - dim=0/T: a point *pt* intersects the exterior of a polyhedral surface *ps*, if it is not poinset contained in *ps*, which can be tested by applying the following condition: *pt* is not contained in the interior of any patch of *ps* and it is not contained in the interior of one component segment s of the boundary of any patch of *ps* and it is not equal to one of the endpoints of s.

C.3.6 - dim=1/T: a linestring ln_1 intersects the exterior of another linestring ln_2, if it is not poinset contained in ln_2, which can be tested by applying the following condition: there exists at least one component segment s of ln_1 which overlaps or contains one component segment of ln_2 or s is disjoint from it.

C.3.7 - dim=1/T: a linestring *ln* intersects the exterior of a polygon *pg*, if it is not poinset contained in *pg*, which can be tested by applying the following condition: either there exists at least one component segment s of *ln* which is not contained in the external patch of *pg* or that is contained in one internal patch of *pg* or that crosses at least one component segment of the boundary of *pg*.

C.3.8 - dim=1/T: a linestring *ln* intersects the exterior of a polyhedral surface *ps*, if it is not poinset contained in *ps*, which can be tested by applying the following condition: there exists at least one component segment s of *ln* which is overlap/intersects/is disjoint from all patches of *ps* and that is not contained or equal to any internal segment of *ps*.

C.3.11 - dim=2/T: a polygon pg_1 intersects the exterior of another polygon pg_2, if it is not poinset contained in pg_2, which can be tested by applying the following condition: there exists at least one component segment s of the boundary of pg_1 which is not contained in the external patch of pg_2 or that is contained in the external patch of pg_2 but it crosses/is contained in at least one internal patch of pg_2.

C.3.16 - dim=2/T: a polyhedral surface ps_1 intersects the exterior of another polyhedral surface ps_2, if it is not poinset contained in ps_2, which can be tested by applying the following condition: there exists at least one patch p_1 of ps_1 such that for all patches p_2 of ps_2, p_1 overlaps/intersects/is disjoint from p_2.

□

Proposition 4 (cell(2,2): Boundary-Boundary intersection)**.** The following table shows all possibile cases in the evaluation of the predicate $dim(B(a) \cap B(b))$, according to the geometric types of a and b ($type(a)/type(b)$).

	pt	*ln*	*pg*	*ps*
pt	**C.4.1** always F	**C.4.2** always F	**C.4.3** always F	**C.4.4** always F
ln	**C.4.5** always F	**C.4.6** same as C.1.1	**C.4.7** (only 2D space) same as C.1.2	**C.4.8** (only 3D space) same as C.1.2
pg	**C.4.9** always F	**C.4.10** same as C.4.7	**C.4.11** (only 2D space) same as C.1.6	**C.4.12** NA
ps	**C.4.13** always F	**C.4.14** same as C.4.8	**C.4.15** NA	**C.4.16** (only 3D space) same as C.1.6

□

Proof. The following cases can be reduced to situations in the previous propositions. In **C.4.6** $ln_1.bnd()$ ($ln_2.bnd()$) is a set of points: the test C.1.1 has to be applied to each pair (v_1, v_2), s.t. $v_1 \in ln_1.bnd()$ and $v_2 \in ln_2.bnd()$, and at least for one pair it must be true. In **C.4.7** $ln.bnd()$ is a set of points while $pg.bnd()$ is a set of 2D ring: the test C.1.2 has to be applied to each pair (v, r_i), s.t. $v \in ln.bnd()$ and $r_i \in pg.bnd()$, and at least one pair must be true. In **C.4.11** $pg_1.bnd()$ ($pg_2.bnd()$) is a set of 2D ring: the test C.1.6 has to be applied to each pair (r_1, r_2), s.t. $r_1 \in pg_1.bnd()$ and $r_2 \in pg_2.bnd()$, and at least for one pair it must be true. Finally, in **C.4.16**, $ps_1.bnd()$ ($ps_2.bnd()$) is a set of 3D ring: the test C.1.6 has to be applied to each pair (r_1, r_2), s.t. $r_1 \in ps_1.bnd()$ and $r_2 \in ps_2.bnd()$, and at least for one pair it must be true.

Cases **C.4.12** and **C.4.15** are not applicable (NA), since polygons and polyhedral surfaces are embedded in two different spaces. There is no new cases to prove with respect to previous propositions. □

Proposition 5 (cell(2,3): Boundary-Exterior intersection)**.** The following table shows the possible cases in the evaluation of the predicate $dim(B(a) \cap E(b))$, according to the possible geometric types of a and b ($type(a)/type(b)$)

	pt	*ln*	*pg*	*ps*
pt	**C.5.1** always F	**C.5.2** always F	**C.5.3** always F	**C.5.4** always F
ln	**C.5.5** if $ln.bnd() \neq \emptyset$ then it is T, otherwise it is F	**C.5.6** if $ln.bnd() = \emptyset$ then it is F, otherwise same as C.3.2	**C.5.7** (only 2D space) if $ln.bnd() = \emptyset$ then it is F, otherwise same as C.3.3	**C.5.8** (only 3D space) if $ln.bnd() = \emptyset$ then it is F, otherwise same as C.3.4
pg	**C.5.9** always T	**C.5.10** same as C.3.6	**C.5.11** (only 2D space) same as C.3.7	**C.5.12** NA
ps	**C.5.13** always T	**C.5.14** same as C.3.6	**C.5.15** NA	**C.5.16** (only 3D space) same as C.3.8

Proof. The following cases can be reduced to situation in the previous propositions. In **C.5.6** $ln_1.bnd()$ is a set of points: the test C.3.2 has to be applied to each pair (v, ln_2), s.t. $v \in ln_1.bnd()$, and at least for one pair it must be true. In **C.5.7** $ln.bnd()$ is a set of points: the test C.3.3 has to be applied to each pair (v, pg), s.t. $v \in ln.bnd()$, and at least for one pair it must be true. In **C.5.8** $ln.bnd()$ is a set of points: the test C.3.4 has to be applied to each pair (v, ps), s.t. $v \in ln.bnd()$, and at least for one pair it must be true. In **C.5.10** $pg.bnd()$ is a set of 2D rings: the test C.3.6 has to be applied to each pair (r, ln), s.t. $r \in pg.bnd()$, and at least for one pair it must be true. In **C.5.11** $pg_1.bnd()$ is a set of 2D ring: the test C.3.7 has to be applied to each pair (r, pg_2), s.t. $r \in pg_1.bnd()$, and at least for one pair it must be true. In **C.5.14** $ps.bnd()$ is a set of 3D rings: the test C.3.6 has to be applied to each pair (r, ln), s.t. $r \in ps.bnd()$, and at least for one pair it must be true. Finally, in **C.5.16**, $ps_1.bnd()$ is a set of 3D ring: the test C.3.8 has to be applied to each pair (r, ps_2), s.t. $r \in ps_1.bnd()$, and at least for one pair it must be true.

Cases **C.5.12** and **C.5.15** are not applicable (NA), since polygons and polyhedral surfaces are embedded in two different spaces. Cases **C.5.1**, **C.5.2**, **C.5.3** and **C.5.4** are always false since the boundary of a point is always empty. Case **C.5.5** is true if $ln.bnd() \neq \emptyset$, since the boundary of a linestring is composed of two points and at least one always intersects the exterior of another point, otherwise, it is false. Case **C.5.9** (**C.5.13**) is true since the boundary of a polygon (polyhedral surface) is a set of 2D (3D) rings and they always intersect the exterior of a point. There is no new cases to prove with respect to previous propositions. □

The reference set of topological relations considered in this paper can be implemented using the formulas presented in the previous propositions. In order to obtain robust tests for their evaluation, the following section formalizes some assumptions about the distributed environment in which the data exchange is performed, and on their basis defines some rules that the vector representation has to satisfy for guaranteeing robustness.

6 Proof Tables

This section reports the proof tables for Prop.-5 in Sec. 5. 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 6: Proof *Interior-Interior Intersection* $(pt/*)$

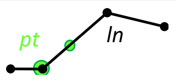
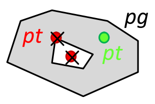
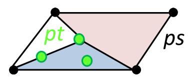
Case	**Testing conditions**	**Scene**
(pt_1/pt_2) $dim{=}0/\mathrm{T}$	$pt_1.eq(pt_2)$	
(pt/ln) $dim{=}0/\mathrm{T}$	$\exists s \in ln(s.cnt(pt)) \vee\ pt.bel(ln.intVer())$	
(pt/pg) $dim{=}0/\mathrm{T}$	$pg.extPat().cnt(pt) \wedge$ $\neg\exists p \in pt.intPat()(p.cnt(pt)$ $\vee\ pt.bel(p.ver())$ $\vee\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

Table 7: Proof *Interior-Interior Intersection* (*ln/ps*) – (only in 3D space)

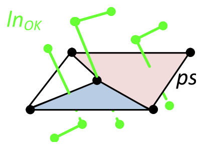
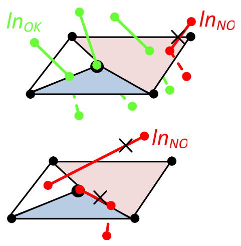
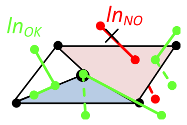
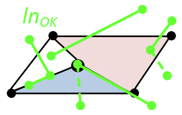
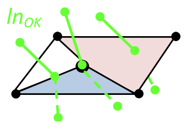
Case	**Necessary conditions**	**Additional conditions**	**Scene**
ln/ps $dim = 0$	$\exists s \in ln(\exists p \in ps($ $p.int(s))\vee$ $\exists s_1 \in ps.intSeg()($ $s.int(s_1))\vee$ $\exists v_1 \in ps.intVer()($ $s.cnt(v_1)))$	$\forall s \in ln(\forall p \in ps$ $(p.int(s) \vee p.dj(s)))\wedge$ $\forall s \in ln(\forall s_1 \in ps.intSeg()$ $(s.dj(s_1) \vee s.int(s_1)))$	
	$\exists v \in ln.intVer()($ $\exists p \in ps(p.cnt(v))\vee$ $\exists s_1 \in ps.intSeg()($ $s_1.cnt(v))\vee$ $\exists v_1 \in ps.intVer()($ $v.eq(v_1)))$	$\forall s \in ln(\forall p \in ps$ $(p.int(s) \vee p.dj(s)))\wedge$ $\forall s \in ln(\forall s_1 \in ps.intSeg()$ $(s.dj(s_1) \vee s.int(s_1)))\wedge$	
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 8: Proof *Interior-Interior intersection* (ps_1/ps_2) dim=0/1/2 – (only in 3D space).

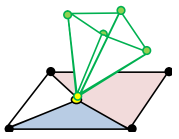
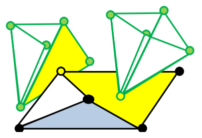
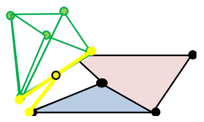
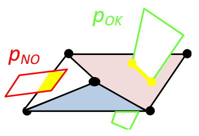
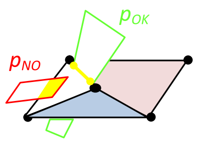
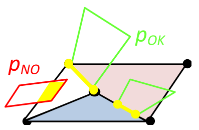
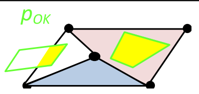
Case	**Necessary conditions**	**Additional conditions**	**Scene**
ps_1/ps_2 $dim{=}0$	$\exists v \in ps_1.intVer()(v.bel(ps_2.intVer()))$	$\forall p_1 \in ps_1(\forall p_2 \in ps_2(p_1.dj(p_2)))$	
	$\exists v \in ps_1.intVer()(\exists p \in ps_2(p.cnt(v)\vee \exists s \in p.bnd()(s.cnt(v))))\vee \exists v \in ps_2.intVer()(\exists p \in ps_1(p.cnt(v)\vee \exists s \in p.bnd()(s.cnt(v))))$	$\forall p_1 \in ps_1(\forall p_2 \in ps_2(p_1.dj(p_2)))$	
	$\exists s_1 \in ps_1.intSeg()(\exists s_2 \in ps_2.intSeg()(s_1.int(s_2)))$	$\forall p_1 \in ps_1(\forall p_2 \in ps_2(p_1.dj(p_2)))$	
ps_1/ps_2 $dim{=}1$	$\exists p_1 \in ps_1(\exists p_2 \in ps_2(p_1.int(p_2)))$	$\forall p_1 \in ps_1(\forall p_2 \in ps_2(p_1.dj(p_2) \vee p_1.int(p_2)$	
	$\exists s \in ps_1.intSeg()(\exists p \in ps_2(p.cnt(s)\vee p.ov(s)))\vee \exists s \in ps_2.intSeg()(\exists p \in ps_1(p.cnt(s)\vee p.ov(s)))$	$\forall p_1 \in ps_1(\forall p_2 \in ps_2(p_1.dj(p_2) \vee p_1.int(p_2)$	
	$\exists s_1 \in ps_1.intSeg()(\exists s_2 \in ps_2.intSeg()(s_1.ov(s_2) \vee s_1.in(s_2)\vee s_1.eq(s_2)))$	$\forall p_1 \in ps_1(\forall p_2 \in ps_2(p_1.dj(p_2) \vee p_1.int(p_2)$	
ps_1/ps_2 $dim{=}2$	$\exists p_1 \in ps_1(\exists p_2 \in ps_2(p_1.eq(p_2) \vee p_1.int_2(p_2)))$		

Table 9: Proof *Interior-Interior Intersection* (pg_1/pg_2) – (only in 2D space)

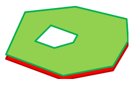
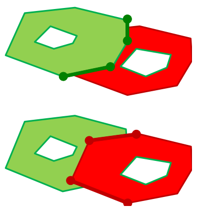
Case	Testing conditions	Scene
pg_1/pg_2 dim=2/T	$pg_1.extPat().eq(pg_2.exPat())$	
	$\exists s \in pg_1.bnd()((pg_2.extPat().ov(s) \vee$ $pg_2.extPat().cnt(s)) \wedge$ $\neg \exists p \in pg_2.intPat()(p.cnt(s))) \vee$ $\exists s \in pg_2.bnd()((pg_1.extPat().ov(s) \vee$ $pg_1.extPat().cnt(s)) \wedge$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

Table 10: 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 8
	$\exists s \in ps_1.intSeg()(\exists p \in ps_2($ $p.ov(s) \vee p.cnt(s))) \vee$ $\exists s \in ps_2.intSeg()(\exists p \in ps_1($ $p.ov(s) \vee p.cnt(s)))$	see Table 8
	$\exists s_1 \in ps_1.intSeg()(\exists s_2 \in ps_2.intSeg()($ $s_1.ov(s_2) \vee s_1.int(s_2) \vee s_2.in(s_1) \vee s_1.eq(s_2) \vee$ $s_1.int(s_2)))$	see Table 8
	$\exists v \in ps_1.intVer()(\exists p \in ps_2(p.cnt(v) \vee$ $\exists s \in p.bnd()(s.cnt(v)))) \vee$ $\exists v \in ps_2.intVer()(\exists p \in ps_1(p.cnt(v) \vee$ $\exists s \in p.bnd()(s.cnt(v))))$	see Table 8
	$\exists v_1 \in ps_1.intVer()(\exists v_2 \in ps_2.intVer()($ $v_1.eq(v_2)))$	see Table 8

Table 11: Proof *Interior-Boundary intersection* (pt/ln) and (ln/ln)

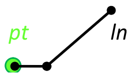
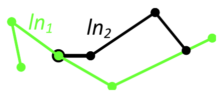
Case	Testing conditions	Scene
pt/ln dim=0/T	$pt.bel(ln.bnd())$	
ln_1/ln_2 dim=0/T	$\exists v \in ln_2.bnd()(v.bel(ln_1.intVer()) \vee$ $\exists s \in ln_1(s.cnt(v)))$	

Table 12: Proof *Interior-Exterior intersection* (*pt*/*ln*), (*pt*/*pg*) and (*pt*/*ps*)

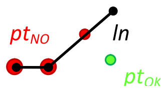
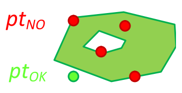
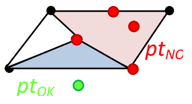
Case	Testing conditions	Scene
pt/*ln* *dim*=0/T	$\forall v \in ln.bnd()(\neg pt.eq(v))\wedge$ $\forall v_i \in ln.intVer()(\neg pt.eq(v_i))$ $\forall s \in ln(\neg s.cnt(v)))$	
pt/*pg* *dim*=0/T	$(\neg pg.extPat().cnt(pt)\wedge$ $\exists p \in pg.intPat()(p.cnt(pt)))\wedge$ $\forall ln \in pg.bnd()(\forall s \in ln(\neg s.cnt(pt)\wedge$ $\forall v \in s.bnd()(\neg pt.eq(v)))$	
pt/*ps* *dim*=0/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 13: Proof *Interior-Exterior intersection* (*ln*/*ln*), (*ln*/*pg*) and (*ln*/*ps*)

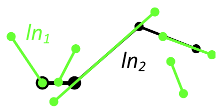
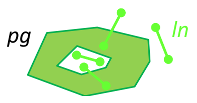
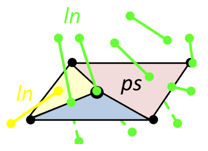
Case	Additional conditions	Scene
ln_1/ln_2 *dim*=1/T	$\exists s_1 \in ln_1(\forall s_2 \in ln_2(s_1.dj(s_2)\vee$ $s_1.int(s_2) \vee s_1.ov(s_2) \vee s_2.in(s_1)))$	
ln/*pg* *dim*=1/T	$\exists s \in ln(\neg pg.extPat().cnt(s))\vee$ $\exists s \in ln(\exists p \in pg.intPat()(p.cnt(s)))\vee$ $\exists ln \in pg.bnd()(\exists s_1 \in ln(s.int(s_1)))$	
ln/*ps* *dim*=1/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 14: Proof *Interior-Exterior intersection* (*pg*/*pg*) and (*ps*/*ps*)– Case C.3.11 and C.3.16.

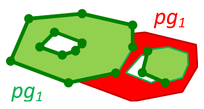
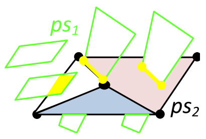
Case	Additional conditions	Scene
pg_1/pg_2 *dim*=2/T	$\exists s \in pg_1.extPat().bnd()($ $(\neg pg_2.extPat().cnt(s)\vee$ $(pg_2.extPat().cnt(s)\wedge$ $\exists p \in pg_2.intPat()(p.ov(s) \vee p.cnt(s)))))$	
ps_1/ps_2 *dim*=2/T	$\exists p_1 \in ps_1(\forall p_2 \in ps_2(p_1.dj(p_2)\vee$ $p_1.ov(p_2)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

Table 15: 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)

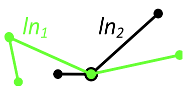
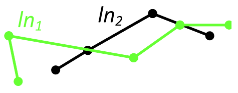
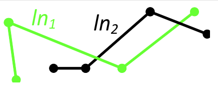
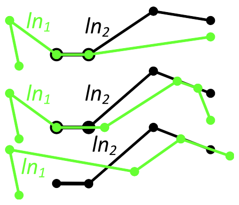
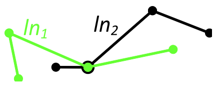
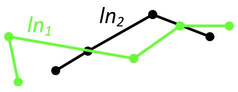
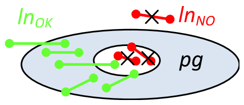
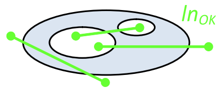
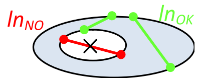
Case	**Necessary conditions**	**Additional conditions**	**Scene**
ln_1/ln_2 $dim{=}0$	$ln_1.intVer() \cap ln_2.intVer() \neq \emptyset$	$\forall s_1 \in ln_1(\forall s_2 \in ln_2(s_1.int(s_2) \vee s_1.dj(s_2)))$	
	$\exists v \in ln_1.intVer()(\exists s \in ln_2(s.cnt(v))) \vee \exists v \in ln_2.intVer()(\exists s \in ln_1(s.cnt(v)))$	$\forall s_1 \in ln_1(\forall s_2 \in ln_2(s_1.int(s_2) \vee s_1.dj(s_2)))$	
	$\exists s_1 \in ln_1(\exists s_2 \in ln_2 (s_1.in(s_2)))$	$\forall s_1 \in ln_1(\forall s_2 \in ln_2(s_1.int(s_2) \vee s_1.dj(s_2)))$	
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 16: Proof *Interior-Interior intersection* (*ln*/*pg*)

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

7 Conclusion and Future Work

Evaluating topological relations in distributed environments with many datasets at hand 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 with 3D types (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 [6] 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 volumes.

References

[1] L. Chen. *Exact Geometric Computation: Theory and Applications*. PhD thesis, New York University, Department of Computer Science, 2001.

[2] Eliseo Clementini and Paolino Di Felice. A Comparison of Methods for Representing Topological Relationships. *Inf. Sci. Appl.*, 3(3):149–178, 1995.

[3] Eliseo Clementini, Paolino Di Felice, and Peter van Oosterom. A Small Set of Formal Topological Relationships Suitable for End-User Interaction. In *Proceedings of the Third International Symposium on Advances in Spatial Databases*, pages 277–295. Springer-Verlag, 1993.

[4] Volker Coors. 3d-gis in networking environments. *Computers, Environment and Urban Systems*, 27(4):345–357, 2003.

[5] M. J. Egenhofer, A. U. Frank, and J. P. Jackson. A topological data model for spatial databases. In *Proceedings of the 1st Symposium on Design and Implementation of Large Spatial Databases (SSD '90)*, pages 271–286, 1990.

[6] M. J. Egenhofer and R. Franzosa. Point-set topological spatial relations. *International Journal of Geographical Information Systems*, 5(2):161–174, 1991.

[7] Ralf Hartmut Güting and Markus Schneider. Realms: A Foundation for Spatial Data Types in Database Systems. In *Int. Symp. on Advances in Spatial Databases*, volume 692, pages 14–35, 1993.

[8] Dan Halperin. Controlled Perturbation for Certified Geometric Computing with Fixed-Precision Arithmetic. In *Proceedings of the Third International Congress Conference on Mathematical Software*, ICMS'10, pages 92–95. Springer-Verlag, 2010.

[9] Dan Halperin and Eli Packer. Iterated snap rounding. *Comput. Geom. Theory Appl.*, 23(2):209–225, 2002.

[10] J. Hobby. Practical segment intersection with finite precision output. *Comp. Geometry Theory and App*, 13:Comp. Geometry Th. and App., 1999.

[11] M. Molenaar. A formal data structure for 3D vector maps. In *Proceedings of EGIS90*, page 770781, 1990.

[12] OGC. *OpenGIS Implementation Standard for Geographic Information – Simple Feature Access – Part 1: Common Architecture*, 2011. version 1.2.1.

[13] Giuseppe Pelagatti, Mauro Negri, Alberto Belussi, and Sara Migliorini. 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*, pages 448–451, New York, NY, USA, 2009. ACM.

[14] M. Pilouk. *Integrated modelling for 3D GIS*. PhD thesis, ITC, The Netherlands, 1996.

[15] Reasey Praing and Markus Schneider. Efficient Implementation Techniques for Topological Predicates on Complex Spatial Objects. *Geoinformatica*, 12(3):313–356, 2008.

[16] David A. Randell, Zhan Cui, and Anthony Cohn. A Spatial Logic Based on Regions and Connection. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR'92)*, pages 165–176. Morgan Kaufmann, 1992.

[17] M. Andrea Rodríguez, Nieves Brisaboa, Jazna Meza, and Miguel R. Luaces. Measuring Consistency with Respect to Topological Dependency Constraints. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '10, pages 182–191, New York, NY, USA, 2010. ACM.

[18] David M. Theobald. Topology Revisited: Representing Spatial Relations. *International Journal of Geographical Information Science*, 15(8):689–705, 2001.

[19] Rodney James Thompson and Peter van Oosterom. Interchange of Spatial Data-Inhibiting Factors. In *Proceeding of the 9th AGILE International Conference on Geographic Information Science*, 2006.

[20] S. Zlatanova. *3D GIS for Urban Development*. PhD thesis, ITC – Faculty of Geo-Information Science and Earth Observation, The Netherlands, 2000.