Geoinformatica (2008) 12:313–356
DOI 10.1007/s10707-007-0035-y

Efficient Implementation Techniques for Topological Predicates on Complex Spatial Objects

Reasey Praing · Markus Schneider

Received: 19 December 2006 / Revised: 10 July 2007 /
Accepted: 14 August 2007 / Published online: 4 October 2007
© Springer Science + Business Media, LLC 2007

Abstract Topological relationships like *overlap*, *inside*, *meet*, and *disjoint* uniquely characterize the relative position between objects in space. For a long time, they have been a focus of interdisciplinary research as in artificial intelligence, cognitive science, linguistics, robotics, and spatial reasoning. Especially as predicates, they support the design of suitable query languages for spatial data retrieval and analysis in spatial database systems and geographical information systems. While, to a large extent, conceptual aspects of topological predicates (like their definition and reasoning with them) as well as strategies for avoiding unnecessary or repetitive predicate executions (like predicate migration and spatial index structures) have been emphasized, the development of robust and efficient implementation techniques for them has been largely neglected. Especially the recent design of topological predicates for all combinations of *complex* spatial data types has resulted in a large increase of their numbers and stressed the importance of their efficient implementation. The goal of this article is to develop correct and efficient implementation techniques of topological predicates for all combinations of complex spatial data types including two-dimensional point, line, and region objects, as they have been specified by different authors and in different commercial and public domain software packages. Our solution consists of two phases. In the *exploration phase*, for a given scene of two spatial objects, all *topological events* like intersection and meeting situations are summarized in two precisely defined *topological feature vectors* (one for each argument object of a topological predicate) whose specifications are characteristic

This work was partially supported by the National Science Foundation (NSF) under grant number NSF-CAREER-IIS-0347574.

R. Praing · M. Schneider (✉)
Department of Computer and Information Science and Engineering,
University of Florida, Gainesville, FL 32611, USA
e-mail: mschneid@cise.ufl.edu

R. Praing
e-mail: rpraing@cise.ufl.edu

and unique for each combination of spatial data types. These vectors serve as input for the *evaluation phase* which analyzes the topological events and determines the Boolean result of a topological predicate (*predicate verification*) or the kind of topological predicate (*predicate determination*) by a formally defined method called *nine-intersection matrix characterization*. Besides this general evaluation method, the article presents an optimized method for predicate verification, called *matrix thinning*, and an optimized method for predicate determination, called *minimum cost decision tree*. The methods presented in this article are applicable to all known complete collections of mutually exclusive topological predicates that are formally based on the well known nine-intersection model.

Keywords spatial database · complex spatial object · topological predicate · predicate verification · predicate determination · topological feature vector · exploration phase · evaluation phase · nine-intersection matrix characterization · matrix thinning · minimum cost decision tree

1 Introduction

Topological predicates (like *overlap*, *meet*, *inside*) between spatial objects (like *points*, *lines*, *regions*) have for a long time been a main area of extensive research on spatial data handling, reasoning, and query languages in a number of disciplines like artificial intelligence, linguistics, robotics, and cognitive science. They characterize the relative position between two (or more) objects in space, are of purely qualitative nature, deliberately exclude any consideration of quantitative, metric measures like distance or direction measures, are associated with notions like adjacency, coincidence, connectivity, inclusion, and continuity, and are preserved under affine transformations such as translation, scaling, and rotation. In particular, they support the design of suitable query languages for spatial data retrieval and analysis in geographical information systems and spatial database systems. The focus of this research has been on the conceptual design of and reasoning with these predicates as well as on strategies for avoiding unnecessary or repetitive predicate executions. The two central conceptual approaches, upon which almost all publications in this field have been based and which have produced very similar results, are the *nine-intersection model* [12] and the *RCC model* [10]. Until recently, topological predicates have only been formally defined for *simple* spatial objects that include single points, continuous lines, and simple regions. In this article, we are especially interested in topological predicates for *complex* spatial objects which comprise point objects containing several single points, line objects consisting of a finite number of curves, and regions consisting of several faces possibly with holes. Complex spatial objects have been proposed in a number of publications and are available in several commercial and public domain software packages. Topological predicates on complex spatial objects have been recently specified in [35] on the basis of the nine-intersection model. In addition, we also consider *dimension-refined topological predicates* [26] which take into account the dimensions of the intersections of two spatial objects and enable the posing of more fine-grained than purely topological queries.

1.1 Motivation

In contrast to the large amount of conceptual work, implementation issues of topological predicates as well as optimization issues for their efficient evaluation have been widely neglected. Since topological predicates are *expensive predicates* that cannot be evaluated in constant time, the strategy in query plans has consequently been to avoid their computation. The extensive work on spatial index structures as a filtering step in query processing is an important example of this strategy. It aims at identifying a hopefully small collection of candidate pairs of spatial objects that could possibly fulfil the predicate of interest and at excluding a large collection of pairs of spatial objects that definitely do not satisfy the predicate.

However, efficient implementation techniques for the topological predicates themselves cannot be found in the literature. Due to the transition from simple to complex spatial objects and the consequential increase of the number of topological predicates between them, novel implementation and optimization techniques turn out to be indispensable. They are challenging for four main reasons.

First, although the plane sweep paradigm [1] is widely accepted as an appropriate implementation concept for topological predicates, the question arises whether each topological predicate requires an own, tailored plane sweep algorithm. This would lead to a large number of algorithms.

A second issue is what kind of information a plane sweep algorithm should output so that this information can be leveraged for predicate evaluation.

Third, the two implementation alternatives of a single, specialized algorithm for each predicate or a single algorithm for all predicates with an exhaustive case analysis are error-prone and thus unacceptable options from a *correctness* point of view. The reason is that each single algorithm finally represents an *ad hoc* implementation from which we do not know whether it really covers all possible cases that make the predicate true. We also do not know whether its result is mutually exclusive to all other predicate results. This means if a predicate implementation yields *true* for a particular spatial configuration, all the other predicate implementations have to return *false*. If it yields *false*, exactly one of the other predicate implementations has to return *true*. Altogether, this is a correctness problem.

A fourth issue deals with the kind of query posed since this has impact on the evaluation process. Given two objects A and B of any complex spatial data type for point, line, or region objects, we can pose at least two kinds of topological queries: (1) "Do A and B satisfy the topological predicate p?" and (2) "What is the topological predicate p between A and B?". Only query 1 yields a Boolean value. It is of interest for the query processing of spatial joins and spatial selections in spatial databases; we call it hence a *verification query*. Query 2 returns a predicate (name), is hence called *determination query*, and is interesting for all applications analyzing the topological relationships of spatial objects. The only alternative would be to transform this query into a large number of verification queries and terminate this process as soon as one of them yields *true*. For both query types, we can even ask more fine-grained queries that include the dimension of an intersection. For example, the topological predicate *meet* between two region objects can be dimensionally refined into 0-*meet*, 1-*meet*, and 01-*meet* respectively. This states that the two region objects meet in a point object (0D), a line object (1D), and a point object and a line object (0D+1D) respectively.

1.2 Goals and overall solution

The overall goal of this article is to develop and present systematic, correct, robust, and efficient implementation strategies and optimized evaluation methods for topological predicates between all combinations of the three complex spatial data types for point, line, and region objects. The strategies and methods are supposed to solve the four main issues mentioned in Section 1.1 and are universal in the sense that they can be leveraged for all available spatial type systems offering simple and/or complex spatial data types and for all complete collections of mutually exclusive topological predicates based on the nine-intersection model.

We distinguish two phases of predicate execution: In an *exploration phase*, a plane sweep scans a given configuration of two spatial objects and identifies all *topological events* like intersections and meeting situations. These events are stored in two precisely defined *topological feature vectors* (one for each argument object of the topological predicate) whose specifications are characteristic and thus unique for each of the six spatial data type combinations point/point, point/line, point/region, line/region, and region/region (the remaining three combinations are symmetric). In other words, different type combinations lead to different topological feature vectors. The exploration phase solves the first and second issue from Section 1.1. The first issue is solved in the sense that for each of the six type combinations *only* a single algorithm is necessary to compute the suitable topological information for *all* topological predicates of that type combination. Topological feature vectors as the only required output are the answer to the second issue.

These vectors serve as input for the *evaluation phase* which analyzes the topological data and determines the Boolean result of a topological predicate (query 1) or the kind of topological predicate (query 2). For this purpose, we introduce a general method called *nine-intersection matrix characterization* that can be leveraged for both predicate verification and predicate determination and that depends on the spatial data type combination under consideration. A slight extension of this method can then be used for the matching of dimension-refined topological predicates. A fine-tuning of the nine-intersection matrix characterization leads to two optimized approaches called *matrix thinning* for predicate verification and *minimum cost decision trees* for predicate determination. The evaluation phase solves the third and second issue from Section 1.1. The third issue is solved by formally proving the correctness of the approach. The fourth issue is answered by designing general and optimized solutions for both predicate determination and predicate determination

Section 2 discusses related work about spatial data types as well as available design and implementation concepts for proper and dimension-refined topological predicates. Section 3 deals with the exploration phase for collecting topological information. By using the plane sweep paradigm, we extract the needed topological information from a given scene of two spatial objects and determine the topological feature vectors that are specific to each type combination. Section 4 focuses on the evaluation phase and matches the acquired topological feature vectors with characteristic properties of the topological predicates by the nine-intersection matrix characterization. Section 5 presents the two fine-tuned and optimized approaches of matrix thinning for predicate verification and minimum cost decision trees for predicate determination. In Section 6, we give a brief overview of the implementation

environment, present our testing strategy, and present an assessment of our overall approach. Finally, Section 7 draws some conclusions.

2 Related work

In this section we present related work on spatial data types as the argument types of topological predicates (Section 2.1), give an overview of the essential conceptual models for topological relationships (Section 2.2) and dimension-refined topological relationships (Section 2.3), and discuss implementation aspects of topological predicates (Section 2.4).

2.1 Spatial data types

In the spatial database and GIS community, *spatial data types* (e.g., [11], [19], [20], [29], [32], [38]) like *point*, *line*, or *region* have found wide acceptance as fundamental abstractions for modeling the structure of geometric entities, their relationships, properties, and operations. They form the basis of a large number of data models and query languages for spatial data and have gained access into commercial software products. Topological predicates operate on spatial objects as instances of these data types. The literature distinguishes *simple* spatial data types (e.g., [11], [19], [29]) and *complex* spatial data types (e.g., [5], [20], [25], [27], [32], [35], [38]), depending on the spatial complexity they are able to model. We use the terms "simple spatial data type" and "complex spatial data type" also as neutral, unifying notations for the various naming schemes of these types in different approaches. Simple spatial data types only provide simple object structures like single points, continuous lines, and simple regions (Fig. 1a–c). Until recently, topological relationships have only been formally defined for them (e.g., [13]). However, from an application perspective, simple spatial data types have turned out to be inadequate abstractions for real spatial applications since they are insufficient to cope with the variety and complexity of geographic reality. From a formal perspective, they are not closed under the geometric set operations *intersection*, *union*, and *difference*. This means that these operations applied to two simple spatial objects can produce a spatial object that is *not* simple. Complex spatial data types solve these problems. They provide universal and versatile spatial objects and are closed under geometric set operations. They allow objects with multiple components, region components that may have holes, and line components that may model ramified, connected geometric networks (Fig. 1d–f).

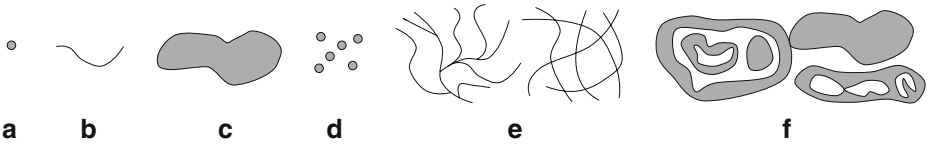

Fig. 1 Examples of a simple point object (**a**), a simple line object (**b**), a simple region object (**c**), a complex point object (**d**), a complex line object (**e**), and a complex region object (**f**)

Implementations of complex spatial data types (under different names) are available in a number of commercial and public domain software systems. Examples are ESRI's Spatial Database Engine (ArcSDE) [17], the Informix Geodetic DataBlade [23] (only complex regions), Oracle Spatial [28], DB2's Spatial Extender [24], and the JTS Topology Suite [37]. All these spatial data type implementations (at least partially) support the specifications of the Open Geospatial Consortium [27], [25] and can be used as a basis for the predicate implementation and evaluation concepts proposed in this article.

2.2 Conceptual models for topological relationships

The conceptual study and determination of topological relationships has led to the two fundamental approaches of the *nine-intersection model* [12], which is based on point set theory and point set topology [18], and the *RCC model* [10], which is based on spatial logic. Despite rather different foundations, both approaches come to very similar results. Our implementation strategy relies heavily on the nine-intersection model since it enables us to create a direct link between the concepts of this model and our implementation approach described in this article as well as to prove the correctness of our strategy.

Based on the nine-intersection model, a complete collection of mutually exclusive topological relationships can be determined for each combination of simple and recently also complex spatial data types. The model is based on the nine possible intersections of the boundary (∂A), the interior (A°), and the exterior (A^-) [18] of a spatial object A with the corresponding components ∂B, B°, and B^- of another spatial object B. Each intersection is tested with regard to the topologically invariant criteria of non-emptiness and emptiness. The topological relationship between two spatial objects A and B can be determined by evaluating the 3×3-matrix in Fig. 2a. We call each matrix element a *matrix predicate*. A total number of $2^9 = 512$ different configurations is possible from which only a certain subset makes sense depending on the *definition* and *combination* of the spatial data types considered. For each combination of spatial types, this means that each of its predicates is associated with a unique *nine-intersection matrix* so that all predicates are mutually exclusive and complete with regard to the topologically invariant criteria of non-emptiness and emptiness.

Topological relationships have been first investigated for simple spatial objects (Fig. 2b), i.e., for two simple regions (*disjoint*, *meet*, *overlap*, *equal*, *inside*, *contains*, *covers*, *coveredBy*) [8], [12], for two simple lines [6], [13], and for a simple line and a simple region [14]. Topological predicates involving simple points are trivial.

$$\begin{pmatrix} A^\circ \cap B^\circ \neq \varnothing & A^\circ \cap \partial B \neq \varnothing & A^\circ \cap B^- \neq \varnothing \\ \partial A \cap B^\circ \neq \varnothing & \partial A \cap \partial B \neq \varnothing & \partial A \cap B^- \neq \varnothing \\ A^- \cap B^\circ \neq \varnothing & A^- \cap \partial B \neq \varnothing & A^- \cap B^- \neq \varnothing \end{pmatrix}$$

a

	point	line	region
point	2/5	3/14	3/7
line	3/14	33/82	19/43
region	3/7	19/43	8/33

b

Fig. 2 The nine-intersection matrix (**a**) and the numbers of topological predicates between two simple/complex spatial objects (**b**)

The implementation concepts in this article can be applied to all these predicate collections.

Two restricted attempts to a definition of topological relationships on more complex spatial objects are the TRCR (topological relationships for composite regions) model [7], which only allows sets of disjoint simple regions without holes and does not derive topological relationships systematically from the underlying model, and the approach in [16], which only considers topological relationships of simple regions with holes, does not permit multi-part regions, and depends on the number of holes of the operand objects. Our predicate implementation concepts are not applicable to these approaches.

Our main target is the implementation of an own, thorough, systematic, and complete specification of mutually exclusive topological relationships for all combinations of complex spatial data types [35] that is also based on the nine-intersection model. This approach uses rules to exclude invalid nine-intersection matrices as well as prototypical drawings of spatial scenarios to show the correctness of the remaining matrices and to visualize the corresponding topological predicates. Figure 2b shows the increase of topological predicates between complex objects compared to those between simple objects and underpins the need for sophisticated and efficient predicate execution techniques. For example, we know 33 topological relationships between two simple line objects but 82 relationships between two complex line objects.

2.3 Conceptual models for dimension-refined topological relationships

A replacement of the topological invariant of non-emptiness and emptiness of an intersection by the topological invariant of the *dimension* of an intersection leads from the nine-intersection matrix (Fig. 2a) to the nine-*intersection dimension matrix* (Fig. 3a). The *dimension-extended model* [4], [8] considers the dimension of intersections and combines it with topological predicates for simple spatial data types. The implementation concepts in this article can be applied to these approaches.

However, our main target is the implementation of an own concept of so-called *dimension-refined topological predicates* [26] for all combinations of complex spatial data types that is based on the nine-intersection dimension matrix and that generalizes the aforementioned approaches. In [26] we have explored which predicates can be derived on the basis of the dimension matrix. It turns out that these predicates are *refinements* of the topological predicates on spatial data types. Figure 3b shows the numbers of these predicates for all combinations of simple spatial data types and for all combinations of complex spatial data types. A comparison with Fig. 2b reveals that all type combinations that include the type *point* cannot be

$$\begin{pmatrix} dim(A^\circ \cap B^\circ) & dim(A^\circ \cap \partial B) & dim(A^\circ \cap B^-) \\ dim(\partial A \cap B^\circ) & dim(\partial A \cap \partial B) & dim(\partial A \cap B^-) \\ dim(A^- \cap B^\circ) & dim(A^- \cap \partial B) & dim(A^- \cap B^-) \end{pmatrix}$$

a

	point	line	region
point	2/5	3/14	3/7
line	3/14	61/146	43/75
region	3/7	43/75	16/53

b

Fig. 3 The nine-intersection dimension matrix (**a**) and the numbers of dimension-refined topological predicates between two simple/complex spatial objects (**b**)

dimension-refined. This is only possible if one-dimensional components of a spatial object like the interior of a line object or the boundary of a region object are involved. These one-dimensional components can interact in several different ways. Either they are disjoint, or they only share a point object, or they only share a line object, or they share a point object and a line object. Consequently, the dimension of the intersection is empty, only zero-dimensional, only one-dimensional, or zero- and one-dimensional. This leads to different refinements of the underlying topological predicates. Our approach enables us to evaluate them.

2.4 Implementation aspects of topological predicates

In queries, topological predicates usually appear as filter conditions of spatial selections and spatial joins. At least two main strategies can be distinguished for their processing. Either we attempt to avoid the execution of topological predicates since they are expensive predicates and cannot be executed in constant time, or we design sophisticated implementation methods for them. The latter strategy is always needed while the former strategy is worthwhile to do.

Avoidance strategies for expensive predicate executions can be leveraged at the algebraic level and at the physical level. At the algebraic level, consistency checking procedures examine the collection of topological relationships contained in a query for topological consistency by employing topological reasoning techniques [31]. Optimization minimizes the number of computations needed [9] and aims at eliminating topological relationships that are implied uniquely by composition [15]. At the physical level, several methods pursue the concept of avoiding unnecessary or repetitive predicate evaluations in query access plans. Examples of these methods are predicate migration [22], predicate placement [21], disjunctive query optimization [3], and approximation-based evaluation [2]. A subsequent, important method is the deployment of spatial index structures to identify those candidate pairs of spatial objects that could possibly fulfil the predicate of interest. This is done by a filtering test on the basis of minimum bounding rectangles.

At some point, avoidance strategies are of no help anymore, and the application of physical predicate execution algorithms is necessary. To the authors' knowledge, there has so far been no published research on the efficient implementation and evaluation of topological predicates, their optimization, and their connection to the underlying theory. All three aspects are main objectives of this article. Only in two own, initial, *ad hoc* attempts [33], [34], the authors extend the concept of the eight topological predicates for two simple regions to a concept and implementation of these eight predicates for two complex regions.

Implementations of topological predicates for complex spatial objects are provided by all spatial data management and analysis packages mentioned in Section 2.1. However, the named topological predicates focus on the eight well known predicates *disjoint*, *meet*, *overlap*, *equal*, *inside*, *contains*, *covers*, and *coveredBy* that are generalized to and unified for all combinations of complex spatial data types. A special method allows the explicit specification of a nine-intersection matrix which is evaluated for two operand objects. In all cases, a formal definition of the behavior of these predicates is missing since descriptions are only informal.

The open source JTS Topology Suite [37] implements the aforementioned eight topological predicates for complex spatial objects through a *topology graph*. Such

a graph stores topology explicitly and records for each node (endpoint) and edge (segment) of a spatial object whether it is located in the interior, in the exterior, or on the boundary of another spatial object. Computing the topology graphs and deriving the nine-intersection matrix from them require quadratic time and quadratic space in terms of the nodes and edges of the two operand objects. Our solution requires linearithmic (loglinear) time and linear space due to the use of the plane sweep paradigm.

3 The exploration phase for collecting topological information

For a given scene of two spatial objects, the goal of the *exploration phase* is to discover appropriate topological information that is characteristic and unique for this scene and that is suitable both for verification queries (query type 1) and determination queries (query type 2). Our general approach is to scan such a scene from left to right by a plane sweep algorithm and to collect appropriate topological data during this traversal in so-called *topological feature vectors*. We call such an algorithm *exploration algorithm*. Later in the evaluation phase, the topological feature vectors help us confirm, deny, or derive the topological relationship between both objects. From both phases, due to the use of plane sweep algorithms, the exploration phase is the computationally (much more) expensive one since topological information has to be explicitly derived by geometric computation.

In Section 3.1, we sketch the well known plane sweep technique and introduce some needed geometric concepts and notations as well as some robust geometric primitives. Section 3.2 motivates the concept of topological feature vectors. The remaining subsections formally define the topological feature vectors for the six different combinations of complex spatial data types.

3.1 Preliminaries

All exploration algorithms are special instances of the plane sweep paradigm. The plane sweep technique is a well known algorithmic scheme in Computational Geometry [1]. Its central idea is to reduce a two-dimensional geometric problem to a simpler one-dimensional geometric problem. A vertical *sweep line* traversing the plane from left to right stops at special *event points* which are stored in a queue called *event point schedule*. The event point schedule must allow one to insert new event points discovered during processing; these are normally the initially unknown intersections of line segments. The state of the intersection of the sweep line with the geometric structure being swept at the current sweep line position is recorded in vertical order in a data structure called *sweep line status*. Whenever the sweep line reaches an event point, the sweep line status is updated. Event points which are passed by the sweep line are removed from the event point schedule. Due to space limitations, we will not discuss the specialties of each exploration algorithm in detail. The interested reader can find their detailed description in [36].

The definition of the topological feature vectors requires some geometric concepts and notations. Due to space limitations, we only explain them informally here; their detailed, formal description can be found in [36]. A *point* object F is represented as a *sequence* of lexicographically ordered points. Let $P(F)$ be the *set* of all points of F. A *line* object F is represented as a *sequence* of ordered *halfsegments*. A halfsegment

(similar to the halfedge concept in CGAL) includes point and segment information and emphasizes one of its end points as the *dominating point*, i.e., point of interest. Hence, it is a hybrid between a (left or right) end point and the incident segment. That is, each segment corresponds to two halfsegments, a left halfsegment and a right halfsegment. In a plane sweep algorithm, a left halfsegment indicates that the sweep line status has to be updated with the segment information; a right halfsegment leads to a removal of the respective segment from the sweep line status. Let $H(F)$ be the *set* of all halfsegments of F. For $f \in H(F)$, let $f.s$ denote its segment component and $dp(f)$ be a function yielding the dominating point of f. Let $B(F)$ be the set of all boundary points of F. These are all those points of F from which exactly one segment emanates. A *region* object F is represented as a *sequence* of ordered *attributed halfsegments*. The attribute is a Boolean flag *ia* for "Interior Above" indicating whether the interior of a region is above or, for vertical segments, left of the halfsegment. Let $H(F)$ be the *set* of all attributed halfsegments of F. For $f \in H(F)$, let $f.s$ denote its segment component and $f.ia$ its attribute component. Note that $H(F)$, where F is a *line* or *region* object, only includes halfsegments that are either disjoint from, equal to, or meeting (in an end point) the halfsegments $H(G)$ of another *line* or *region* object G due to our special splitting strategy during plane sweeps [36].

The definition of topological feature vectors makes use of some *robust geometric primitives* that we assume as predefined. The predicates "$=$" and "$\neq$" check the equality and inequality of two single points respectively. The predicates *on* and *poiInRegion* check whether a single point is located on a segment or inside a *region* object respectively. The predicate *segInRegion* tests whether a segment is located inside a *region* object. The predicates *poiIntersect* and *segIntersect* test whether two segments intersect in a point or a segment respectively. The operation *poiIntersection* returns the intersection point of two segments.

3.2 Topological feature vectors

Our research shows that on the one hand it is unnecessary to design an own exploration algorithm for each single topological predicate and that on the other hand it is unfavorable to aim at designing a *universal* exploration algorithm that covers all topological predicates of all combinations of spatial data types. This has three main reasons. First, each of the data types *point*, *line*, and *region* has very type-specific, well known properties that are different from each other (like different dimensionality). Second, for each type combination of spatial data types, the topological information collected is very specific but equal for all topological predicates of each type combination and different from all other type combinations. Third, the topological information we collect about each single spatial data type is different in different type combinations.

Therefore, we only have to distinguish six cases due to the six possible type combinations between the three spatial data types if we assume that the first operand has an equal or lower dimension than the second operand.[1] This leads to six

[1]If, in the determination query case, a predicate $p(A, B)$ has to be processed, for which the dimension of object A is higher than the dimension of object B, we process the converse predicate $p^{\text{conv}}(B, A)$ where p^{conv} has the transpose of the nine-intersection matrix (see Fig. 2a) of p.

exploration algorithms which except for the *point*/*point* case require the plane sweep technique. Depending on the types of spatial objects involved, a Boolean vector v_F consisting of a special set of *topological feature flags* is assigned to *each* object F. Each topological feature flag indicates the existence or non-existence of a characteristic topological feature, and we call the vector a *topological feature vector*. Its flags are all initialized to *false*. Once certain topological information about an object has been discovered, the corresponding flag of its topological feature vector is set to *true*. For example, if two *line* objects share a segment, the corresponding flag $v_F[seg\_shared]$ is set to *true* (see Definition 4(1) below). For all type combinations, we aim at minimizing the number of topological feature flags of both spatial argument objects. In symmetric cases, only the first object gets the flag. The topological feature vectors are later used in the evaluation phase for predicate matching. Hence, the selection of topological feature flags is highly motivated by the requirements of the evaluation phase (Section 4).

3.3 Exploring topological information for the *point*/*point* case

The first and simplest case considers the exploration of topological information for two *point* objects F and G. Here, the topological facts of interest are whether (1) both objects have a point in common and (2) F (G) contains a point that is not part of G (F). Hence, both topological feature vectors v_F and v_G get the flag *poi_disjoint*. But only v_F in addition gets the flag *poi_shared* since the sharing of a point is symmetric. We obtain (the symbol "$:\Leftrightarrow$" means "equivalent by definition"):

Definition 1 Let $F, G \in point2D$, and let v_F and v_G be their topological feature vectors. Then

1. $v_F[poi\_shared] \quad :\Leftrightarrow \quad \exists f \in P(F)\, \exists g \in P(G) : f = g$
2. $v_F[poi\_disjoint] \quad :\Leftrightarrow \quad \exists f \in P(F)\, \forall g \in P(G) : f \neq g$
3. $v_G[poi\_disjoint] \quad :\Leftrightarrow \quad \exists g \in P(G)\, \forall f \in P(F) : f \neq g$

3.4 Exploring topological information for the *point*/*line* case

In case of a *point* object F and a *line* object G, at the lowest level of detail, we are interested in the possible relative positions between the individual points of F and the halfsegments of G. This requires a precise understanding of the definition of the boundary of a *line* object, as it has been given in [5], [35]. It follows from this definition that each boundary point of G is an endpoint of a (half)segment of G but not necessarily vice versa, as Fig. 4 indicates. The black segment endpoints belong to the boundary of G since exactly one segment emanates from each of them. Intuitively, they "bound" G. In contrast, the grey segment endpoints belong to the interior of G since several segments emanate from each of them. Intuitively, they are "connector points" between different segments of G.

The following argumentation leads to the needed topological feature flags for F and G. Seen from the perspective of F, we can distinguish three cases since the boundary of F is empty [35] and the interior of F can interact with the exterior, interior, or boundary of G. First, (the interior of) a point f of F can be disjoint from

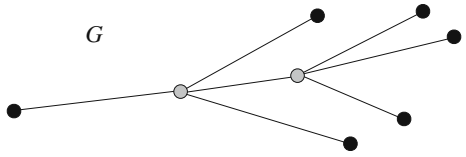

Fig. 4 Boundary points (*in black*) and connector points (*in grey*) of a *line* object

G (flag *poi_disjoint*). Second, a point f can lie in the interior of a segment of G (flag *poi_on_interior*). This includes an endpoint of such a segment, if the endpoint is a connector point of G. Third, a point f can be equal to a boundary point of G (flag *poi_on_bound*).

Seen from the perspective of G, we can distinguish four cases since the boundary and the interior of G can interact with the interior and exterior of F. First, G can contain a boundary point that is unequal to all points in F (flag *bound_poi_disjoint*). Second, G can have a boundary point that is equal to a point in F. But the flag *poi_on_bound* already takes care of this situation. Third, the interior of a segment of G (including connector points) can comprehend a point of F. This situation is already covered by the flag *poi_on_interior*. Fourth, the interior of a segment of G can be part of the exterior of F. This is always true since a segment of G represents an infinite point set that cannot be covered by the finite number of points in F. Hence, we need not handle this as a special situation. More formally, we define the semantics of the topological feature flags as follows:

Definition 2 Let $F \in point$, $G \in line$, and v_F and v_G be their topological feature vectors. Then

1. $v_F[poi\_disjoint] \quad :\Leftrightarrow \exists f \in P(F)\, \forall g \in H(G) : \neg on(f, g.s)$
2. $v_F[poi\_on\_interior] \quad :\Leftrightarrow \exists f \in P(F)\, \exists g \in H(G)\, \forall b \in B(G) : on(f, g.s) \wedge f \neq b$
3. $v_F[poi\_on\_bound] \quad :\Leftrightarrow \exists f \in P(F)\, \exists g \in B(G) : f = g$
4. $v_G[bound\_poi\_disjoint] :\Leftrightarrow \exists g \in B(G)\, \forall f \in P(F) : f \neq g$

3.5 Exploring topological information for the *point*/*region* case

In case of a *point* object F and a *region* object G, the situation is simpler than in the previous case. Seen from the perspective of F, we can again distinguish three cases between (the interior of) a point of F and the exterior, interior, or boundary of G. Either a point of F lies either inside G (flag *poi_inside*), on the boundary of G (flag *poi_on_bound*), or outside of region G (flag *poi_outside*).

Seen from the perspective of G, we can distinguish four cases between the boundary and the interior of G with the interior and exterior of F. The intersection of the boundary (interior) of G with the interior of F implies that a point of F is located on the boundary (inside) of G. This situation is already covered by the flag *poi_on_bound* (*poi_inside*). The intersection of the boundary (interior) of G with the exterior of F is always true since F as a finite point set cannot cover G's boundary segments (interior) representing an infinite point set. More formally, we define the semantics of the topological feature flags as follows (note that v_G is not needed):

Definition 3 Let $F \in point$, $G \in region$, and v_F and v_G be their topological feature vectors. Then

1. $v_F[poi\_inside] \quad :\Leftrightarrow \exists f \in P(F) : poiInRegion(f, G)$
2. $v_F[poi\_on\_bound] :\Leftrightarrow \exists f \in P(F)\, \exists g \in H(G) : on(f, g.s)$
3. $v_F[poi\_outside] \quad :\Leftrightarrow \exists f \in P(F) \forall g \in H(G) : \neg poiInRegion(f, G) \wedge \neg on(f, g.s)$

3.6 Exploring topological information for the *line*/*line* case

Next, we consider the case for two *line* objects F and G. Seen from the perspective of F, we can differentiate six cases between the interior and boundary of F and the interior, boundary, and exterior of G. First, the interiors of two segments of F and G can partially or completely coincide (flag *seg_shared*). Second, if a segment of F does not partially or completely coincide with any segment of G, we register this in the flag *seg_unshared*. Third, we set the flag *interior_poi_shared* if two segments intersect in a single point that does not belong to the boundaries of F or G. Fourth, a boundary endpoint of a segment of F can be located in the interior of a segment (including connector points) of G (flag *bound_on_interior*). Fifth, both objects F and G can share a boundary point (flag *bound_shared*). Sixth, if a boundary endpoint of a segment of F lies outside of all segments of G, we set the flag *bound_disjoint*.

Seen from the perspective of G, we can identify the same cases. But due to the symmetry of three of the six topological cases, we do not need all flags for G. For example, if a segment of F partially coincides with a segment of G, this also holds vice versa. Hence, it is sufficient to introduce the flags *seg_unshared*, *bound_on_interior*, and *bound_disjoint* for G. More formally, we define the semantics of the topological feature flags as follows:

Definition 4 Let $F, G \in line$, and let v_F and v_G be their topological feature vectors. Then

1. $v_F[seg\_shared] \quad :\Leftrightarrow \exists f \in H(F)\, \exists g \in H(G) : segIntersect(f.s, g.s)$
2. $v_F[seg\_unshared] \quad :\Leftrightarrow \exists f \in H(F)\, \forall g \in H(G) : \neg segIntersect(f.s, g.s)$
3. $v_F[interior\_poi\_shared] :\Leftrightarrow \exists f \in H(F)\, \exists g \in H(G)\, \forall p \in B(F) \cup B(G) : poiIntersect(f.s, g.s) \wedge poiIntersection(f.s, g.s) \neq p$
4. $v_F[bound\_on\_interior] :\Leftrightarrow \exists f \in H(F)\, \exists g \in H(G)\, \exists p \in B(F) \setminus B(G) : poiIntersection(f.s, g.s) = p$
5. $v_F[bound\_shared] \quad :\Leftrightarrow \exists p \in B(F)\, \exists q \in B(G) : p = q$
6. $v_F[bound\_disjoint] \quad :\Leftrightarrow \exists p \in B(F)\, \forall g \in H(G) : \neg on(p, g.s)$
7. $v_G[seg\_unshared] \quad :\Leftrightarrow \exists g \in H(G)\, \forall f \in H(F) : \neg segIntersect(f.s, g.s)$
8. $v_G[bound\_on\_interior] :\Leftrightarrow \exists f \in H(F)\, \exists g \in H(G)\, \exists p \in B(G) \setminus B(F) : poiIntersection(f.s, g.s) = p$
9. $v_G[bound\_disjoint] \quad :\Leftrightarrow \exists q \in B(G)\, \forall f \in H(F) : \neg on(q, f.s)$

3.7 Exploring topological information for the *line*/*region* case

Next, we describe the case of a *line* object F and a *region* object G. Seen from the perspective of F, we can distinguish six cases between the interior and boundary of F and the interior, boundary, and exterior of G. First, the intersection of the interiors of F and G means that a segment of F lies in G (flag *seg_inside*). Second, the interior

of a segment of F intersects with a boundary segment of G if either both segments partially or fully coincide (flag *seg_shared*), or if they properly intersect in a single point (flag *poi_shared*). Third, the interior of a segment of F intersects with the exterior of G if the segment is disjoint from G (flag *seg_outside*). Fourth, a boundary point of F intersects the interior of G if the boundary point lies inside of G (flag *bound_inside*). Fifth, if it lies on the boundary of G, we set the flag *bound_shared*. Sixth, if it lies outside of G, we set the flag *bound_disjoint*.

Seen from the perspective of G, we can differentiate the same six cases as before and obtain most of the topological flags as before. First, if the interiors of G and F intersect, a segment of F must partially or totally lie in G (already covered by flag *seg_inside*). Second, if the interior of G and the boundary of F intersect, the boundary point of a segment of F must be located in G (already covered by flag *bound_inside*). Third, the case that the interior of G intersects the exterior of F is always true due to the different dimensionality of both objects; hence, we do not need a flag. Fourth, if the boundary of G intersects the interior of F, a segment of F must partially or fully coincide with a boundary segment of G (already covered by flag *seg_shared*). Fifth, if the boundary of G intersects the boundary of F, a boundary point of a segment of F must lie on a boundary segment of G (already covered by flag *bound_shared*). Sixth, if the boundary of G intersects the exterior of F, a boundary segment of G must be disjoint from F (new flag *seg_unshared*). More formally, we define the semantics of the topological feature flags as follows:

Definition 5 Let $F \in \mathit{line}$, $G \in \mathit{region}$, and v_F and v_G be their topological feature vectors. Then

1. $v_F[\mathit{seg\_inside}] :\Leftrightarrow \exists f \in H(F)\, \forall g \in H(G) : \neg \mathit{segIntersect}(f.s, g.s) \wedge \mathit{segInRegion}(f.s, G)$
2. $v_F[\mathit{seg\_shared}] :\Leftrightarrow \exists f \in H(F)\, \exists g \in H(G) : \mathit{segIntersect}(f.s, g.s)$
3. $v_F[\mathit{seg\_outside}] :\Leftrightarrow \exists f \in H(F)\, \forall g \in H(G) : \neg \mathit{segIntersect}(f.s, g.s) \wedge \neg \mathit{segInRegion}(f.s, G)$
4. $v_F[\mathit{poi\_shared}] :\Leftrightarrow \exists f \in H(F)\, \exists g \in H(G) : \mathit{poiIntersect}(f.s, g.s) \wedge \mathit{poiIntersection}(f.s, g.s) \notin B(F)$
5. $v_F[\mathit{bound\_inside}] :\Leftrightarrow \exists f \in H(F) : \mathit{poiInRegion}(dp(f), G) \wedge dp(f) \in B(F)$
6. $v_F[\mathit{bound\_shared}] :\Leftrightarrow \exists f \in H(F)\, \exists g \in H(G) : \mathit{poiIntersect}(f.s, g.s) \wedge \mathit{poiIntersection}(f.s, g.s) \in B(F)$
7. $v_F[\mathit{bound\_disjoint}] :\Leftrightarrow \exists f \in H(F)\, \forall g \in H(G) : \neg \mathit{poiInRegion}(dp(f), G) \wedge dp(f) \in B(F) \wedge \neg \mathit{on}(dp(f), g.s)$
8. $v_G[\mathit{seg\_unshared}] :\Leftrightarrow \exists g \in H(G)\, \forall f \in H(F) : \neg \mathit{segIntersect}(f.s, g.s)$

3.8 Exploring topological information for the *region*/*region* case

The case of two *region* objects is quite different from the preceding five cases since it has to take into account the areal extent of both objects. The indices of the vector fields, with one exception described below, are *segment classes*. The segment class of a segment s is a pair (m/n) of *overlap numbers* indicating the number of overlapping *region* objects below/above (right of/left of) s in a given scene. In our case, $0 \leq m, n \leq 2$ holds. The fields of each topological feature vector again contain Boolean values that are initialized with *false*. The main goal is to determine the existing segment classes in each *region* object. Hence, the topological feature vector for each of the

two *region* objects is a *segment classification vector*. Each vector contains a field for the segment classes (0/1), (1/0), (0/2), (2/0), (1/2), (2/1), and (1/1). The segment classes (0/1) and (1/0) indicate that a segment of one *region* object is disjoint from the other *region* object and that the interior of the first *region* object is above or below the segment respectively The segment classes (0/2) and (2/0) mean that a segment of one *region* object shares a segment of the other *region* object and that their interiors are on the same side. The segment classes (1/2) and (2/1) indicate that a segment of one *region* object is located inside the other *region* object and that the interior of the first *region* object is above or below the segment respectively. The segment class (1/1) implies that a segment of one *region* object shares a segment of the other *region* object and that the interiors are on different sides. The flag *bound_poi_shared* indicates whether any two unequal boundary segments of both objects share a common point. Before the splitting, such a point may have been a segment endpoint or a proper segment intersection point for each object.

The following definition makes a connection between representational concepts and point set topological concepts as it is later needed in the evaluation phase. For a segment $s = (p, q) \in seg$, the function *pts* yields the infinite point set of s as $pts(s) = \{r \in \mathbb{R}^2 \mid r = p + \lambda(q - p), \lambda \in \mathbb{R}, 0 \leq \lambda \leq 1\}$. Further, for $F \in region$, we define $\partial F = \bigcup_{f \in H(F)} pts(f.s)$, $F^\circ = \{p \in \mathbb{R}^2 \mid poiInRegion(p, F)\}$, and $F^- = \mathbb{R}^2 - \partial F - F^\circ$. We now define the semantics of the vector v_F as follows:

Definition 6 Let $F, G \in region$ and v_F be the segment classification vector of F. Then

1. $v_F[(0/1)] \quad :\Leftrightarrow \exists f \in H(F) : f.ia \wedge pts(f.s) \subset G^-$
2. $v_F[(1/0)] \quad :\Leftrightarrow \exists f \in H(F) : \neg f.ia \wedge pts(f.s) \subset G^-$
3. $v_F[(1/2)] \quad :\Leftrightarrow \exists f \in H(F) : f.ia \wedge pts(f.s) \subset G^\circ$
4. $v_F[(2/1)] \quad :\Leftrightarrow \exists f \in H(F) : \neg f.ia \wedge pts(f.s) \subset G^\circ$
5. $v_F[(0/2)] \quad :\Leftrightarrow \exists f \in H(F)\, \exists g \in H(G) : f.s = g.s \wedge f.ia \wedge g.ia$
6. $v_F[(2/0)] \quad :\Leftrightarrow \exists f \in H(F)\, \exists g \in H(G) : f.s = g.s \wedge \neg f.ia \wedge \neg g.ia$
7. $v_F[(1/1)] \quad :\Leftrightarrow \exists f \in H(F)\, \exists g \in H(G) : f.s = g.s \wedge ((f.ia \wedge \neg g.ia) \vee (\neg f.ia \wedge g.ia))$
8. $v_F[bound\_poi\_shared] :\Leftrightarrow \exists f \in H(F)\, \exists g \in H(G) : f.s \neq g.s \wedge dp(f) = dp(g)$

The segment classification vector v_G of G includes the cases (1) to (4) with F and G swapped; we omit the flags for the cases (5) to (8) due to their symmetry (or equivalence) to flags of F.

4 The evaluation phase for matching topological relationships

In the previous section, we have determined the topological feature vectors v_F and v_G of two complex spatial objects $F \in \alpha$ and $G \in \beta$ with $\alpha, \beta \in \{point, line, region\}$. The vectors v_F and v_G contain specific topological feature flags for each type combination. The flags capture all topological situations between F and G and are different for different type combinations. The goal of the evaluation phase is to leverage the output of the exploration phase, i.e., v_F and v_G, either for verifying a given (proper or dimension-refined) topological predicate or for determining such a predicate. Our general evaluation strategy is to accommodate the objects' topological feature

vectors with an existing topological predicate for both predicate verification and predicate determination.

Section 4.1 presents an *ad hoc* evaluation method called *direct predicate characterization*. Learning from its shortcomings, in Section 4.2, we propose a novel, systematic, provably correct, and general evaluation method called *nine-intersection matrix characterization*. The next two subsections elaborate on a particular step of the general method that is dependent on the type combination under consideration. Section 4.4 deals with the special *region*/*region* case while Section 4.3 handles the cases of all other type combinations. Section 4.5 applies and extends our approach to the evaluation of dimension-refined topological predicates.

4.1 Direct predicate characterization as a simple evaluation method

The first method provides a *direct predicate characterization* of all n topological predicates of each type combination (see Fig. 2b) for the different values of n) and is based on the topological feature flags of v_F and v_G of the two spatial argument objects F and G. That is, for the *line*/*line* case, we have to determine which topological feature flags of v_F and v_G must be turned on and which flags must be turned off so that a given topological predicate (verification query) or a predicate to be found (determination query) is fulfilled. For the *region*/*region* case, the central question is to which segment classes the segments of both objects must belong so that a given topological predicate or a predicate to be found is satisfied. The direct predicate characterization gives an answer for each individual predicate of each individual type combination. This means that we obtain 184 individual predicate characterizations without converse predicates and 248 individual predicate characterizations with converse predicates. In general, each characterization is a Boolean expression in conjunctive normal form and expressed in terms of the topological feature vectors v_F and v_G.

We give two examples of direct predicate characterizations. As a first example, we consider the topological predicate number 8 (*meet*) between two *line* objects F and G (Fig. 5a and [35]) and see how the flags of the topological feature vectors (Definition 4) are used.

$$\begin{aligned} p_8(F,G) \;:\Leftrightarrow\; & \neg v_F[\mathit{seg\_shared}] \wedge \neg v_F[\mathit{interior\_poi\_shared}] \wedge v_F[\mathit{seg\_unshared}]v \;\wedge \\ & \neg v_F[\mathit{bound\_on\_interior}] \wedge v_F[\mathit{bound\_shared}] \wedge v_F[\mathit{bound\_disjoint}] \wedge \\ & v_G[\mathit{seg\_unshared}] \wedge \neg v_G[\mathit{bound\_on\_interior}] \wedge v_G[\mathit{bound\_disjoint}] \end{aligned}$$

$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

a **b**

Fig. 5 The nine-intersection matrix number 8 for the predicate *meet* between two *line* objects (**a**) and the nine-intersection matrix number 7 for the predicate *inside* between two *region* objects (**b**)

If we take into account the semantics of the topological feature flags, the right side of the equivalence means that both objects may only and must share boundary parts. More precisely and by considering the matrix in Fig. 5a, intersections between both interiors ($\neg v_F$[*seg_shared*], $\neg v_F$[*interior_poi_shared*]) as well as between the boundary of one object and the interior of the other object ($\neg v_F$[*bound_on_interior*], $\neg v_G$[*bound_on_interior*]) are not allowed; besides intersections between both boundaries (v_F[*bound_shared*), each component of one object must interact with the exterior of the other object (v_F[*seg_unshared*], v_G[*seg_unshared*], v_F[*bound_disjoint*], v_G[*bound_disjoint*]).

Next, we view the topological predicate number 7 (*inside*) between two *region* objects F and G (Fig. 5b and [35]) and see how the segment classes kept in the topological feature vectors (Definition 6) are used.

$$p_7(F,G) \; :\Leftrightarrow \neg v_F[(0/1)] \wedge \neg v_F[(1/0)] \wedge \neg v_F[(0/2)] \wedge \neg v_F[(2/0)] \wedge \neg v_F[(1/1)] \wedge$$
$$\neg v_F[\textit{bound\_poi\_shared}] \wedge (v_F[(1/2)] \vee v_F[(2/1)]) \wedge$$
$$\neg v_G[(1/2)] \wedge \neg v_G[(2/1)] \wedge (v_G[(0/1)] \vee v_G[(1/0)])$$

For the *inside* predicate, the segments of F must be located inside of G since the interior and boundary of F must be located in the interior of G; hence they must all have the segment classes (1/2) or (2/1). This "for all" quantification is tested by checking whether $v_F[(1/2)]$ or $v_F[(2/1)]$ are *true* and whether *all* other vector fields are *false*. The fact that all other vector fields are *false* means that the interior and boundary of F do not interact with the boundary and exterior of G. That is, the segments of G must be situated outside of F, and thus they *all* must have the segment classes (0/1) or (1/0); other segment classes are forbidden for G. Further, we must ensure that no segment of F shares a common point with any segment of G ($\neg v_F$[*bound_poi_shared*]).

The predicate characterizations can be read in both directions. If we are interested in *predicate verification*, i.e., in evaluating a specific topological predicate, we look from left to right and check the respective right side of the predicate's direct characterization. This corresponds to an explicit implementation of each individual predicate. If we are interested in *predicate determination*, i.e., in deriving the topological relationship from a given spatial configuration of two spatial objects, we have to look from right to left. That is, consecutively we evaluate the right sides of the predicate characterizations by applying them to the given topological feature vectors v_F and v_G. For the characterization that matches we look on its left side to obtain the name or number of the predicate.

The direct predicate characterization demonstrates how we can leverage the concept of topological feature vectors. However, this particular evaluation method has three main drawbacks. First, the method depends on the number of topological predicates. That is, each of the 184 (248) topological predicates between complex spatial objects requires an own specification. Second, in the worst case, all direct predicate characterizations with respect to a particular type combination have to be checked for predicate determination. Third, the direct predicate characterization is error-prone. It is difficult to ensure that each predicate characterization is correct and unique and that all predicate characterizations together are mutually exclusive

and cover all topological predicates. From this standpoint, this solution is an *ad hoc* approach.

4.2 The nine-intersection matrix characterization method

The drawbacks of the direct predicate characterization are the motivation for another, novel approach called *nine-intersection matrix characterization* (*9IMC*) that avoids these shortcomings. In particular, its correctness can be formally proved. Instead of characterizing each topological predicate directly, the central idea of our second approach is to uniquely characterize each element of the 3×3-matrix of the nine-intersection model (Fig. 2a) by means of the topological feature vectors v_F and v_G. As we know, each matrix element is a predicate called *matrix predicate* that checks one of the nine intersections between the boundary ∂F, interior F°, or exterior F^- of a spatial object F with the boundary ∂G, interior G°, or exterior G^- of another spatial object G for inequality to the empty set. For each topological predicate, its specification is then given as the logical conjunction of the characterizations of the nine matrix predicates. Since the topological feature vectors are different for each type combination, the characterization of each matrix predicate is different for each type combination too. The characterizations themselves are the themes of the next subsections.

The general method for *predicate verification* works as follows. Based on the topological predicate p to be verified as well as v_F and v_G as input, we evaluate in a loop the characterizations of all matrix predicates numbered from left to right and from top to bottom. The ninth matrix predicate $F^- \cap G^- \neq \varnothing$ always yields *true* [35]; hence, we do not have to check it. After the computation of the value of the matrix predicate i ($1 \leq i \leq 8$), we compare it to the corresponding value of the matrix predicate $p(i)$ of p. If the values are equal, we proceed with the next matrix predicate $i + 1$. Otherwise, we stop, and p yields *false*. If there is a coincidence between the computed values of all matrix predicates with the corresponding values of p's matrix, p yields *true*. The benefit of this approach is that it only requires eight predicate characterizations and that these characterizations are the same for each of the n topological predicates of the same type combination. In particular, an individual characterization of all n topological predicates is not needed. In Section 5.1, we show that this method can be even further improved.

The general method for *predicate determination* works as follows. Based on v_F and v_G as input, we evaluate the 9IM characterizations of all eight matrix predicates and insert the Boolean values into an intersection matrix m initialized with *true* for each matrix predicate. Matrix m is then compared against the matrices p_i ($1 \leq i \leq n$) of all n topological predicates. We know that one of them must match m. The merit of this approach is that only eight characterizations are needed to determine the intersection matrix of the topological predicate. But unfortunately we need n matrix comparisons to determine the pertaining topological predicate in the worst case. In Section 5.2, we introduce a method that eliminates this problem. But the method here is already a significant improvement compared with the necessity to compute all n direct predicate characterizations.

We apply the general method and the characterizations to both proper and dimension-refined topological predicates. However, the dimension-refined predicates require some additional characterizations.

4.3 Type combination dependent nine-intersection matrix characterization

The last, missing step refers to the characterizations of the eight matrix predicates of the nine-intersection matrix for all spatial data type combinations. A 9IMC means that each matrix predicate, which takes abstract, infinite point sets F and G representing spatial objects as arguments, is uniquely characterized by the topological feature vectors v_F and v_G, which are discrete implementation concepts. For this purpose, for each discrete spatial object $F \in \alpha \in \{point, line, region\}$, we determine the corresponding abstract point sets of its boundary, interior, and exterior. For the *region* data type, we have already done this for Definition 6. For $F \in point$, we define $\partial F = \varnothing$, $F^\circ = P(F)$, and $F^- = \mathbb{R}^2 - P(F)$. For $F \in line$, we define $\partial F = \{p \in \mathbb{R}^2 \mid card(\{f \in H(F) \mid p = dp(f)\}) = 1\}$, $F^\circ = \bigcup_{f \in H(F)} pts(f.s) - \partial F$, and $F^- = \mathbb{R}^2 - \partial F - F^\circ$. As we will see, each characterization can be performed in constant time, and its correctness can be shown by a simple proof. In this subsection, we present the characterizations for all type combinations except for the more complicated case of two *region* objects; this case is dealt with in the next subsection. The central idea in the proofs of the lemmas below is to accomplish a correspondence between a matrix predicate based on the point sets ∂F, F°, F^-, ∂G, G°, and G^- and an equivalent Boolean expression based on finite representations like $P(F)$, $H(F)$, $B(F)$, $P(G)$, $H(G)$, and $B(G)$.

In case of two *point* objects, the 3×3-matrix is reduced to a 2×2-matrix since the boundary of a *point* object is defined to be empty [35]. We obtain the following statement:

Lemma 1 *Let $F, G \in point$. Then the characterization of the matrix predicates of the (reduced) nine-intersection matrix is as follows:*

1. $F^\circ \cap G^\circ \neq \varnothing \quad \Leftrightarrow \quad v_F[poi\_shared]$
2. $F^\circ \cap G^- \neq \varnothing \quad \Leftrightarrow \quad v_F[poi\_disjoint]$
3. $F^- \cap G^\circ \neq \varnothing \quad \Leftrightarrow \quad v_G[poi\_disjoint]$
4. $F^- \cap G^- \neq \varnothing \quad \Leftrightarrow \quad true$

Proof In (1), the intersection of the interiors of F and G is non-empty if, and only if, both objects share a point. That is, $\exists\, f \in P(F) \exists\, g \in P(G) : equal(f, g)$. This matches directly the definition of $v_F[poi\_shared]$ in Definition 1(1). In (2), a point of F can only be part of the exterior of G if it does not belong to G. That is, $\exists\, f \in P(F) \forall\, g \in P(G) : \neg equal(f, g)$. This fits directly to the definition of $v_F[poi\_disjoint]$ in Definition 1(2). Case (3) is symmetric to (2). Case (4) follows from Lemma 5.1.2 in [35]. □

In case of a *point* object and a *line* object, the 3×3-matrix is reduced to a 2×3-matrix since the boundary of a *point* object is defined to be empty. We obtain the following statement:

Lemma 2 *Let $F \in point$ and $G \in line$. Then the characterization of the matrix predicates of the (reduced) nine-intersection matrix is as follows:*

1. $F^\circ \cap G^\circ \neq \varnothing \quad \Leftrightarrow \quad v_F[poi\_on\_interior]$
2. $F^\circ \cap \partial G \neq \varnothing \quad \Leftrightarrow \quad v_F[poi\_on\_bound]$

3. $F^\circ \cap G^- \neq \varnothing \quad \Leftrightarrow \quad v_F[poi\_disjoint]$
4. $F^- \cap G^\circ \neq \varnothing \quad \Leftrightarrow \quad true$
5. $F^- \cap \partial G \neq \varnothing \quad \Leftrightarrow \quad v_G[bound\_poi\_disjoint]$
6. $F^- \cap G^- \neq \varnothing \quad \Leftrightarrow \quad true$

Proof In (1), the intersection of the interiors of F and G is non-empty if, and only if, a point of F is located on G but is not a boundary point of G. That is, $\exists f \in P(F) \exists g \in H(G) \forall b \in B(G) : on(f, g.s) \ \wedge \ f \neq b$. This corresponds directly to the definition of $v_F[poi\_on\_interior]$ in Definition 2(2). In (2), the intersection of the interior of F and the boundary of G is non-empty if, and only if, a point of F coincides with a boundary point of G. That is, $\exists f \in P(F) \exists g \in B(G) : f = g$. But this matches the definition of $v_F[poi\_on\_bound]$ in Definition 2(3). Statement (3) is satisfied if, and only if, a point of F is outside of G. That is, $\exists f \in P(F) \forall g \in H(G) : \neg on(f, g.s)$. But this is just the definition of $v_F[poi\_disjoint]$ in Definition 2(1). Statement (4) always holds according to Lemma 6.1.2 in [35]. To be fulfilled, statement (5) requires that a boundary point of G lies outside of F. That is, $\exists g \in B(G) \forall f \in P(F) : f \neq g$. This corresponds to the definition of $v_G[bound\_poi\_disjoint]$ in Definition 2(4). The last statement follows from Lemma 6.1.3 in [35]. □

In case of a *point* object and a *region* object, we also obtain a reduction of the 3×3-matrix to a 2×3-matrix. We obtain the following statement:

Lemma 3 *Let $F \in$ point and $G \in$ region. Then the characterization of the matrix predicates of the* (*reduced*) *nine-intersection matrix is as follows:*

1. $F^\circ \cap G^\circ \neq \varnothing \quad \Leftrightarrow \quad v_F[poi\_inside]$
2. $F^\circ \cap \partial G \neq \varnothing \quad \Leftrightarrow \quad v_F[poi\_on\_bound]$
3. $F^\circ \cap G^- \neq \varnothing \quad \Leftrightarrow \quad v_F[poi\_outside]$
4. $F^- \cap G^\circ \neq \varnothing \quad \Leftrightarrow \quad true$
5. $F^- \cap \partial G \neq \varnothing \quad \Leftrightarrow \quad true$
6. $F^- \cap G^- \neq \varnothing \quad \Leftrightarrow \quad true$

Proof Statement (1) requires that a point of F is located inside G but not on the boundary of G. That is, $\exists f \in P(F) : poiInRegion(f, G)$ (where *poiInRegion* is the predicate which checks whether a single point lies inside a *region* object). This corresponds directly to the definition of $v_F[poi\_inside]$ in Definition 3(1). In (2), the intersection of F and the boundary of G is non-empty if, and only if, a point of F lies on one of the boundary segments of G. That is, $\exists f \in P(F) \exists (g, ia) \in H(G) : on(f, g.s)$. This matches the definition of $v_F[poi\_on\_bound]$ in Definition 3(2). Statement (3) is satisfied if, and only if, a point of F is outside of G. That is, $\exists f \in P(F) \forall (g, ia) \in H(G) : \neg poiInRegion(f, G) \ \wedge \ \neg on(f, g.s)$. This corresponds to the definition of $v_F[poi\_outside]$ in Definition 3(3). Statements (4) and (5) follow from Lemma 6.2.3 in [35]. The last statement follows from Lemma 6.2.1 in [35]. □

In case of two *line* objects, we obtain the following statement:

Lemma 4 *Let $F, G \in line$. Then the characterization of the matrix predicates of the nine-intersection matrix is as follows:*

1. $F^\circ \cap G^\circ \neq \varnothing \quad \Leftrightarrow \quad v_F[seg\_shared] \vee v_F[interior\_poi\_shared]$
2. $F^\circ \cap \partial G \neq \varnothing \quad \Leftrightarrow \quad v_G[bound\_on\_interior]$
3. $F^\circ \cap G^- \neq \varnothing \quad \Leftrightarrow \quad v_F[seg\_unshared]$
4. $\partial F \cap G^\circ \neq \varnothing \quad \Leftrightarrow \quad v_F[bound\_on\_interior]$
5. $\partial F \cap \partial G \neq \varnothing \quad \Leftrightarrow \quad v_F[bound\_shared]$
6. $\partial F \cap G^- \neq \varnothing \quad \Leftrightarrow \quad v_F[bound\_disjoint]$
7. $F^- \cap G^\circ \neq \varnothing \quad \Leftrightarrow \quad v_G[seg\_unshared]$
8. $F^- \cap \partial G \neq \varnothing \quad \Leftrightarrow \quad v_G[bound\_disjoint]$
9. $F^- \cap G^- \neq \varnothing \quad \Leftrightarrow \quad true$

Proof In (2), the interiors of two *line* objects intersect if, and only if, any two segments partially or completely coincide or if two segments share a single point that does not belong to the boundaries of F and G. That is, $\exists f \in H(F) \exists g \in H(G) : segIntersect(f.s, g.s) \vee \exists f \in H(F) \exists g \in H(G) \forall p \in B(F) \cup B(G) : poiIntersect(f.s, g.s) \wedge poiIntersection(f.s, g.s) \neq p$. The first expression corresponds to the definition of $v_F[seg\_shared]$ in Definition 4(1). The second expression is the definition of $v_F[interior\_poi\_shared]$ in Definition 4(2). Statement (2) requires that an intersection point p of F and G exists such that p is a boundary point of G but not a boundary point of F. That is, $\exists f \in H(F) \exists g \in H(G) \exists p \in B(G) \backslash B(F) : poiIntersection(f.s, g.s) = p$. This matches the definition of $v_G[bound\_on\_interior]$ in Definition 4(8). Statement (3) is satisfied if, and only if, there is a segment of F that is outside of G. That is, $\exists f \in H(F) \forall g \in H(G) : \neg segIntersect(f.s, g.s)$. This corresponds to the definition of $v_F[seg\_unshared]$ in Definition 4(3). Statement (4) is symmetric to statement (2) and based on Definition 4(4). In (5), the boundaries of F and G intersect if, and only if, they share a boundary point. That is, $\exists p \in B(F) \exists q \in B(G) : p = q$. This matches the definition of $v_F[bound\_shared]$ in Definition 4(5). Statement (6) requires the existence of a boundary point of F that is not located on any segment of G. That is, $\exists p \in B(F) \forall g \in H(G) : \neg on(p, g.s)$. This corresponds to the definition of $v_F[bound\_disjoint]$ in Definition 4(6). Statement (7) is symmetric to statement (3) and based on Definition 4(7). Statement (8) is symmetric to statement (6) and based on Definition 4(9). The last statement follows from Lemma 5.2.1 in [35]. □

In case of a *line* object and a *region* object, we obtain the following statement:

Lemma 5 *Let $F \in line$ and $G \in region$. Then the characterization of the matrix predicates of the nine-intersection matrix is as follows*:

1. $F^\circ \cap G^\circ \neq \varnothing \quad \Leftrightarrow \quad v_F[seg\_inside]$
2. $F^\circ \cap \partial G \neq \varnothing \quad \Leftrightarrow \quad v_F[seg\_shared] \vee v_F[poi\_shared]$
3. $F^\circ \cap G^- \neq \varnothing \quad \Leftrightarrow \quad v_F[seg\_outside]$
4. $\partial F \cap G^\circ \neq \varnothing \quad \Leftrightarrow \quad v_F[bound\_inside]$
5. $\partial F \cap \partial G \neq \varnothing \quad \Leftrightarrow \quad v_F[bound\_shared]$
6. $\partial F \cap G^- \neq \varnothing \quad \Leftrightarrow \quad v_F[bound\_disjoint]$

7. $F^- \cap G^\circ \neq \varnothing \quad \Leftrightarrow \quad true$
8. $F^- \cap \partial G \neq \varnothing \quad \Leftrightarrow \quad v_G[seg\_unshared]$
9. $F^- \cap G^- \neq \varnothing \quad \Leftrightarrow \quad true$

Proof In (1), the interiors of F and G intersect if, and only if, a segment of F is located in G but does not coincide with a boundary segment of G. That is, $\exists f \in H(F) \forall g \in H(G) : \neg segIntersect(f.s, g.s) \ \wedge \ segInRegion(f.s, G)$. This corresponds to the definition of $v_F[seg\_inside]$ in Definition 5(1). Statement (2) requires that either F and G share a segment, or they share an intersection point that is not a boundary point of F. That is, $\exists f \in H(F) \exists g \in H(G) : segIntersect(f.s, g.s) \ \vee \ \exists f \in H(F) \exists g \in H(G) : poiIntersect(f.s, g.s) \ \wedge \ poiIntersection(f.s, g.s) \notin B(F)$. The first argument of the disjunction matches the definition of $v_F[seg\_shared]$ in Definition 5(2). The second argument matches the definition of $v_F[poi\_shared]$ in Definition 5(4). Statement (3) is satisfied if, and only if, a segment of F is located outside of G. That is, $\exists f \in H(F) \forall g \in H(G) : \neg segIntersect(f.s, g.s) \ \wedge \ \neg segInRegion(f.s, G)$. This corresponds to the definition of $v_F[seg\_outside]$ in Definition 5(3). Statement (4) holds if, and only if, a segment of F lies inside G and one of the end points of the segment is a boundary point. That is, $\exists f \in H(F) : poiInRegion(dp(f), G) \ \wedge \ dp(f) \in B(F)$. This corresponds to the definition of $v_F[bound\_inside]$ in Definition 5(5). In (5), we must find a segment of F and a segment of G which intersect in a point that is a boundary point of F. That is, $\exists f \in H(F) \exists g \in H(G) : poiIntersect(f.s, g.s) \ \wedge \ poiIntersection(f.s, g.s) \in B(F)$. This matches the definition of $v_F[bound\_shared]$ in Definition 5(6). Statement (6) requires the existence of an endpoint of a segment of F that is a boundary point and not located inside or on any segment of G. That is, $\exists f \in H(F) \forall g \in H(G) : \neg poiInRegion(dp(f), G) \ \wedge \ dp(f) \in B(F) \ \wedge \ \neg on(dp(f), g.s)$. This corresponds to the definition of $v_F[bound\_disjoint]$ in Definition 5(7). Statement (7) always holds according to Lemma 6.3.2 in [35]. Statement (8) is satisfied if, and only if, a segment of G does not coincide with any segment of F. That is, $\exists g \in H(G) \forall f \in H(F) : \neg segIntersect(f.s, g.s)$. This fits to the definition of $v_F[seg\_unshared]$ in Definition 5(8). The last statement follows from Lemma 6.3.1 in [35]. □

4.4 Nine-intersection matrix characterization of the *region*/*region* case

As shown in Section 3.8, exploring the *region*/*region* case is quite different from exploring the other type combinations and requires another kind of exploration algorithm. It has to take into account the areal extent of both objects and has resulted in the concepts of overlap number, segment classes, and segment classification vector. In this subsection, we deal with the 9IMC based on two segment classification vectors. The goal of the following lemmas is to prepare the unique characterization of all matrix predicates by means of segment classes. The first lemma provides a translation of each segment class into a Boolean matrix predicate expression.

Lemma 6 *Let $F, G \in region$ and v_F and v_G be their segment classification vectors. Then we can infer the following implications and equivalences between segment classes and matrix predicates*:

1. $v_F[(0/1)] \ \vee \ v_F[(1/0)] \quad \Leftrightarrow \quad \partial F \cap G^- \neq \varnothing$
2. $v_G[(0/1)] \ \vee \ v_G[(1/0)] \quad \Leftrightarrow \quad F^- \cap \partial G \neq \varnothing$

3. $v_F[(1/2)] \vee v_F[(2/1)] \Leftrightarrow \partial F \cap G^\circ \neq \varnothing$
4. $v_G[(1/2)] \vee v_G[(2/1)] \Leftrightarrow F^\circ \cap \partial G \neq \varnothing$
5. $v_F[(0/2)] \vee v_F[(2/0)] \Rightarrow \partial F \cap \partial G \neq \varnothing \wedge F^\circ \cap G^\circ \neq \varnothing$
6. $v_F[(1/1)] \Rightarrow \partial F \cap \partial G \neq \varnothing \wedge F^\circ \cap G^- \neq \varnothing \wedge F^- \cap G^\circ \neq \varnothing$

Proof According to Definition 6(1) and (2), the left side of (1) is equivalent to the expression $\exists f \in H(F) : pts(f.s) \subset G^-$. This is equivalent to $\partial F \cap G^- \neq \varnothing$. The proof of (3) is similar and based on Definition 6(3) and (4); only the term G^- has to be replaced by G°. The proof of (2) can be obtained by swapping the roles of F and G in (1). Similarly, the proof of (4) requires a swapping of F and G in (3). According to Definition 6(5) and (6), the left side of (5) is equivalent to the expression $\exists f \in H(F) \exists g \in H(G) : f.s = g.s \wedge ((f.ia \wedge g.ia) \vee (\neg f.ia \wedge \neg g.ia))$. From the first element of the conjunction, we can (only) conclude that $\partial F \cap \partial G \neq \varnothing$. Equivalence does not hold since two boundaries can also intersect if they only share single intersection or meeting points but not (half)segments. The second element of the conjunction requires that the interiors of both *region* objects are located on the same side. Hence, $F^\circ \cap G^\circ \neq \varnothing$ must hold. Also this is only an implication since an intersection of both interiors is possible without having any (0/2)- or (2/0)-segments. According to Definition 6(7), the left side of (6) is equivalent to the expression $\exists f \in H(F) \exists g \in H(G) : f.s = g.s \wedge ((f.ia \wedge \neg g.ia) \vee (\neg f.ia \wedge g.ia))$. The first element of the conjunction implies that $\partial F \cap \partial G \neq \varnothing$. The second element of the conjunction requires that the interiors of both *region* objects are located on different sides. Since the definition of type *region* disallows (1/1)-segments for single objects, the interior of F must intersect the exterior of G, and vice versa. This is only an implication since an intersection of the interior of one *region* object with the exterior of another *region* object is possible without having (1/1)-segments. □

The second lemma provides a translation of some matrix predicates into segment classes.

Lemma 7 *Let $F, G \in$ region and v_F and v_G be their segment classification vectors. Then we can infer the following implications between matrix predicates and segment classes*:

1. $F^\circ \cap G^\circ \neq \varnothing \Rightarrow v_F[(0/2)] \vee v_F[(2/0)] \vee v_F[(1/2)] \vee v_F[(2/1)] \vee v_G[(1/2)] \vee v_G[(2/1)]$
2. $F^\circ \cap G^- \neq \varnothing \Rightarrow v_F[(0/1)] \vee v_F[(1/0)] \vee v_F[(1/1)] \vee v_G[(1/2)] \vee v_G[(2/1)]$
3. $F^- \cap G^\circ \neq \varnothing \Rightarrow v_F[(1/2)] \vee v_F[(2/1) \vee v_F[(1/1)] \vee v_G[(0/1)] \vee v_G[(1/0)]$

Proof In (1), the intersection of the interiors of F and G implies that both objects share a common area. Consequently, this area must have overlap number 2 so that at least one of the two objects must have a $(a/2)$- or $(2/a)$-segment with $a \in \{0, 1\}$. In (2), the fact that F° intersects G^- means that F contains an area which it does not share with G. That is, the overlap number of this area is 1, and F must have a $(a/1)$- or a $(1/a)$-segment with $a \in \{0, 1\}$. The fact that a part of the interior of F is located outside of G implies two possible topological situations for G: either both objects share a common segment and their interiors are on different sides, i.e., G

has a (1/1)-segment (covered by $v_F[(1/1)]$), or the interior of F is intersected by the boundary and the interior of G so that G has a (1/2)- or (2/1)-segment. We prove (3) by swapping F and G in (2). □

The third lemma states some implications between matrix predicates.

Lemma 8 *Let $F, G \in$ region. Then we can infer the following implications between matrix predicates*:

1. $v_F[bound\_poi\_shared] \Rightarrow \partial F \cap \partial G \neq \varnothing$
2. $\partial F \cap G^- \neq \varnothing \Rightarrow F^\circ \cap G^- \neq \varnothing \wedge F^- \cap G^- \neq \varnothing$
3. $F^- \cap \partial G \neq \varnothing \Rightarrow F^- \cap G^\circ \neq \varnothing \wedge F^- \cap G^- \neq \varnothing$
4. $\partial F \cap G^\circ \neq \varnothing \Rightarrow F^\circ \cap G^\circ \neq \varnothing \wedge F^- \cap G^\circ \neq \varnothing$
5. $F^\circ \cap \partial G \neq \varnothing \Rightarrow F^\circ \cap G^\circ \neq \varnothing \wedge F^\circ \cap G^- \neq \varnothing$

Proof Statement (1) can be shown by considering the definition of *bound_poi_shared*. This flag is *true* if any two halfsegments of F and G share a single meeting or intersection point. Hence, the intersection of both boundaries is non-empty. The proofs for (2) to (5) require point set topological concepts. Statements (2) and (3) follow from Lemma 5.3.6 in [35]. Statements (4) and (5) result from Lemma 5.3.5 in [35]. □

The following theorem collects the results we have already obtained so far and proves the lacking parts of the nine matrix predicate characterizations.

Theorem 1 *Let $F, G \in$ region and v_F and v_G be their segment classification vectors. Then the matrix predicates of the nine-intersection matrix are equivalent to the following segment class characterizations*:

1. $F^\circ \cap G^\circ \neq \varnothing \Leftrightarrow v_F[(0/2)] \vee v_F[(2/0)] \vee v_F[(1/2)] \vee v_F[(2/1)] \vee v_G[(1/2)] \vee v_G[(2/1)]$
2. $F^\circ \cap \partial G \neq \varnothing \Leftrightarrow v_G[(1/2)] \vee v_G[(2/1)]$
3. $F^\circ \cap G^- \neq \varnothing \Leftrightarrow v_F[(0/1)] \vee v_F[(1/0)] \vee v_F[(1/1)] \vee v_G[(1/2)] \vee v_G[(2/1)]$
4. $\partial F \cap G^\circ \neq \varnothing \Leftrightarrow v_F[(1/2)] \vee v_F[(2/1)]$
5. $\partial F \cap \partial G \neq \varnothing \Leftrightarrow v_F[(0/2)] \vee v_F[(2/0)] \vee v_F[(1/1)] \vee v_F[bound\_poi\_shared]$
6. $\partial F \cap G^- \neq \varnothing \Leftrightarrow v_F[(0/1)] \vee v_F[(1/0)]$
7. $F^- \cap G^\circ \neq \varnothing \Leftrightarrow v_F[(1/2)] \vee v_F[(2/1)] \vee v_F[(1/1)] \vee v_G[(0/1)] \vee v_G[(1/0)]$
8. $F^- \cap \partial G \neq \varnothing \Leftrightarrow v_G[(0/1)] \vee v_G[(1/0)]$
9. $F^- \cap G^- \neq \varnothing \Leftrightarrow true$

Proof For (1), the forward implication corresponds to Lemma 7(1). The backward implication can be derived from Lemma 6(5) for (0/2)- and (2/0)-segments of F (and G). For (1/2)- and (2/1)-segments, Lemma 6(3) and 6(4) imply $\partial F \cap G^\circ \neq \varnothing$ and $F^\circ \cap \partial G \neq \varnothing$, respectively. From these two implications, by using Lemma 8(4) and 8(5), we can derive in both cases $F^\circ \cap G^\circ \neq \varnothing$. Statements (2) and (4) correspond to Lemma 6(4) and 6(3), respectively. For (3) [(7)], the forward implication corresponds to Lemma 7(2) [7(3)]. The backward implication for (3) [(7)] requires Lemma 6(1) [6(2)] and Lemma 8(2) [8(3)] for the (0/1)- and (1/0)-segments of F [G], Lemma 6(6) [6(6)] for the (1/1)-segments of F (and hence G), as well as Lemma 6(4)

[6(3)] and Lemma 8(5) [8(4)] for the (1/2)- and (2/1)-segments of G [F]. For (5), the forward implication can be shown as follows: if the boundaries of F and G intersect, then either they share a common meeting or intersection point, i.e., the flag v_F[*bound_poi_shared*] is set, or there are two halfsegments of F and G whose segment components are equal. No other alternative is possible due to our splitting strategy for halfsegments during the plane sweep. As we know, equal segments of F and G must have the segment classes (0/2), (2/0), or (1/1). The backward implication requires Lemma 6(5) for (0/2)- and (2/0)-segments of F (and hence G), Lemma 6(6) for (1/1)-segments of F (and hence G), and Lemma 8(1) for single meeting and intersection points. Statement (6) [(8)] corresponds to Lemma 6(1) [6(2)]. Statement (9) turns out to be always true since our assumption in an implementation is that our universe of discourse U is always properly larger than the union of spatial objects contained in it. This means for F and G that always $F \cup G \subset U$ holds. We can conclude that $U - (F \cup G) \neq \varnothing$. According to DeMorgan's Laws, this is equivalent to $(U - F) \cap (U - G) \neq \varnothing$. But this leads us to the statement that $F^- \cap G^- \neq \varnothing$. □

Summarizing our results from the last two subsections, we see that Lemmas 1 to 5, and Theorem 1 provide us with a unique characterization of each individual matrix predicate of the nine-intersection matrix for each type combination. This approach has several benefits. First, it is a systematically developed and not an *ad hoc* approach. Second, it has a formal and sound foundation. Hence, we can be sure about the correctness of topological feature flags and segment classes assigned to matrix predicates, and vice versa. Third, this evaluation method is independent of the number of topological predicates and only requires a constant number of evaluations for matrix predicate characterizations. Instead of nine, even only eight matrix predicates have to be checked since the predicate $F^- \cap G^- \neq \varnothing$ yields *true* for all type combinations. Fourth, we have proved the correctness of our provided implementation.

Based on this result, we accomplish the *predicate verification* of a topological predicate p with respect to a particular spatial data type combination on the basis of p's nine-intersection matrix (as an example, see the complete matrices of the 33 topological predicates of the *region*/*region* case in Fig. 10 and the complete matrices for the remaining cases in [30]) and the topological feature vectors v_F and v_G as follows: Depending on the spatial data type combination, we evaluate the logical expression (given in terms of v_F and v_G) on the right side of the first 9IMC according to Lemma 1, 2, 3, 4, 5, or Theorem 1, respectively. We then match the Boolean result with the Boolean value at the respective position in p's intersection matrix. If both Boolean values are equal, we proceed with the next matrix predicate in the nine-intersection matrix; otherwise p is *false*, and the algorithm terminates. Predicate p yields *true* if the Boolean results of the evaluated logical expressions of *all* 9IMCs coincide with the corresponding Boolean values in p's intersection matrix. This requires constant time.

Predicate determination also depends on a particular combination of spatial data types and leverages nine-intersection matrices and topological feature vectors. In a first step, depending on the spatial data type combination and by means of v_F and v_G, we evaluate the logical expressions on all right sides of the 9IMCs according to Lemma 1, 2, 3, 4, 5, or Theorem 1, respectively. This yields a Boolean nine-intersection matrix. In a second step, this Boolean matrix is checked consecutively for equality against all nine-intersection matrices of the topological predicates of

the particular type combination. If $n_{\alpha,\beta}$ with $\alpha, \beta \in \{point, line, region\}$ is the number of topological predicates between the types α and β, this requires $n_{\alpha,\beta}$ tests in the worst case.

4.5 Evaluation of dimension-refined topological predicates

The 9IMC described so far is applicable to the evaluation of proper topological predicates. In this subsection, we show that, with a few extensions, the 9IMC can also evaluate *dimension-refined* topological predicates (Section 2.3) for which the dimension of an intersection of two spatial objects plays an essential role. That is, the exploration algorithms provide us with enough topological information to perform this task. In this sense, dimension-refined topological predicates are a refinement of proper topological predicates. For example, in a meeting situation, we could be interested in a query whether two *line* objects meet in a zero-dimensional object (i.e., a *point* object), or in a one-dimensional object (i.e., a *line* object), or in both. Dimension refinement is only possible for the type combinations *line*/*line*, *line*/*region*, and *region*/*region*. We distinguish the different dimensions of the maximal connected components of a given point set in the two-dimensional space and define a "dimension type" as follows [26]:

$$dimType = \{\bot, 0D, 1D, 2D, 01D, 02D, 12D, 012D\}$$

This type allows us to represent the dimension of a given point set as the "union" of the dimensions of its maximal connected components. If a point set consists only of single points, only of lines, or only of regions, the corresponding value of type *dimType* is $0D$, $1D$, or $2D$ respectively. If a point set contains maximal connected components of different dimensions, the corresponding value is $01D$, $02D$, $12D$, and $012D$ respectively. The $\bot$ symbol is used to represent the undefined dimension of an empty point set.

In [26], we have shown that the dimension of the intersection of a zero-, one-, or two-dimensional spatial component with another zero-, one-, or two-dimensional spatial component can only have the value $\bot$, $0D$, $1D$, $2D$, or $01D$. Further, only in case of two one-dimensional components, their intersection may lead to components of different dimensions. This refers exclusively to the intersection between the interiors of two *line* objects, between the interior of a *line* object and the boundary of a *region* object, or between the boundaries of two *region* objects. The possible dimension values for these three cases are $\bot$, $0D$, $1D$, or $01D$. They are evaluated or determined by our extended 9IMC method described now.

Let $F, G \in \alpha \in \{line, region\}$, p be a topological predicate between F and G, and p_d be a dimension-refined predicate that is a refinement of p. We leverage the following relationship between p and p_d:

$$(p_d(F, G) \Rightarrow p(F, G)) \Leftrightarrow (\neg p(F, G) \Rightarrow \neg p_d(F, G))$$

For the predicate verification of p_d, we first apply the 9IMC for the predicate verification of p. From the right side of the equivalence, we conclude that, if $p(F, G)$ yields *false*, $p_d(F, G)$ must yield *false* too. Otherwise, if $p(F, G)$ yields *true*, we cannot make a statement about p_d because the satisfaction of p is only a necessary but not sufficient condition for the satisfaction of p_d. In this case, we need additional *dimension characterizations* that are given in Table 1. For each type combination,

Table 1 Dimension characterizations for dimension-refined topological predicates

Type combination	Intersection	Dimension	Dimension characterization
line × *line*	$F^\circ \cap G^\circ = \varnothing$	$\bot$	$\neg v_F[seg\_shared] \wedge \neg v_F[interior\_poi\_shared]$
	$F^\circ \cap G^\circ \neq \varnothing$	$0D$	$\neg v_F[seg\_shared] \wedge v_F[interior\_poi\_shared]$
		$01D$	$v_F[seg\_shared] \wedge v_F[interior\_poi\_shared]$
		$1D$	$v_F[seg\_shared] \wedge \neg v_F[interior\_poi\_shared]$
line × *region*	$F^\circ \cap \partial G = \varnothing$	$\bot$	$\neg v_F[seg\_shared] \wedge \neg v_F[poi\_shared]$
	$F^\circ \cap \partial G \neq \varnothing$	$0D$	$\neg v_F[seg\_shared] \wedge v_F[poi\_shared]$
		$01D$	$v_F[seg\_shared] \wedge v_F[poi\_shared]$
		$1D$	$v_F[seg\_shared] \wedge \neg v_F[poi\_shared]$
region × *region*	$\partial F \cap \partial G = \varnothing$	$\bot$	$\neg(v_F[(0/2)] \vee v_F[(2/0)] \vee v_F[(1/1)] \vee v_F[(bound\_poi\_shared)])$
	$\partial F \cap \partial G \neq \varnothing$	$0D$	$\neg(v_F[(0/2)] \vee v_F[(2/0)] \vee v_F[(1/1)]) \wedge v_F[(bound\_poi\_shared)]$
		$01D$	$(v_F[(0/2)] \vee v_F[(2/0)] \vee v_F[(1/1)]) \wedge v_F[(bound\_poi\_shared)]$
		$1D$	$(v_F[(0/2)] \vee v_F[(2/0)] \vee v_F[(1/1)]) \wedge \neg v_F[(bound\_poi\_shared)]$

the table shows the relevant intersection between the one-dimensional components of these types, the possible dimensions such an intersection can have, and for each dimension its characterization on the basis of the topological feature vectors of this type combination. Since the dimension characterizations are different for each type combination, they are unique. Therefore, we only have to look up the type combination and the dimension of interest, evaluate the corresponding dimension characterization, and obtain its Boolean value as the result for the dimension-refined predicate.

For the predicate determination of the matching dimension-refined topological predicate p_d between F and G, we first apply the 9IMC for the predicate determination of the matching topological predicate p between F and G. Afterwards, we consecutively evaluate the four dimension characterizations of the type combination under consideration from Table 1. If the resulting dimension is the $\bot$ element, the dimension-refined predicate p_d coincides with p. Otherwise, we obtain either $p_d = 0D\text{-}p$, $p_d = 01D\text{-}p$, or $p_d = 1D\text{-}p$.

5 Optimized evaluation methods

Based on the exploration phase and leveraging the nine-intersection matrix characterization, we have found a universal, correct, complete, and effective method for both predicate verification and predicate determination of proper and dimension-refined topological predicates. So far, we have focused on the general applicability and universality of our overall approach. In this section, we show that it is even possible to fine-tune and thus improve our 9IMC approach with respect to efficiency if we look at predicate verification and predicate determination separately. Section 5.1 delineates a novel method called *matrix thinning* for speeding up predicate verification. Section 5.2 describes a fine-tuned method called *minimum cost decision tree* for accelerating predicate determination.

5.1 Matrix thinning for predicate verification

The approach of *matrix thinning* (*MT*) described in this subsection is based on the observation that for predicate verification only a subset of the nine matrix predicates has to be evaluated in order to determine the validity of a given topological relationship between two spatial objects F and G. For example, for the predicate 1 (*disjoint*) of the *region*/*region* case, the combination that $F^\circ \cap G^\circ = \varnothing \ \wedge \ \partial F \cap \partial G = \varnothing$ holds (indicated by two 0's) is unique among the 33 predicates. Consequently, only these two matrix predicates have to be tested in order to decide about *true* or *false* of this topological predicate.

The question arises how the nine-intersection matrices can be systematically "thinned out" and nevertheless remain unique among the $n_{\alpha,\beta}$ topological predicates between two spatial data types α and β. We use a brute-force algorithm (Fig. 6) that is applicable to all type combinations and that determines the thinned out version of each intersection matrix associated with one of the $n_{\alpha,\beta}$ topological predicates. Since this algorithm only has to be executed once for each type combination, runtime performance and space efficiency are not so important here.

In a first step (lines 8 to 10), we create a matrix *pos* of so-called *position matrices* corresponding to all possible nine-intersection matrices, i.e., to the binary versions of the decimal numbers 1 to 511 if we read the nine-intersection matrix (9IM) entries row by row. Each "1" in a position matrix indicates a position or entry that is later used for checking two intersection matrices against each other. A "0" in a position matrix means that the corresponding entries in two compared intersection matrices are not compared and hence ignored.

Because our goal is to minimize the number of matrix predicates that have to be evaluated, in a second step, we sort the position matrices with respect to the number of ones in increasing order (lines 11 to 12). That is, the list of position matrices will first contain all matrices with a single "1", then the matrices with two ones, etc., until we reach the matrix with nine ones. At the latest here, it is guaranteed that an intersection matrix is different to all the other $n_{\alpha,\beta} - 1$ intersection matrices. Hence, our algorithm terminates.

```
01 algorithm MatrixThinning
02 input:  Three-dimensional 9IM im. im[i, l, m] ∈ {0, 1}
03         denotes entry (l, m) (1 ≤ l, m ≤ 3) of the ith
04         9IM (1 ≤ i ≤ n_{α,β}).
05 output: Three-dimensional thinned out 9IM tim.
06         tim[i, l, m] ∈ {0, 1, ∗}. '∗' is 'don't care' symbol.
07 begin
08   Create three-dimensional matrix pos of 'position'
09     matrices where pos[j, l, m] ∈ {0, 1} denotes entry
10     (l, m) of the jth possible 9IM (1 ≤ j ≤ 511);
11   Sort pos increasingly with respect to the number of
12     ones in a matrix;
13   Initialize all entries of matrices of tim with '∗'; r := 1;
14   // Compute thinned out matrices
15   for each i in 1 . . . n_{α,β} do
16     j := 1; stop := false;
17     while j ≤ 511 and not stop do
18       k := 1; unequal := true;
19       while 1 ≤ k ≤ n_{α,β} and i ≠ k and unequal do
20         equal := im[i] and im[k] have the same values
21           at all positions (l, m) where pos[j, l, m] = 1;
22         unequal := unequal and not equal; inc(k);
23       endwhile;
24       if unequal then // Thin out im[i] by pos[j].
25         for each l, m in 1 . . . 3 do
26           if pos[j, l,
27           then tim[r, l, m] := im[i, l, m] endif
28         endfor;
29         inc(r); stop := true;
30       else inc(j);
31       endif
32     endwhile
33   endfor
34 end MatrixThinning.
```

Fig. 6 Algorithm for computing the thinned out versions of the $n_{\alpha,\beta}$ intersection matrices associated with the topological predicates between two spatial data types α and β

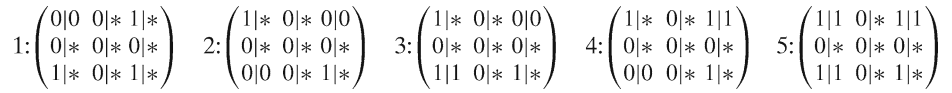

Fig. 7 Complete and thinned out matrices for the five topological predicates of the *point*/*point* case

In a third step, we initialize the entries of all $n_{\alpha,\beta}$ thinned out intersection matrices with the "don't care" symbol "$*$".

The fourth and final step computes the thinned out matrices (lines 15 to 33). The idea is to find for each intersection matrix (line 15) a minimal number of entries that together uniquely differ from the corresponding entries of all the other $n_{\alpha,\beta} - 1$ intersection matrices. Therefore, we start traversing the 511 position matrices (line 17). For all "1"-positions of a position matrix we find out whether for the intersection matrix under consideration another intersection matrix exists that has the same matrix values at these positions (lines 20 to 21). As long as no equality has been found, the intersection matrix under consideration is compared to the next intersection matrix (lines 19 to 23). If an equality is found, the next position matrix is taken (line 30). Otherwise, we have found a minimal number of matrix predicates that are sufficient and unique for evaluation (line 24). It remains to copy the corresponding values of the nine-intersection matrix into the thinned out matrix (lines 25 to 28).

Note that for the same intersection matrix it may be possible to find several thinned out matrices with the same number of matrix predicates to be checked such that each of them represents the intersection matrix uniquely among the $n_{\alpha,\beta}$ intersection matrices. Our algorithm always computes the thinned out matrix with the "lowest numerical value". The complete and thinned out matrices for the *point*/*point* case are shown in Fig. 7, for the *point*/*line* case in Fig. 8, for the *point*/*region* case in Fig. 9, and for the *region*/*region* case in Fig. 10. The complete and thinned out matrices for the *line*/*line* case and the *line*/*region* case can be found in [30]. Definition 7 defines the measures we use to summarize and interpret these results.

Definition 7 Let IM^{MT} be a thinned out 9IM, and *cnt* be a function that counts the number of relevant matrix predicates of IM^{MT}. Let $n_{\alpha,\beta}$ with $\alpha, \beta \in \{point, line, region\}$ be the number of (thinned out) 9IMs of the topological predicates

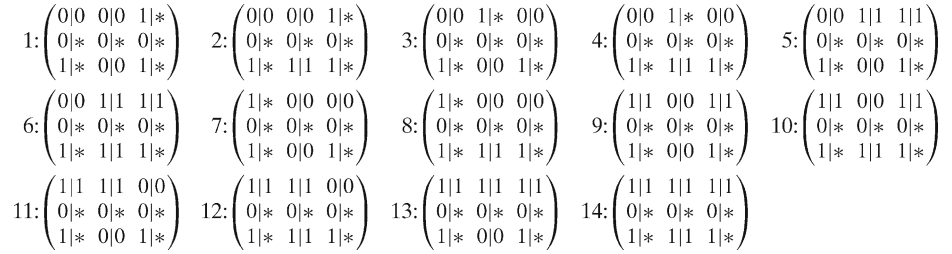

Fig. 8 Complete and thinned out matrices for the 14 topological predicates of the *point*/*line* case

$$1:\begin{pmatrix} 0|0 & 0|0 & 1|* \\ 0|* & 0|* & 0|* \\ 1|* & 1|* & 1|* \end{pmatrix} \quad 2:\begin{pmatrix} 0|0 & 1|* & 0|0 \\ 0|* & 0|* & 0|* \\ 1|* & 1|* & 1|* \end{pmatrix} \quad 3:\begin{pmatrix} 0|0 & 1|1 & 1|1 \\ 0|* & 0|* & 0|* \\ 1|* & 1|* & 1|* \end{pmatrix} \quad 4:\begin{pmatrix} 1|* & 0|0 & 0|0 \\ 0|* & 0|* & 0|* \\ 1|* & 1|* & 1|* \end{pmatrix} \quad 5:\begin{pmatrix} 1|1 & 0|0 & 1|1 \\ 0|* & 0|* & 0|* \\ 1|* & 1|* & 1|* \end{pmatrix}$$

$$6:\begin{pmatrix} 1|1 & 1|1 & 0|0 \\ 0|* & 0|* & 0|* \\ 1|* & 1|* & 1|* \end{pmatrix} \quad 7:\begin{pmatrix} 1|1 & 1|1 & 1|1 \\ 0|* & 0|* & 0|* \\ 1|* & 1|* & 1|* \end{pmatrix}$$

Fig. 9 Complete and thinned out matrices for the seven topological predicates of the *point/region* case

between the types α and β, and $n^k_{\alpha,\beta}$ be the number of thinned out 9IMs for which k (with $1 \leq k \leq 9$) matrix predicates have to be evaluated. Let the cost, i.e., the total number of matrix predicates to be evaluated for α and β, be $C_{\alpha,\beta}$ without matrix thinning and $C^{MT}_{\alpha,\beta}$ with matrix thinning. We then denote with $RAC^{MT}_{\alpha,\beta}$ the reduced average cost in percent when using matrix thinning. We obtain:

1. $cnt(IM^{MT}) = |\{(l,m) \mid 1 \leq l,m \leq 3, IM^{MT}[l,m] \in \{0,1\}\}|$
2. $n^k_{\alpha,\beta} = \left|\left\{IM^{MT}_i \mid 1 \leq i \leq n_{\alpha,\beta}, 1 \leq k \leq 9, cnt(IM^{MT}_i) = k\right\}\right|$
3. $n_{\alpha,\beta} = \sum_{k=1}^{9} n^k_{\alpha,\beta}$
4. $C_{\alpha,\beta} = 8 \cdot n_{\alpha,\beta}$
5. $AC_{\alpha,\beta} = C_{\alpha,\beta}/n_{\alpha,\beta} = 8$
6. $C^{MT}_{\alpha,\beta} = \sum_{k=1}^{9} k \cdot n^k_{\alpha,\beta}$
7. $AC^{MT}_{\alpha,\beta} = C^{MT}_{\alpha,\beta}/n_{\alpha,\beta}$
8. $RAC^{MT}_{\alpha,\beta} = 100 \cdot AC^{MT}_{\alpha,\beta}/AC_{\alpha,\beta} = 100 \cdot C^{MT}_{\alpha,\beta}/C_{\alpha,\beta}$

$$1:\begin{pmatrix} 0|0 & 0|* & 1|* \\ 0|* & 0|0 & 1|* \\ 1|* & 1|* & 1|* \end{pmatrix} \quad 2:\begin{pmatrix} 0|0 & 0|* & 1|* \\ 0|* & 1|* & 0|0 \\ 1|* & 1|* & 1|* \end{pmatrix} \quad 3:\begin{pmatrix} 0|0 & 0|* & 1|* \\ 0|* & 1|* & 1|* \\ 1|* & 0|0 & 1|* \end{pmatrix} \quad 4:\begin{pmatrix} 0|0 & 0|* & 1|* \\ 0|* & 1|1 & 1|1 \\ 1|* & 1|1 & 1|* \end{pmatrix} \quad 5:\begin{pmatrix} 1|* & 0|* & 0|0 \\ 0|* & 1|* & 0|* \\ 0|0 & 0|* & 1|* \end{pmatrix}$$

$$6:\begin{pmatrix} 1|* & 0|* & 0|0 \\ 0|0 & 1|* & 0|* \\ 1|* & 1|1 & 1|* \end{pmatrix} \quad 7:\begin{pmatrix} 1|* & 0|* & 0|0 \\ 1|* & 0|0 & 0|* \\ 1|* & 1|* & 1|* \end{pmatrix} \quad 8:\begin{pmatrix} 1|* & 0|* & 0|0 \\ 1|* & 1|* & 0|* \\ 1|1 & 0|0 & 1|* \end{pmatrix} \quad 9:\begin{pmatrix} 1|* & 0|* & 0|0 \\ 1|1 & 1|1 & 0|* \\ 1|* & 1|1 & 1|* \end{pmatrix} \quad 10:\begin{pmatrix} 1|1 & 0|0 & 1|1 \\ 0|0 & 1|* & 0|0 \\ 1|* & 1|* & 1|* \end{pmatrix}$$

$$11:\begin{pmatrix} 1|* & 0|0 & 1|* \\ 0|* & 1|* & 1|1 \\ 0|0 & 0|* & 1|* \end{pmatrix} \quad 12:\begin{pmatrix} 1|1 & 0|0 & 1|* \\ 0|0 & 1|* & 1|* \\ 1|1 & 0|0 & 1|* \end{pmatrix} \quad 13:\begin{pmatrix} 1|1 & 0|0 & 1|* \\ 0|0 & 1|* & 1|1 \\ 1|* & 1|1 & 1|* \end{pmatrix} \quad 14:\begin{pmatrix} 1|* & 0|0 & 1|* \\ 1|1 & 0|0 & 1|1 \\ 1|* & 1|* & 1|* \end{pmatrix} \quad 15:\begin{pmatrix} 1|* & 0|0 & 1|1 \\ 1|* & 1|* & 0|0 \\ 1|* & 0|0 & 1|* \end{pmatrix}$$

$$16:\begin{pmatrix} 1|* & 0|0 & 1|1 \\ 1|1 & 1|* & 0|0 \\ 1|* & 1|1 & 1|* \end{pmatrix} \quad 17:\begin{pmatrix} 1|* & 0|0 & 1|* \\ 1|1 & 1|* & 1|1 \\ 1|* & 0|0 & 1|* \end{pmatrix} \quad 18:\begin{pmatrix} 1|* & 0|0 & 1|* \\ 1|1 & 1|1 & 1|1 \\ 1|* & 1|1 & 1|* \end{pmatrix} \quad 19:\begin{pmatrix} 1|* & 1|* & 1|* \\ 0|* & 0|0 & 1|* \\ 0|0 & 0|* & 1|* \end{pmatrix} \quad 20:\begin{pmatrix} 1|* & 1|1 & 1|* \\ 0|0 & 0|0 & 1|* \\ 1|* & 1|1 & 1|* \end{pmatrix}$$

$$21:\begin{pmatrix} 1|* & 1|* & 1|1 \\ 0|* & 1|* & 0|0 \\ 0|0 & 0|* & 1|* \end{pmatrix} \quad 22:\begin{pmatrix} 1|* & 1|* & 1|* \\ 0|0 & 1|* & 0|0 \\ 1|1 & 0|0 & 1|* \end{pmatrix} \quad 23:\begin{pmatrix} 1|* & 1|1 & 1|* \\ 0|0 & 1|* & 0|0 \\ 1|* & 1|1 & 1|* \end{pmatrix} \quad 24:\begin{pmatrix} 1|* & 1|1 & 1|* \\ 0|* & 1|1 & 1|1 \\ 0|0 & 0|* & 1|* \end{pmatrix} \quad 25:\begin{pmatrix} 1|* & 1|1 & 1|* \\ 0|0 & 1|* & 1|1 \\ 1|1 & 0|0 & 1|* \end{pmatrix}$$

$$26:\begin{pmatrix} 1|* & 1|1 & 1|* \\ 0|0 & 1|1 & 1|1 \\ 1|* & 1|1 & 1|* \end{pmatrix} \quad 27:\begin{pmatrix} 1|* & 1|* & 1|1 \\ 1|* & 0|0 & 0|0 \\ 1|* & 1|* & 1|* \end{pmatrix} \quad 28:\begin{pmatrix} 1|* & 1|* & 1|* \\ 1|* & 0|0 & 1|* \\ 1|1 & 0|0 & 1|* \end{pmatrix} \quad 29:\begin{pmatrix} 1|* & 1|1 & 1|* \\ 1|1 & 0|0 & 1|1 \\ 1|* & 1|1 & 1|* \end{pmatrix} \quad 30:\begin{pmatrix} 1|* & 1|1 & 1|* \\ 1|1 & 1|* & 0|0 \\ 1|* & 0|0 & 1|* \end{pmatrix}$$

$$31:\begin{pmatrix} 1|* & 1|1 & 1|* \\ 1|1 & 1|1 & 0|0 \\ 1|* & 1|1 & 1|* \end{pmatrix} \quad 32:\begin{pmatrix} 1|* & 1|1 & 1|* \\ 1|1 & 1|1 & 1|1 \\ 1|* & 0|0 & 1|* \end{pmatrix} \quad 33:\begin{pmatrix} 1|* & 1|1 & 1|* \\ 1|1 & 1|1 & 1|1 \\ 1|* & 1|1 & 1|* \end{pmatrix}$$

Fig. 10 Complete and thinned out matrices for the 33 topological predicates of the *region/region* case

Table 2 Summary of complete and thinned out 9IMs for the topological predicates of all type combinations

Type combination	$n_{\alpha,\beta}$	$n^k_{\alpha,\beta}$ with $k =$									$C_{\alpha,\beta}$	$AC_{\alpha,\beta}$	$C^{MT}_{\alpha,\beta}$	$AC^{MT}_{\alpha,\beta}$	$RAC^{MT}_{\alpha,\beta}$
		1	2	3	4	5	6	7	8	9					
point / point	5	1	3	1	0	0	0	0	0	0	40	8	10	2.00	25.00
line / line	82	0	0	2	12	4	50	12	2	0	656	8	474	5.78	72.26
region / region	33	0	6	6	10	11	0	0	0	0	264	8	125	3.79	47.35
point / line	14	0	0	6	8	0	0	0	0	0	112	8	50	3.57	44.64
point / region	7	0	3	4	0	0	0	0	0	0	56	8	18	2.57	32.14
line / region	43	0	0	5	18	12	7	1	0	0	344	8	196	4.56	56.98

$AC^{MT}_{\alpha,\beta}$ denotes the average number of matrix predicates to be evaluated. Table 2 shows a summary of the results and in the last two columns the considerable performance enhancement of matrix thinning. The reduction of matrix predicate computations ranges from 27% for the *line*/*line* case to 75% for the *point*/*point* case.

5.2 The minimum cost decision tree for predicate determination

In Section 4, we have seen that, in the worst case, $n_{\alpha,\beta}$ matching tests are needed to determine the topological relationship between any two spatial objects. For each test, Boolean expressions have to be evaluated that are equivalent to the eight matrix predicates and based on topological feature vectors. We propose two methods to improve the performance. The first method reduces the number of matrix predicates to be evaluated. This goal can be directly achieved by applying the method of matrix thinning described in Section 5.1. That is, the number $n_{\alpha,\beta}$ of tests remains the same but for each test we can reduce the number of matrix predicates that have to be evaluated by taking the thinned out instead of the complete nine-intersection matrices.

The second method, which will be our focus in this subsection, aims at reducing the number $n_{\alpha,\beta}$ of tests. This method is based on the complete nine-intersection matrices but also manages to reduce the number of matrix predicates that have to be evaluated. We propose a *global* concept called *minimum cost decision tree* (*MCDT*) for this purpose. The term "global" means that we do not look at each intersection matrix individually but consider all $n_{\alpha,\beta}$ intersection matrices together. The idea is to construct a *full binary decision tree* whose inner nodes represent all matrix predicates, whose edges represent the Boolean values *true* or *false*, and whose leaf nodes are the $n_{\alpha,\beta}$ topological predicates. Note that, in a full binary tree, each node has exactly zero or two children. For searching, we employ a depth-first search procedure that starts at the root of the tree and proceeds down to one of the leaves which represents the matching topological predicate. The performance gain through the use of a decision tree is significant since the tree partitions the search space at each node and gradually excludes more and more topological predicates. In the best case, at each node of the decision tree, the search space, which comprises the remaining topological predicates to be assigned to the remaining leaves of the node's subtree, is partitioned into two halves so that we obtain a perfectly balanced tree. This would guarantee a search time of $O(\log n_{\alpha,\beta})$. But in general, we cannot expect to obtain a bisection of topological

predicates at each node since the number of topological predicates yielding *true* for the node's matrix predicate will be different from the number of topological predicates yielding *false* for that matrix predicate. An upper bound is the number 8, since at most eight matrix predicates have to be checked to identify a topological predicate uniquely; the ninth matrix predicate yields always *true*. Hence, our goal is to determine a nearly balanced, cost-optimal, full binary decision tree for each collection of $n_{\alpha,\beta}$ intersection matrices.

If we do not have specific knowledge about the probability distribution of topological predicates in an application (area), we can only assume that they occur with equal distribution. But sometimes we have more detailed information. For example, in cadastral map applications, an adequate estimate is that 95% (or even more) of all topological relationships between regions are *disjoint* and the remaining 5% are *meet*. Our algorithm for constructing MCDTs considers these frequency distributions. It is based on the following cost model:

Definition 8 Let $M_{\alpha,\beta}$ be an MCDT for the spatial data types $\alpha, \beta \in \{point, line, region\}$, w_i be the weight of the topological predicate p_i with $1 \leq i \leq n_{\alpha,\beta}$ and $0 < w_i < 1$, and d_i with $1 \leq d_i \leq 8$ be the depth of a node in $M_{\alpha,\beta}$ at which p_i is identified. We define the total cost $C_{\alpha,\beta}^{MCDT}$ of $M_{\alpha,\beta}$ as

$$C_{\alpha,\beta}^{MCDT} = \sum_{i=1}^{n_{\alpha,\beta}} w_i \cdot d_i \qquad \text{with} \sum_{i=1}^{n_{\alpha,\beta}} w_i = 1$$

That is, our cost model is to sum up all the weighted path lengths from each leaf node representing a topological predicate to the root of the MCDT. If all topological predicates occur with equal probability, we set $w_i = \frac{1}{n_{\alpha,\beta}}$. The issue now is how to find and build an optimal MCDT with minimal total cost $C_{\alpha,\beta}^{MCDT}$ on the basis of a given probability distribution (weighting) for the topological predicates. If all topological predicates occur with equal probability, this problem corresponds to finding an optimal MCDT that requires the minimal averaqe number of matrix predicate evaluations to arrive at an answer.

Figure 11 shows our recursive algorithm *MCDT* for computing a minimum cost decision tree for a set *im* of $n_{\alpha,\beta}$ nine-intersection matrices that are annotated with

```
01 algorithm MCDT
02 input:   list im = ⟨(im_1, w_1), ..., (im_{n_{α,β}}, w_{n_{α,β}})⟩ of 9IMs
03          with weights, list mp of the eight matrix predicates
04 output:  MCDT M_{α,β}
05 begin
06   best_node := new_node(); stop := false;
07   discriminator := select_first(mp);
08   while not eol(mp) and not stop do
09     node := new_node();
10     node.discr := discriminator; node.im := im;
11     if no_of_elem(im) = 1 then /* leaf node */
12       best_node := node; best_node.cost := 0;
13       stop := true;
14     else
15       /* Let im = ⟨(im_{k_1}, w_{k_1}), ..., (im_{k_n}, w_{k_n})⟩
16          with 1 ≤ k_1 ≤ ... ≤ k_n ≤ n_{α,β}. */
17       partition(im, discriminator, im_l, im_r);
18       if no_of_elem(im_l) ≠ 0 and
19          no_of_elem(im_r) ≠ 0 then
20         copy(mp, new_mp); del(new_mp, discriminator);
21         node.lchild := MCDT(im_l, new_mp);
22         node.rchild := MCDT(im_r, new_mp);
23         node.cost := node.lchild.cost + node.rchild.cost
24                      + Σ_{i=k_1}^{k_n} w_i;
25         if node.cost < best_node.cost
26         then best_node := node; endif;
27       endif;
28       discriminator := select_next(mp);
29     endif
30   endwhile;
31   return best_node;
32 end MCDT.
```

Fig. 11 Minimum cost decision tree algorithm

a weight representing the corresponding predicates's probability of occurrence, as it is characteristic in a particular application (line 2). Later these matrices become the leaves of the decision tree. In addition, the algorithm takes as an input the list *mp* of eight matrix predicates (we remember that the exterior/exterior intersection yields always *true*) that serve as discriminators and are attached to the inner nodes (line 3). This list is necessary to ensure that a matrix predicate is not used more than once as a discriminator in a search path. During the recursion, the while-loop (lines 8 to 30) terminates if either the list *mp* of matrix predicates to be processed is empty or the list *im* of nine-intersection matrices contains only a single element. For each matrix predicate used as a discriminator, the operation *new_node* creates a new tree node *node* (line 9). The matrix predicate *discriminator* as well as the list *im* annotate the tree node *node* (line 10). If *im* has only one element (line 11), we know that *node* is a leaf node representing the topological predicate pertaining to the single element in *im*. The cost for this leaf node is 0 since its current depth is 0 (line 12). Otherwise, if *im* consists of more than one element, we partition it into two lists im_l and im_r (line 17). The partitioning is based on the values of each nine-intersection matrix in *im* with respect to the matrix predicate serving as the discriminator. If such a value is 0 (*false*), the corresponding nine-intersection matrix is added to the list im_l; otherwise, it is added to the list im_r. A special case now is that *im* has not been partitioned so that either im_l or im_r is empty (condition in lines 18 to 19 yields *false*). In this case, the discriminator does not contribute to a decision and is skipped; the next discriminator is selected (line 28). If both lists im_l and im_r are nonempty (lines 18 to 19), we remove the discriminator from a new copy *new_mp* of the list *mp* (line 20) and recursively find the minimum cost decision trees for the nine-intersection matrices in im_l (line 21) and in im_r (line 22). Eventually, all recursions will reach all leaf nodes and begin returning while recursively calculating the cost of each subtree found. The cost of a leaf node is 0. The cost of an inner node *node* can be expressed in terms of the cost of its two nonempty subtrees *node.lchild* and *node.rchild* processing the lists im_l and im_r respectively. The depth of each leaf node with respect to *node* is exactly one larger than the depth of the same leaf node with respect to either *node.lchild* or *node.rchild*. Therefore, besides the costs of these two subtrees, for each leaf node of the subtree with root *node*, we have to add the leaf node's cost (weight) one time (lines 23 to 24). These weights are stored in *node.im*. The cost of *node* is then compared with the best cost determined so far, and the minimum will be the new best option (lines 25 to 26). Eventually, when all the matrix predicates have been considered, we obtain the best choice and return the corresponding minimum cost decision tree (line 31).

Table 3 shows the results of this algorithm by giving a textual pre-order (depth-first search) encoding of all MCDTs for all type combinations on the basis of equal probability of occurrence of all topological predicates. The encodings allow an easy construction of the MCDTs. Since MCDTs are full binary trees, each node has exactly zero or two child nodes. We leverage this feature by representing an MCDT as the result of a pre-order tree traversal. The pre-order sequence of nodes is unique in this case for constructing the decision tree since inner nodes with only one child node cannot occur. Each inner node in the pre-order sequence is described as a term XY where $X, Y \in \{^\circ, \partial, ^-\}$. Such a term represents a matrix predicate $AX \cap BY \neq \varnothing$ serving as a discriminator. For example, the term $XY = ^\circ\, \partial$ denotes the matrix predicate $A^\circ \cap \partial B \neq \varnothing$ (prefix notation for boundary). Each leaf node represents

Table 3 MCDT pre-order representations for all type combinations on the basis of equal probability of occurrence of all topological predicates

Type combination	MCDT preorder representation																			
point / point	∘−	−∘	2	3	∘∘	1	−∘	4	5											
line / line	°∂	∂°	∂∂	∘−	∘−	−∘	33	−∂	34	35	∘∘	−∂	1	2	−∘	43	−∂	44	45	∘∘
	−∂	3	4	−∘	46	−∂	47	48	∘−	∘−	−∘	36	−∂	37	38	∘∘	−∂	5	6	−∘
	49	−∂	50	51	∘∘	−∂	7	8	−∘	52	−∂	53	54	∘∘	∂∂	∘−	−∂	9	10	−∂
	11	12	∘−	−∂	13	14	−∂	15	16	∘−	∂∂	−∂	39	40	−∂	41	42	∂∂	∘−	−∂
	55	56	−∂	57	58	∘−	−∂	59	60	−∂	61	62	∘∘	∂°	∂∂	∘−	−∂	17	18	−∂
	19	20	∘−	−∂	21	22	−∂	23	24	∂∂	∘−	−∂	25	26	−∂	27	28	∘−	−∂	29
	30	−∂	31	32	∂°	∂∂	∘−	−∘	63	−∂	64	65	−∘	66	−∂	67	68	∘−	−∘	69
	−∂	70	71	−∘	72	−∂	73	74	∂∂	∘−	−∂	75	76	−∂	77	78	∘−	−∂	79	80
	−∂	81	82																	
region / region	°∂	∂°	∘−	∘−	−∘	5	6	∘∘	2	10	∘∘	∂∂	1	−∂	3	4	−∘	11	−∂	12
	13	∘−	∂∂	7	−∂	8	9	∘−	−∂	15	16	∂∂	14	−∂	17	18	∂°	∘−	−∘	21
	−∂	22	23	∂∂	−∘	19	20	−∘	24	−∂	25	26	∂∂	∘−	27	−∂	28	29	∘−	−∂
	30	31	−∂	32	33															
point / line	∘∘	°∂	−∂	1	2	∘−	−∂	3	4	−∂	5	6	°∂	∘−	−∂	7	8	−∂	9	10
	∘−	−∂	11	12	−∂	13	14													
point / region	∘∘	°∂	1	∘−	2	3	°∂	∘−	4	5	∘−	6	7							
line / region	∂°	∘∘	∂∂	∘−	∘−	−∂	5	6	°∂	1	−∂	8	9	°∂	2	−∂	10	11	°∂	∘−
	3	4	∘−	7	∘−	12	13	°∂	∘−	∂∂	14	15	∂∂	∘−	18	19	∘−	20	21	∘−
	∂∂	−∂	26	27	28	∂∂	∘−	−∂	32	33	−∂	34	35	∘−	36	37	°∂	∘−	∂∂	16
	17	∂∂	∘−	22	23	∘−	24	25	∘−	∂∂	−∂	29	30	31	∂∂	∘−	−∂	38	39	−∂
	40	41	∘−	42	43															

the nine-intersection matrix number of a topological predicate. The matrix numbers are specified in the Figs. 7, 8, 9 and 10 as well as in [30], [35].

Figures 12 shows a visualization of the MCDTs of three spatial data type combinations on the assumption that all topological predicates occur with equal probability. The MCDTs for the other type combinations have been omitted due to their very large size. Each inner node is annotated with a label XY where $X \in \{A^\circ, \partial A, A^-\}$ and $Y \in \{B^\circ, \partial B, B^-\}$. A label represents a matrix predicate $X \cap Y \neq \varnothing$ serving as a discriminator. For example, the label $XY = A^\circ \partial B$ denotes the matrix predicate $A^\circ \cap \partial B \neq \varnothing$. If the evaluation of a matrix predicate yields *false*, we move to the left child; otherwise, we move to the right child. Each leaf node represents the nine-intersection matrix number of a topological predicate.

The following definition specifies measures that we use to summarize and interpret these results. We are especially interested in the average number of matrix predicates to be evaluated.

Definition 9 Let $C^{MCDT}_{\alpha,\beta}$ denote the total cost of an MCDT $M_{\alpha,\beta}$ according to Definition 8. Let $n_{\alpha,\beta}$ with $\alpha, \beta \in \{point2D, line2D, region2D\}$ be the number of 9IMs of the topological predicates between the types α and β, IM_i with $1 \leq i \leq n_{\alpha,\beta}$ be a 9IM, and $d^k_{\alpha,\beta}$ be the number of topological predicates associated with leaf nodes in $M_{\alpha,\beta}$ of depth k (with $1 \leq k \leq 9$). Further, let $C_{\alpha,\beta}$ be the cost without using an MCDT, $AC_{\alpha,\beta}$ be the average cost without using an MCDT, $AC^{MCDT}_{\alpha,\beta}$ be the average

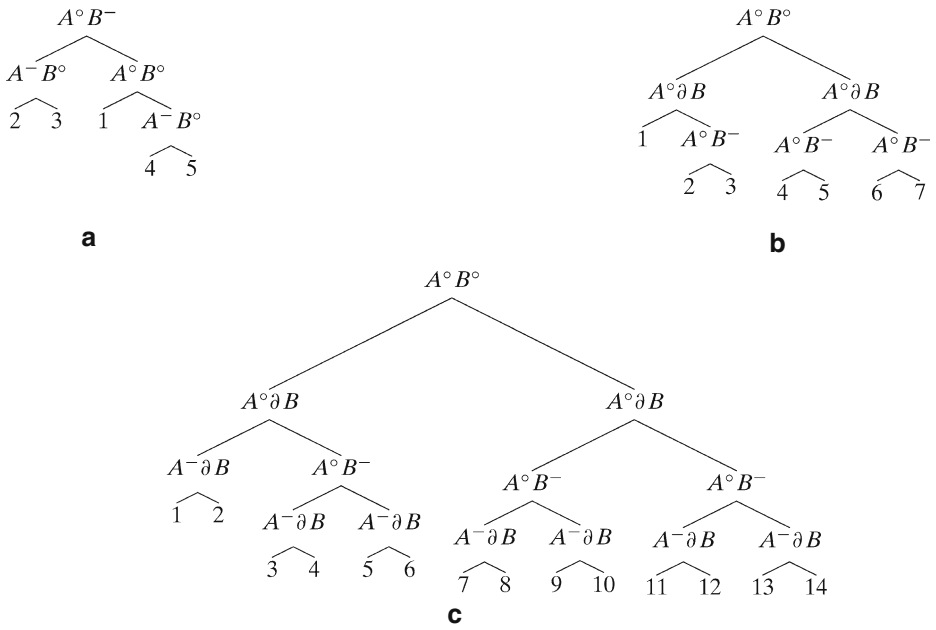

Fig. 12 Minimum cost decision trees for the 5 topological predicates of the point/point case (**a**), the 7 topological predicates of the point/region case (**b**), and the 14 topological predicates of the point/line case (**c**) under the assumption that all topological predicates occur with equal probability

cost when using an MCDT, and $RAC_{\alpha,\beta}^{MCDT}$ be the reduced average cost in percent when using an MCDT. The measures are defined as:

1. $d_{\alpha,\beta}^{k} = |\{IM_i \mid 1 \leq i \leq n_{\alpha,\beta}, 1 \leq k \leq 9, depth(IM_i, M_{\alpha,\beta}) = k\}|$
2. $n_{\alpha,\beta} = \sum_{k=1}^{9} d_{\alpha,\beta}^{k}$
3. $C_{\alpha,\beta} = 8 \cdot n_{\alpha,\beta}$
4. $AC_{\alpha,\beta} = 4 \cdot (n_{\alpha,\beta} + 1)$
5. $AC_{\alpha,\beta}^{MCDT} = C_{\alpha,\beta}^{MCDT} / n_{\alpha,\beta}$
6. $RAC_{\alpha,\beta}^{MCDT} = 100 \cdot AC_{\alpha,\beta}^{MCDT} / AC_{\alpha,\beta}$

To determine the average cost $AC_{\alpha,\beta}$ without using an MCDT in (4), we observe that the best case is to check 8 matrix predicates, the second best case is to check 16 matrix predicates, etc., and the worst case is to check all $8 \cdot n_{\alpha,\beta}$ matrix predicates. The average number of matrix predicates that has to be checked without using an MCDT is therefore $8 \cdot (1 + 2 + \ldots + n_{\alpha,\beta})/n_{\alpha,\beta} = 4 \cdot (n_{\alpha,\beta} + 1)$. $AC_{\alpha,\beta}^{MCDT}$ in (5) yields the average number of matrix predicates to be evaluated. Table 4 shows a summary of the results and in the last two columns the considerable performance enhancement of minimum cost decision trees. The reduction of matrix predicate computations ranges from 90% for the *point*/*point* case to 98% for the *line*/*line* case.

The MCDT approach is similar to a technique introduced in [9] for topological predicates between simple regions. However, their method of determining a binary decision tree rests on the thinned out nine-intersection matrices and results in a near optimal algorithm and solution. The reason why optimality is not achieved is that a topological predicate can have multiple, equipollent thinned out matrices, i.e., thinned out matrices are not unique. Therefore, using a specific set of thinned out matrices as the basis for partitioning the search space can only lead to an optimal decision tree for this set of thinned out matrices and may not be optimal in the general case. Our algorithm rests on the complete nine-intersection matrices. It produces an optimal decision tree (several optimal trees with the same total cost may exist) for the specified set of nine-intersection matrices and the given probability distribution. One can verify this by applying our algorithm to the eight nine-intersection matrices for two simple regions and the same probability distribution as specified in [9]. Our algorithm produces an optimal tree with the total cost of 2.13 while the so-called

Table 4 Summary of the MCDTs for all type combinations on the basis of equal probability of occurrence of all topological predicates

Type combination	$n_{\alpha,\beta}$	$d_{\alpha,\beta}^{k}$ with $k =$									$C_{\alpha,\beta}$	$AC_{\alpha,\beta}$	$C_{\alpha,\beta}^{MCDT}$	$AC_{\alpha,\beta}^{MCDT}$	$RAC_{\alpha,\beta}^{MCDT}$
		1	2	3	4	5	6	7	8	9					
point / *point*	5	0	3	2	0	0	0	0	0	0	40	24	12	2.40	10.00
line / *line*	82	0	0	0	0	0	48	30	4	0	656	332	530	6.46	1.95
region / *region*	33	0	0	0	3	22	8	0	0	0	264	136	170	5.15	3.79
point / *line*	14	0	0	2	12	0	0	0	0	0	112	60	54	3.86	6.43
point / *region*	7	0	1	6	0	0	0	0	0	0	56	32	20	2.86	8.94
line / *region*	43	0	0	0	3	15	19	6	0	0	344	176	243	5.65	3.21

"refined cost method" in [9], which uses thinned out matrices, produces a tree with the total cost of 2.16.

We can observe the following relationship between MCDTs and thinned out matrices:

Lemma 9 *For each combination of spatial data types α and β, the total cost of its minimum cost decision tree* (*given in Table 4*) *is greater than or equal to the total cost of all its thinned out matrices* (*given in Table 2*)*, i.e.,*

$$C^{MCDT}_{\alpha,\beta} \geq C^{MT}_{\alpha,\beta}$$

Proof The proof is given by contradiction. Assume that for a spatial data type combination the total cost of its MCDT is less than the total cost of all its thinned out matrices. Consequently, there must be at least one path from the root to a leaf in the MCDT that contains a smaller number of matrix predicates than the number of matrix predicates in the thinned out matrix for the topological predicate associated with that leaf. This implies that we can identify this topological predicate with a smaller number of matrix predicate decisions than the number of matrix predicates in its thinned out matrix. But this contradicts the definition of a thinned out matrix. □

6 Implementation, testing, approach assessment, and performance study

Section 6.1 describes our implementation approach and environment for the concepts presented in this article. Section 6.2 delineates various tests based on this implementation in order to verify the correctness of the concepts. We perform an overall assessment of our approach from two perspectives. From a qualitative perspective, in Section 6.3, we briefly summarize the benefits of our design decision and compare our approach with an alternative *ad hoc* approach. From a quantitative perspective, in Section 6.4, we conduct a performance study and analyze the results.

6.1 Implementation

To verify the feasibility, practicality, correctness, and efficiency of the concepts presented, we have implemented and tested our approach. In order to have full control about the implementation and the testing of our concepts, we have implemented them in our own algebra package *SPAL2D* (*Spatial Algebra 2D*) for handling two-dimensional spatial data. The implementation makes use of the complex spatial data types *point*, *line*, and *region*, as they have been described in Sections 2.1. Note that all predicate implementation and evaluation concepts presented in this article could also be integrated into commercial and public domain software systems like ESRI's Spatial Database Engine (ArcSDE), the Informix Geodetic DataBlade, Oracle Spatial, DB2's Spatial Extender, and the JTS Topology Suite (see Section 2.1) since the only precondition is the existence of complex spatial data types. The concepts could also be used for implementing the spatial data type specifications of the Open Geospatial Consortium.

SPAL2D provides the three universal interface methods *TopPredExploration*, *TopPredVerification*, and *TopPredDetermination* for providing the functionality of the exploration phase and the evaluation phase. The method *TopPredExploration*

explores the topological data of interest for two interacting spatial objects. This interface is overloaded to accept two spatial objects of any type combination as input. Depending on the input object types, it executes one of the six plane sweep based exploration algorithms from Section 3 and [36]. The output consists of two topological feature vectors which hold the relevant topological information for both argument objects.

The methods *TopPredVerification* and *TopPredDetermination* handle predicate verification and predicate determination queries respectively. Both interfaces are overloaded and take two topological feature vectors as input. Both methods leverage the general evaluation method of nine-intersection matrix characterization from Section 4. The interface method *TopPredVerification* takes a predicate identification number as an additional input parameter. It corresponds to the matrix number (specified in [35] and used in Figs. 7–10 as well as in [30]) of the topological predicate to be evaluated. The method implements the optimized evaluation technique of matrix thinning from Section 5. The output is the Boolean value *true* if the topological relationship between the two spatial objects corresponds to the specified predicate; otherwise, the value is *false*. The interface method *TopPredDetermination* implements the optimized evaluation technique of minimum cost decision trees from Section 5 and outputs the matrix number of the topological predicate corresponding to the topological relationship between the two spatial objects.

6.2 Testing

For testing the results of the exploration phase, our collection of test cases consists of 184 different scenes corresponding to the total number of topological predicates between spatial objects. A special test case generation technique has been leveraged to check the functionality of the exploration algorithms and the correctness of the resulting values of the topological feature vectors. The vector values have to be independent of the location of the two spatial objects involved. In order to check this, this technique is able to generate arbitrarily many different orientations of a topologically identical scene of two spatial objects with respect to the same sweep line and coordinate system. The idea is to rotate such a scene iteratively by a random angle around a central reference point. Special test cases like vertical segments are considered too. For the 184 explicitly constructed base cases, we have generated at least 20,000 test cases for the topological predicates of each of the six type combinations by our random scene rotation technique. In total, more than 120,000 test cases have been successfully generated, tested, and checked for predicate verification and predicate determination and indicate the correctness of our concepts and the ability of our algorithms to correctly discover the needed topological information from any given scene.

For testing the results of the evaluation phase, we take the topological feature vectors as input for the nine-intersection matrix characterization method and the optimization methods of matrix thinning and minimum cost decision trees. The correctness of all methods has been checked by a technique known as *gray-box testing*, which combines the advantages of two other techniques called *black-box testing* and *white-box testing*. The black-box testing technique arranges for well defined input and output objects. In our case, the input consists of two correct

topological feature vectors as well as a matrix number of the topological predicate to be verified in case of predicate verification. This enables us to test the functional behavior of the three method implementations. The output is guaranteed to be either a Boolean value (predicate verification) or a valid matrix number of a topological relationship predefined for the type combination under consideration (predicate determination). The white-box testing technique considers every single execution path and guarantees that each statement is executed at least once. This ensures that all cases that are specified and handled by the algorithms are properly tested.

All cases have been successfully tested and indicate the correctness of our concepts and the ability of our algorithms to correctly verify or determine a topological predicate.

6.3 Qualitative assessment

Although it is usually accepted that some kind of plane-sweep algorithm is sufficient for implementing topological predicates, this article has demonstrated that the decision on an appropriate and sophisticated implementation strategy is of crucial importance. The possible *ad hoc* approach of implementing a separate algorithm for each of the topological predicates results in a large number of algorithms possibly up to the total number of topological predicates of a type combination. Even though this approach is relatively straightforward, it suffers from many problems including large system implementation, non-guaranteed correctness of the algorithms, error-proneness, redundancy, testing and evaluation difficulties, and performance degradation. An essential problem of the *ad hoc* approach is the difficulty in handling predicate determination queries. No particular algorithm is suitable for this task, thus requiring a linear iteration through the large number of algorithms for all topological predicates.

Unlike the *ad hoc* approach, our approach does not suffer from these problems. In our implementation, only a single, generic, and parameterized plane-sweep algorithm is employed for all exploration algorithms. Only a single exploration algorithm is implemented for all topological predicates of each type combination. This implementation strategy allows us to take advantage of significantly smaller system implementation, widespread code reusability and sharing, manageable system testing, and efficient handling of both predicate determination and verification queries. This centralized approach is possible because, instead of considering each topological predicate individually, we look deeper into their common definition blocks which are the nine matrix predicates of the nine-intersection matrix. This leads us to a systematic method. By creating a bidirectional link between the matrix predicates and topological feature vectors, we are able to give a unique characterization for each matrix predicate. This unique characterization frees us from providing algorithms for each topological predicate in case of predicate verification and from evaluating all topological predicates in case of predicate determination. Furthermore, the correctness of the method is formally proven. Last but not least, based on the concept of topological feature vectors, predicate matching techniques such as matrix thinning and minimum cost decision trees can be used to increase the efficiency of answering predicate verification and predicate determination queries respectively.

6.4 Performance study and analysis

Three main reasons impede a comparison of our concepts with available commercial and public domain implementations of topological relationships. First, in Section 2.4 we have seen that all available implementations of topological relationships mainly focus on the eight predicates *disjoint*, *meet*, *overlap*, *equal*, *inside*, *contains*, *covers*, and *coveredBy* that have been generalized to and unified for all combinations of complex spatial data types. Hence, a comparison to our much more fine-grained collection of 184 (248) topological predicates (Section 2.2) is not possible. Second, studying the corresponding documentation, we have not found a formal definition of the semantics of these eight generic predicates for all type combinations. Third, an implementation of our collection of topological predicates and predicate execution techniques is not trivial in the context of commercial implementations since their algorithms and program code are not publicly available and their system environments are very special. For example, the algorithms for the eight topological predicates have not been published.

Instead, we have performed a performance study that underpins the strengths of our approach by quantitatively comparing the performance of our non-optimized alternative (only 9IMC) with our optimized evaluation techniques (9IMC plus matrix

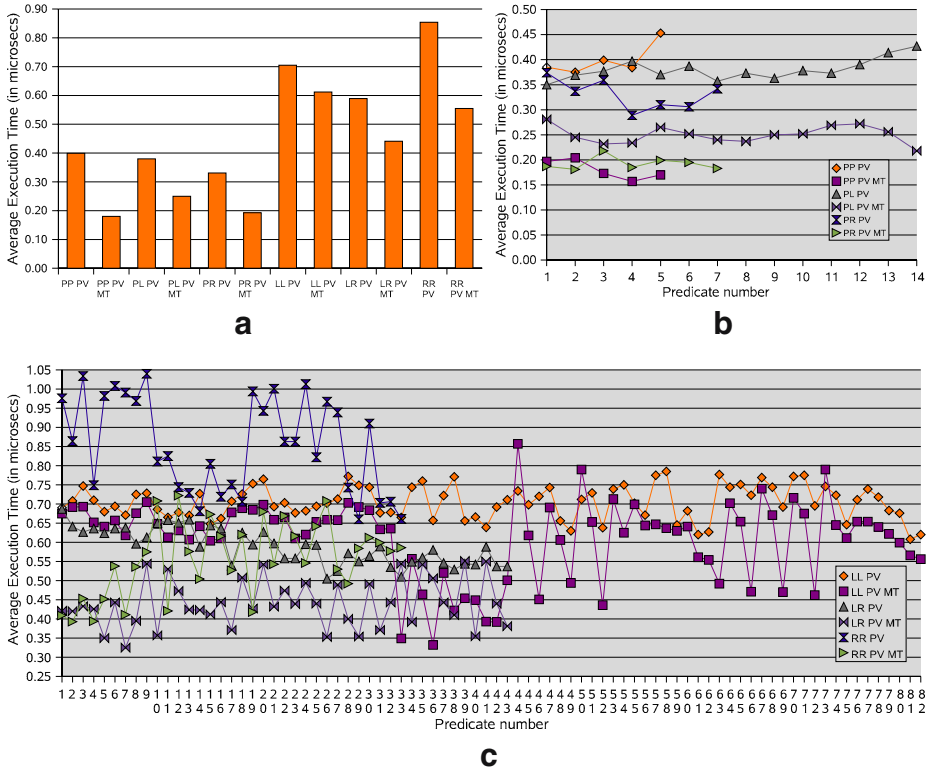

Fig. 13 Predicate verification without and with matrix thinning

thinning, 9IMC plus minimum cost decision tree). Our study shows that our approach does not only provide qualitative (in terms of correctness) but also quantitative benefits by applying optimization methods for the evaluation of topological predicates. For each type combination, we measure and calculate the average execution time for verifying and determining each predicate both without and with optimization. Of course, we cannot expect a time gain of one or more orders of magnitude since a plane sweep is involved in all exploration algorithms, which prevents a better performance [30].

For predicate verification (PV), Fig. 13b and c illustrate the average execution time for each predicate of each type combination without and with matrix thinning (MT). The overall average for each type combination is shown in Fig. 13a. The performance improvements from using matrix thinning are quite noticeable and range from 13% execution time reduction for the *line*/*line* case up to 55% for the *point*/*point* case.

Similarly, for predicate determination (PD), Fig. 14b and c show the average execution time for each predicate of each type combination without and with the use of minimum cost decision trees. The overall average for each type combination is shown in Fig. 14a. The results indicate significant performance improvements from using minimum cost decision trees. The improvements range from 75% execution time reduction for the *point*/*region* case up to 91% for the *line*/*line* case.

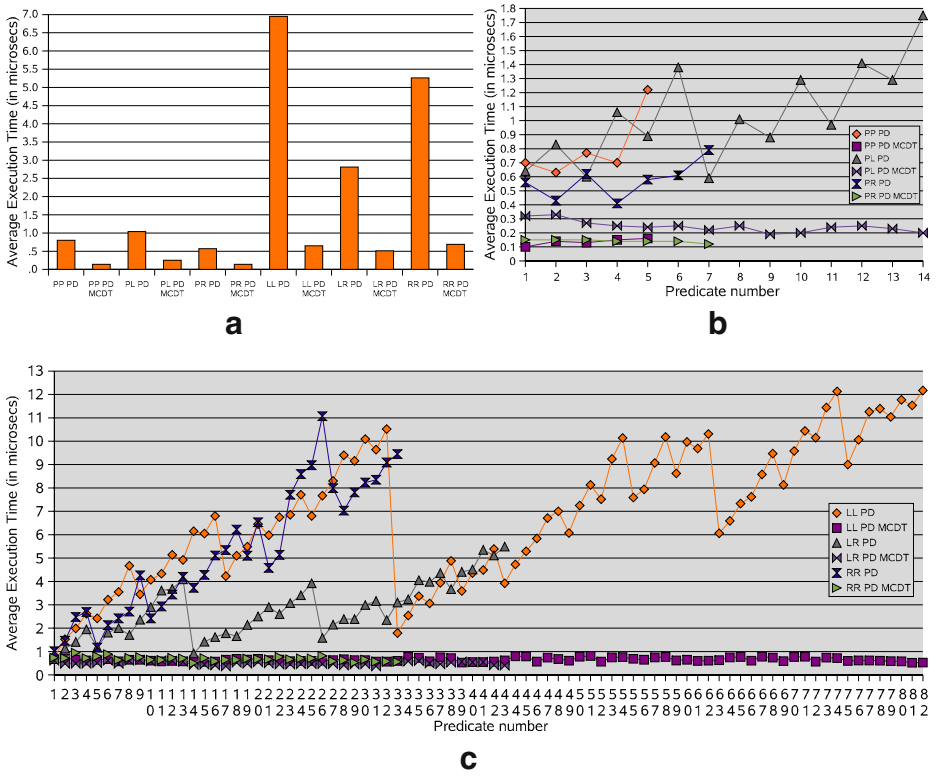

Fig. 14 Predicate determination without and with MCDT

Although the execution time reductions are remarkable for both predicate verification and especially predicate determination and clearly reflect the trend, the empirical results shown in Figs. 13 and 14 are not as optimistic as the computational results given in Tables 2 and 4. The reason that we cannot reach these lower bounds in practice consists in programming and runtime overheads such as extra conditional checks, construction of thinned out matrices and minimum cost decision trees, and their traversals. However, even with these overheads, it is evident that our approach provides considerable performance optimizations.

7 Conclusions

While, from a conceptual perspective, topological predicates between spatial objects have been investigated to a large extent, the design of efficient implementation methods for them has been widely neglected. But an efficient implementation is of essential importance for the processing of queries that contain spatial joins and spatial selections. Especially, the introduction of *complex* spatial data types and the resulting increase of topological predicates between all their combinations require correct and efficient implementation concepts.

We propose a two-phase approach that consists of an *exploration phase* and an *evaluation phase* and that can be applied to both *predicate verification* and *predicate determination*. The goal of the exploration phase is to traverse a given scene of two objects in space, collect any topological information of importance, and store it in *topological feature vectors*. The goal of the evaluation phase is to interpret the gained topological information and match it against the topological predicates by leveraging a general evaluation method called *nine-intersection matrix characterization*. This method generates a formally proven connection between theoretical concepts (nine-intersection matrix, matrix predicate) and implementation concepts (topological feature vector, segment class). It is independent of the number of topological predicates of a particular type combination. Two sophisticated optimization techniques, called *matrix thinning* and *minimum cost decision tree*, are introduced which help us speed up predicate determination and predicate verification respectively. We have also provided an experimental performance study which documents the efficiency of our approach. We have implemented our concepts as part of our SPAL2D software library which is a spatial type system currently under development and determined for an integration into extensible databases.

References

1. M. de Berg, M. van Krefeld, M. Overmars, and O. Schwarzkopf. Computational Geometry: Algorithms and Applications. 2nd edition, Springer-Verlag, 2000.
2. A. Brodsky and X.S. Wang. On Approximation-based Query Evaluation, Expensive Predicates and Constraint Objects. *Int. Workshop on Constraints, Databases, and Logic Programming*, 1995.
3. J. Claussen, A. Kemper, G. Moerkotte, K. Peithner, and M. Steinbrunn. "Optimization and evaluation of disjunctive queries," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 12(2):238–260, 2000.
4. E. Clementini and P. Di Felice. "A comparison of methods for representing topological relationships," *Information Sciences Applications*, Vol. 3(3):149–178, 1995.

5. E. Clementini and P. Di Felice. "A model for representing topological relationships between complex geometric features in spatial databases," *Information Systems*, Vol. 90(1–4):121–136, 1996.
6. E. Clementini and P. Di Felice. "Topological invariants for lines," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 10(1), 1998.
7. E. Clementini, P. Di Felice, and G. Califano. "Composite regions in topological queries," *Information Systems*, Vol. 20(7):579–594, 1995.
8. E. Clementini, P. Di Felice, and P. van Oosterom. "A small set of formal topological relationships suitable for end-user interaction," in *3rd Int. Symp. on Advances in Spatial Databases, LNCS 692*, pp. 277–295, 1993.
9. E. Clementini, J. Sharma, and M.J. Egenhofer. "Modeling topological spatial relations: strategies for query processing," *Computers and Graphics*, Vol. 18(6):815–822, 1994.
10. Z. Cui, A.G. Cohn, and D.A. Randell. "Qualitative and topological relationships," in *3rd Int. Symp. on Advances in Spatial Databases, LNCS 692*, pp. 296–315, 1993.
11. M.J. Egenhofer. "Spatial SQL: A query and presentation language," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 6(1):86–94, 1994.
12. M.J. Egenhofer and J. Herring. "A mathematical framework for the definition of topological relationships," in *4th Int. Symp. on Spatial Data Handling*, pp. 803–813, 1990.
13. M.J. Egenhofer and J. Herring. Categorizing binary topological relations between regions, lines, and points in geographic databases. Technical Report 90-12, National Center for Geographic Information and Analysis, University of California, Santa Barbara, 1990.
14. M.J. Egenhofer and D. Mark. "Modeling conceptual neighborhoods of topological line-region relations," *Int. Journal of Geographical Information Systems*, Vol. 9(5):555–565, 1995.
15. M.J. Egenhofer. "Deriving the composition of binary topological relations," *Journal of Visual Languages and Computing*, Vol. 2(2):133–149, 1994.
16. M.J. Egenhofer, E. Clementini, and P. Di Felice. "Topological relations between regions with holes," *Int. Journal of Geographical Information Systems*, Vol. 8(2):128–142, 1994.
17. ESRI Spatial Database Engine (SDE). Environmental Systems Research Institute, Inc., 1995.
18. S. Gaal. Point Set Topology. Academic Press, 1964.
19. R.H. Güting. "Geo-relational algebra: A model and query language for geometric database systems," in *Int. Conf. on Extending Database Technology*, pp. 506–527, 1988.
20. R.H. Güting and M. Schneider. "Realm-based spatial data types: The ROSE algebra," *VLDB Journal*, Vol. 4:100–143, 1995.
21. J.M. Hellerstein. "Practical predicate placement," in *ACM SIGMOD Int. Conf. on Management of Data*, pp. 325–335.
22. J.M. Hellerstein and M. Stonebraker. "Predicate migration: Optimizing queries with expensive predicates," in *ACM SIGMOD Int. Conf. on Management of Data*, pp. 267–276, 1993.
23. IBM. Informix geodetic datablade module: User's guide, 2002.
24. IBM. DB2 spatial extender and geodetic data management feature—user's guide and reference, 2006.
25. International Standard Organization. ISO 19107: Geographic information—Spatial schema, 2003.
26. M. McKenney, A. Pauly, R. Praing, and M. Schneider. Dimension-Refined Topological Predicates. *13th ACM Symp. on Geographic Information Systems*, pp. 240–249, 2005.
27. Open Geospatial Consortium Incorporation. OpenGIS implementation specification for geographic information – simple feature access – Part 1: Common architecture, 2006.
28. Oracle Corporation. Oracle Spatial User's Guide and Reference 10g Release 2, 2005.
29. J.A. Orenstein and F.A. Manola. "PROBE Spatial Data Modeling and Query Processing in an Image Database Application," *IEEE Transactions on Software Engineering*, Vol. 14:611–629, 1988.
30. R. Praing and M. Schneider. Efficient implementation techniques for topological predicates on complex spatial objects: The evaluation phase. Technical Report, University of Florida, Department of Computer & Information Science & Engineering, 2006.
31. M.A. Rodriguez, M.J. Egenhofer, and A.D.Blaser. "Query pre-processing of topological constraints: Comparing a composition-based with neighborhood-based approach," in *Int. Symp. on Spatial and Temporal Databases, LNCS 2750*, pp. 362–379. Springer-Verlag, 2003.
32. M. Schneider. "Spatial data types for database systems-finite resolution geometry for geographic information systems," volume LNCS 1288. Springer-Verlag, Berlin Heidelberg, 1997.
33. M. Schneider. "Implementing topological predicates for complex regions," in *Int. Symp. on Spatial Data Handling*, pp. 313–328, 2002.

34. M. Schneider. "Computing the topological relationship of complex regions," in *15th Int. Conf. on Database and Expert Systems Applications*, pp. 844–853, 2004.
35. M. Schneider and T. Behr. "Topological relationships between complex spatial objects," *ACM Transactions on Database Systems*, Vol. 31(1):39–81, 2006.
36. M. Schneider and R. Praing. Efficient implementation techniques for topological predicates on complex spatial objects: The exploration phase. Technical Report, University of Florida, Department of Computer & Information Science & Engineering, 2006.
37. Vivid Solutions. JTS Topology Suite: Technical Specifications, 2003.
38. M.F. Worboys and P. Bofakos. "A canonical model for a class of areal spatial objects," in *3rd Int. Symp. on Advances in Spatial Databases* (*LNCS 692*), pp. 36–52. Springer-Verlag, 1993.

Reasey Praing is a Ph.D. student and a research assistant in the Computer and Information Science and Engineering department at the University of Florida. He has a Master of Science degree from the University of Southern California. His research interests are spatial, spatio-temporal, and moving objects databases. He has published about 10 articles and conference papers on spatial and spatio-temporal database systems.

Markus Schneider is an Assistant Professor of Computer Science at the University of Florida and holds a doctoral degree from the University of Hagen, Germany. His research interests are databases in general, advanced databases for new, emerging applications, spatial databases, fuzzy spatial databases, and spatio-temporal and moving objects databases. He is coauthor of a textbook on moving objects databases, author of a monograph in the area of spatial databases, author of a German textbook on implementation concepts for database systems, and has published about 70 articles, conference papers, and book chapters on database systems. He is on the editorial board of GeoInformatica.