Geoinformatica (2019) 23: 79–104
https://doi.org/10.1007/s10707-018-00335-w

Query rewriting for semantic query optimization in spatial databases

Eduardo Mella[1] · M. Andrea Rodríguez[1] · Loreto Bravo[2] · Diego Gatica[1]

Received: 26 January 2018 / Revised: 21 October 2018 / Accepted: 28 November 2018 /
Published online: 4 January 2019

Abstract
Query processing is an important challenge for spatial databases due to the use of complex data types that represent spatial attributes. In particular, due to the cost of spatial joins, several optimization algorithms based on indexing structures exist. The work in this paper proposes a strategy for semantic query optimization of spatial join queries. The strategy detects queries with empty results and rewrites queries to eliminate unnecessary spatial joins or to replace spatial by thematic joins. This is done automatically by analyzing the semantics imposed by the database schema through topological dependencies and topological referential integrity constraints. In this way, the strategy comes to complement current state-of-art algorithms for processing spatial join queries. The experimental evaluation with real data sets shows that the optimization strategy can achieve a decrease in the time cost of a join query using indexing structures in a spatial database management system (SDBMS).

Keywords Spatial databases · Semantic optimization · Spatial query rewriting · Spatial integrity constraints

1 Introduction

Query processing in spatial databases can be very expensive due to the complex data types that represent spatial attributes. Indeed, spatial joins are one of the most expensive

✉ M. Andrea Rodríguez
andrea.rodriguez@udec.cl

Eduardo Mella
edumella@udec.cl

Loreto Bravo
bravo@udd.cl

Diego Gatica
dgatica@udec.cl

[1] Computer Science Department, Millenium Institute for Foundational Research on Data, Universidad de Concepción, Concepción, Chile

[2] Data Science Institute, Universidad del Desarrollo, Santiago, Chile

operations in spatial databases, which involve a selection based on a spatial relation, in most cases, a topological relation such as overlap or intersect. Due to the increasing use of geographic data in different domains, there exists a large effort to design optimization methods as part of the spatial database management systems. These methods typically rely on a prefiltering step using spatial indexing and the approximation of geometries in terms of their minimum bounding boxes upon which the index partitions the data and performs internal-memory spatial joins. Then, a refinement step verifies the exact spatial relation between objects [2, 5, 14, 16]. The work in this paper proposes a method for semantic query optimization [8, 13] that uses integrity constraints to rewrite a query that generates the same answer set in a more efficient way. In this sense, this strategy comes to complement current state-of-art algorithms for spatial joins.

Semantic query optimization complements existing methods for query processing by using knowledge that is application dependent. It requires the specification of integrity constraints or other types of knowledge by the database designers who are not necessarily the final users of the database. We argue that users (i.e., persons or systems) who are not the designers of the databases may write queries that do not exploit the semantics of the data to reduce the processing cost and, in consequence, providing automatic mechanisms for semantic query optimization contributes to the development of more efficient information systems.

Several strategies for semantic query optimization exist: join elimination if the value is known in advance, join addition if the result is highly selective, predicate elimination if the predicate is always true, and predicate addition if the indexed attributes produce faster access plan [6, 7, 15]. For relational databases, a method of query optimization based on query rewriting is the *Chase* and *Backchase* [8]. The *Chase* rewrites the original query into a query that integrates alternative pathways that are allowed by the constraints. Then, the *Backchase* produces new queries by eliminating various combinations of redundancies according to constraints [7].

For extended relational databases, the work in [17] investigates a general approach for semantic query optimization. They use integrity constraints to partition the process of conjunctive queries in order to reduce the number of tests on expensive conditions. The basic idea of this work is that when a query q can be partially answered by another q' that is considerably cheaper than q, then it can be useful to divide the process of q in three steps: (i) answering q' and ignoring the tuple combinations in this answer from the original base relations, (ii) checking possible answers in the tuple combinations that are left, and then (iii) combining answers.

A recent work [3] proposes to use an ontology and semantic integrity constraints to reduce the number of spatial relations for knowledge discovery. The basic idea is to avoid the computation of relations which are irrelevant to the discovery process. For example, if an island by definition is surrounded by water, their spatial relation is explicitly specified by the ontology or integrity constraints and does not need to be computed.

Inspired by the *Chase* and *Backchase* strategy [8], we propose in this paper a strategy that rewrites queries, which can then be processed with efficient algorithms. Indeed, our work does not provide a new algorithm for spatial joins, but rather algorithms that preprocess integrity constraints and rewrite queries. The basic assumption of this work is that spatial joins are more costly than traditional thematic joins, and therefore, by eliminating unnecessary joins and replacing them by thematic joins, we can improve the performance of query processing. We show in this paper that this process can be done automatically based on integrity constraints. Consequently, we validate our proposal with an experimental evaluation that uses a real data set on a real SDBMS. This evaluation compares the temporal cost of join queries with and without rewriting, and with a partition-based strategy for query optimization.

Although the theories behind previous strategies can be applicable to not only relational databases, to the best of our knowledge, there has been no attempt to use spatial semantic integrity constraints with optimization purposes in spatial databases. The work in this paper uses topological dependency and topological referential constraints [4] in addition to key constraints to optimize spatial join queries. It describes algorithms for the query rewriting and provides an experimental evaluation to validate the usefulness of the proposed strategy.

The organization of this paper is as follows. Section 2 revises preliminary concepts that are needed to understand the optimization method defined in Section 3. The experiments are in Section 4 followed by conclusions and future research directions in Section 5.

2 Preliminary

In this section, we provide a brief formalization of spatial databases and the integrity constraints used in this work.

2.1 Spatial database model

A spatial database is seen as an extension of a relational database that contains both thematic and spatial attributes. Formally, a spatio-relational database schema is a finite set $\mathcal{R}$ of relational predicates, each predicate with a finite and ordered set of attributes. We use $\mathbf{R}(B_1, \ldots, B_l)$ to represent the predicate $\mathbf{R}$ with its attributes $B_1, \ldots, B_l$. To simplicity the presentation, we will use *single-spatial* schemas where every $\mathbf{R} \in \mathcal{R}$ contains at most one spatial attribute that represents regions, lines or points.

Example 1 Consider a spatial database that stores information of places and administrative boundaries of counties. The schema of the spatio-relational database $\mathcal{R}$ is composed of a set of relational predicates $\mathcal{R} = \{\, \mathbf{St}(Id_s, Name, G_s),\ \mathbf{Co}(Id_c, Id_s, Name, G_c)$, and $\mathbf{Pl}(Id_p, Id_c, Name, G_p)\}$. Predicate **St** stores information of states, where the thematic attributes are Id_s and *Name*, and the only spatial attribute that stores the states' boundaries is G_s. Predicate **Co** stores information of counties with thematic attributes Id_c, Id_s, and *Name*, and a spatial attribute G_c that stores the boundaries of counties. Predicate **Pl** stores information of places and has thematic attributes Id_p, *Name*, and Id_c, and a spatial attribute G_p. Figure 1 shows an instance of this schema, in which thick lines represent boundaries of states, dashed lines represent counties, and gray rectangles represent places.

Spatial objects are related by spatial relations, which are usually classified into distance, orientation, or topological relations [21, 22]. Among these spatial relations, topological relations have gained much interest [10, 19], and are currently implemented in Spatial SQL Languages (SSQLs) [18].

Topological relations and composition Topological relations are invariant under continuous transformations such as translation, rotation, and scaling. There exist eight base topological relations between two non-empty regions x and y: Overlap(x, y), Equal(x, y), Inside(x, y), Includes(x, y), Covers(x, y), CoveredBy(x, y), Meet(x, y), and Disjoint(x, y) [9][1]. These are called *base relations* because only one of them relates two given regions and,

[1]The names given in [9] have their equivalence in the relations in [19].

St		
Id_s	$Name$	G_s
s_1	$name_1$	s_1
s_2	$name_2$	s_2

Pl			
Id_p	Id_c	$Name$	G_p
p_1	c_3	$name_2$	p_1
p_2	c_6	$name_3$	p_2

Co			
Id_c	Id_s	$Name$	G_c
c_1	s_1	$name_4$	g_1
c_2	s_1	$name_5$	g_2
c_3	s_1	$name_6$	g_3
c_4	s_2	$name_7$	g_4
c_5	s_2	$name_8$	g_5
c_6	s_2	$name_9$	g_6

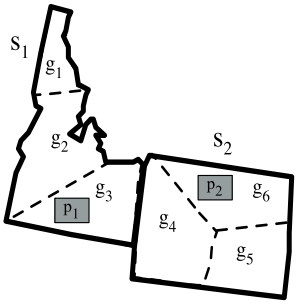


Fig. 1 A spatial database instance

therefore, these relations can be used to represent in an unambiguous way the topological relations between regions.

A derived topological relation is a set of base topological relations that is satisfied if any of its base relations is true. Commonly used derived relations are Contains and Within. Contains(x, y) holds if either Equal(x, y), Includes(x, y), or Covers(x, y) is true. Likewise, object x is Within object y if either Equal(x, y), Inside(x, y) or CoveredBy(x, y) is true. Using set notation, we write Contains $=$ {Equal, Includes, Covers} and Within $=$ {Equal, Inside, CoveredBy}. In addition, it is also possible to distinguish topological relations depending on the type of spatial feature (i.e., points, lines, or polygons) [11].

Current Spatial Database Systems, such as PostGIS, implement relations as defined by the Open Geospatial Consortium [18]. They have languages with relations Overlap, Equal, Disjoint, Meet[2], and the derived relations Contains and Within. Derived relations that are singletons correspond to their base relation, e.g., for a derived relation T_t = {Overlap}, it holds that $\mathsf{T}_t(x, y)$ if and only if Overlap(x, y). In this specification, these topological relations may apply over different types of geometries depending on its realizability. For example, the relation Equal does not apply between lines and polygons.

Table 1 contains the definitions of the topological relations currently used by PostGIS and useful for the purpose of the work in this paper, where x and y are non-empty regions. In the experimental evaluation we use the names of topological relations as used in current query languages and defined in [18].

Because we define topological relations as a set of base relations, we can naturally define inclusion ($\subseteq$) and intersection ($\cap$) between them. For example, {Disjoint} $\subseteq$ {Disjoint, Overlap} and {Disjoint} $\cap$ {Disjoint, Overlap} = {Disjoint}.

It is worth to notice that each topological relation has its converse relation. In particular, Disjoint, Meet, Equal, and Overlap are symmetric relations such that they are also their converse relations. Within and Contains, however, are the converse relation to each other.

An important concept used later for semantic query optimization is the notion of the *composition* of topological relations (see [9] for a complete description of the composition of topological relations). The *composition* of two topological rela-

[2]Note that PostGIS uses the name Touches instead of relation Meet.

Table 1 Definition of topological relations by the Open Geospatial Consortium [18]

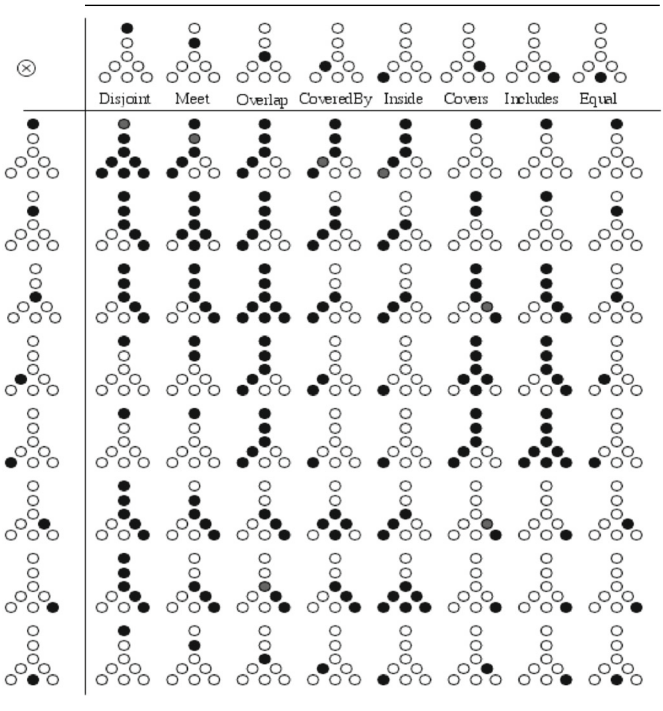
Relation	Is true if ...
Disjoint(x, y)	The intersection of x and y is the empty set
Meet(x, y)	The only points in common between x and y lie in the union of the boundaries of x and y
Overlap$s(x, y)$	The intersection of x and y results in a value of the same dimension as x and y that is different from both x and y
Equal(x, y)	x and y are equal
Within(x, y)	x is completely contained in y
Contains(x, y)	x is completely inside of y

tions $\mathsf{T}_1(x, y)$ and $\mathsf{T}_2(y, z)$ over a common object y, denoted by $\mathsf{T}_1(x, y) \otimes \mathsf{T}_2(y, z)$, enables the derivation of the topological relation between objects x and z. For example, Overlap$(x, y) \otimes$ CoveredBy(y, z)=$\mathsf{T}_\otimes(x, z)$, where $\mathsf{T}_\otimes$={Overlap, CoveredBy, Inside} and thus, if Overlap(a, b) and CoveredBy(b, c) are true, we know that either Overlap(a, c), CoveredBy(a, c), or Inside(a, c) is also true. Table 2 shows the composition of base topological relations between regions [12].

2.2 Integrity constraints

We complement the definition of the database schema with a set of integrity constraints that define the valid states of the database instances. The work in this paper concentrates and generalizes a subset of spatial semantic constraints as defined in [4].

Table 2 Composition of basic topological relations [12]



In what follows, we denote $x_1, \ldots, x_n, u, z$ variables in thematic attributes; $\bar{x}_1, \ldots, \bar{x}_n, \bar{u}, \bar{z}$, possible empty, sequences of distinct variables in thematic attributes; $g, g_1, \ldots, g_n$ variables in spatial attributes; and when necessary for space, $\overline{\forall}$ and $\bar{\exists}$ are universal and existential quantifications, respectively, over all variables (e.g., $\forall x_1 y_1 z_1 ...$).

Given a database schema $\mathcal{R}$, $\mathbf{P}$ and $\mathbf{R}$ two, not necessarily different, relational predicates in $\mathcal{R}$, and a topological relation T, a *spatial semantic integrity constraint* (SSIC) is one of the following:

Topological Dependencies (TDs)

$$\forall \bar{x} \bar{y} g_1 g_2 (\mathbf{P}(\bar{x}, g_1) \wedge \mathbf{R}(\bar{y}, g_2) \wedge \varphi \rightarrow \mathsf{T}(g_1, g_2)),$$

where $\bar{x} \cap \bar{y} = \emptyset$ and φ is a possibly empty CNF formula of equalities or inequalities between variables in $\bar{x}$ and/or $\bar{y}$.

Example 2 (Example 1 cont.) Possible TDs for the schema in Example 1 are:

$$\overline{\forall}(\ \mathbf{St}(x_1, y_1, g_1) \wedge\ \mathbf{St}(x_2, y_2, g_2) \wedge x_1 = x_2 \ \rightarrow\ \mathsf{Equal}(g_1, g_2)) \quad (1)$$

$$\overline{\forall}(\ \mathbf{Co}(x_1, y_1, z_1, g_1) \wedge\ \mathbf{Co}(x_2, y_2, z_2, g_2) \wedge x_1 = x_2 \ \rightarrow\ \mathsf{Equal}(g_1, g_2)) \quad (2)$$

$$\overline{\forall}(\ \mathbf{Pl}(x_1, y_1, z_1, g_1) \wedge\ \mathbf{Pl}(x_2, y_2, z_2, g_2) \wedge x_1 = x_2 \ \rightarrow\ \mathsf{Equal}(g_1, g_2)) \quad (3)$$

$$\overline{\forall}(\ \mathbf{Co}(x_1, y_1, z_1, g_1) \wedge\ \mathbf{Co}(x_2, y_2, z_2, g_2) \wedge x_1 \neq x_2 \ \rightarrow\ \mathsf{Meet}(g_1, g_2) \vee \mathsf{Disjoint}(g_1, g_2)) \quad (4)$$

$$\overline{\forall}(\ \mathbf{St}(x_1, y_1, g_1) \wedge\ \mathbf{St}(x_2, y_2, g_2) \wedge x_1 \neq x_2 \ \rightarrow\ \mathsf{Meet}(g_1, g_2) \vee \mathsf{Disjoint}(g_1, g_2)) \quad (5)$$

The first three TDs are classical dependency constraints applied over spatial attributes, that is, they express that two tuples of states, counties, or places with the same identification must have the same geometry. The last two TDs express that two different counties or states must have geometries that are disjoint or meet to each other.

Spatial Referential Dependencies (RDs)

$$\forall \bar{u} \bar{x} g_1 (\mathbf{P}(\bar{u}, \bar{x}, g_1) \rightarrow \exists \bar{v} \bar{y} g_2 (\mathbf{R}(\bar{v}, \bar{y}, g_2) \wedge \bar{u} = \bar{v} \wedge \mathsf{T}(g_1, g_2))),$$

where $\bar{u}$, $\bar{x}$, $\bar{v}$ and $\bar{y}$ are all disjoint, and where if $\bar{u} = [u_1, u_2, \ldots, u_n]$ and $\bar{v} = [v_1, v_2, \ldots, v_n]$, then $\bar{u} = \bar{v}$ denotes the conjunction $(u_1 = v_1) \wedge \ldots \wedge (u_n = v_n)$.

For the purpose of the optimization strategy, we assume that the referencing attributes of every foreign and referential dependency is set to be not null.

Example 3 (Example 1 cont.) Possible RDs for the schema in Example 1 are:

$$\forall x_1 y_1 z_1 g_1 (\ \mathbf{Pl}(x_1, y_1, z_1, g_1) \rightarrow \exists x_2 y_2 z_2 g_2 (\ \mathbf{Co}(x_2, y_2, z_2, g_2) \wedge y_1 = x_2 \wedge \mathsf{Within}(g_1, g_2))) \quad (6)$$

$$\forall x_1 y_1 z_1 g_1 (\ \mathbf{Co}(x_1, y_1, z_1, g_1) \rightarrow \exists x_2 y_2 g_2 (\ \mathbf{St}(x_2, y_2, g_2) \wedge y_1 = x_2 \wedge \mathsf{Within}(g_1, g_2))) \quad (7)$$

The first RD expresses that for every place, there must be a county that contains it. Similarly, the second RD expresses that for every county, there must exist a state that contains it. □

Relational key constraints In addition to the spatial integrity constraints, we also have the well-known primary-key and foreign-key constraints provided by the relational database model. Recall that primary keys satisfy functional dependencies and foreign keys satisfy referential dependencies over thematic attributes.

Example 4 (Example 1 cont.) Given the schema in Example 1, for each predicate the following thematic attributes are the corresponding primary and foreign keys.

Predicate	Primary key	Foreign key
St	Id_s	–
Co	Id_c	Id_s
Pl	Id_p	Id_c

Remark 1 When identifying primary keys, it is possible to automatically derive TDs. Indeed, the first three TDs of Example 2 are constraints derived from primary-key constraints. Also, if we have spatial semantics for foreign-key constraints between predicates with spatial reference, we can express spatial referential dependencies. For example, by knowing that foreign-key constraints imply an inclusion relationship, we could express constraints of Example 3.

2.3 Spatial join queries

We concentrate on join queries, which are conjunctive queries of the form:

$$q(\bar{x}) : \exists \bar{z}(\mathbf{P}(\bar{y}_1, g_1) \wedge \mathbf{R}(\bar{y}_2, g_2) \wedge \psi_q \wedge \mathsf{T}_q(g_1, g_2)),$$

where $\bar{x}$ are free variables, this is, $\bar{x} = (\bar{y}_1 \cup \bar{y}_2 \cup \{g_1, g_2\}) \setminus \bar{z}$, and ψ_q is a possibly empty formula of equalities or inequalities of variables in $\bar{y}_1 \cup \bar{y}_2$.

Example 5 (Example 1 cont.) A possible join query in the schema of Example 1 is:

$$q(x_1, x_2, g_1, g_2) : \exists y_1 z_1 y_2 z_2 (\ \mathbf{Co}(x_1, y_1, z_1, g_1) \wedge\ \mathbf{Co}(x_2, y_2, z_2, g_2) \wedge \mathsf{Meet}(g_1, g_2)).$$

This query requests the identification and geometries of counties that meet each other.

A join query compares the spatial attributes from two tuples in order to find the pairs of spatial attributes where the topological search condition $\mathsf{T}(g_1, g_2)$ holds true. The execution cost of queries that involve search conditions over spatial attributes is high in comparison with queries using search conditions on thematic attributes.

3 Semantic query optimization

Given a spatial join query over a database instance that satisfies a set of TDs and RDs, we propose an optimization strategy that:

(i) detects if the answer to the query is the empty set, or
(ii) generates a new query, which is equivalent to but less expensive than the original query, what we call rewriting.

The method detects an empty answer by finding a contradiction between the spatial restriction (i.e, restriction expressed by the topological relation) of the query with respect to the spatial restriction imposed by the integrity constraints. When rewriting the query, the main idea is to derive the topological relation that exists between spatial attributes and reformulate the query to remove the spatial condition (i.e., spatial join). In case we cannot

remove the spatial condition the method tries to create a more selective query by adding a new thematic condition what will filter tuples over which to check topological relations.

As it was stated before, a well-known strategy for query rewriting is the *Chase* and *Backchase* [8], which basically expands and then refines the query in terms of integrity constraints. In principle, because our constraints are expressions in first order logic, we could apply the same strategy by using the spatial constraints and composition of topological relations in the *Backchase* step. In what follows, we present an implementation of the semantic optimization that precomputes the knowledge about topological relations derived from constraints and composition, and that can be applied to any query. This avoids the process of *Chase* and *Backchase* for each query. We argue that this is a good approach because the number of constraints is usually much less than the number of queries imposed to a database.

The basic idea is to match relational predicates and the topological relation of a query with the predicates and the topological relation of spatial integrity constraints which are known to be satisfied by the database instance. Because we assume that the data satisfies the topological relations imposed by the constraints, we can simplify queries by removing conditions that are known to be satisfied. This is a simple idea, but the challenge is to create a systematic way to apply it.

To apply a matching process between constraints and queries, we first want to obtain the knowledge that can be used in this process. In consequence, we apply an inference process described in the next subsection that takes the original constraints and derives all possible additional constraints. This process includes combinations of TDs, RDs, and foreign-key conditions as we show below. Thus, given an original set of TDs and RDs, we first infer additional TDs and a generalization of TDs, called extended topological dependencies (ETDs). ETDs are a generalization of TDs as defined in [4] that can have more than two relational predicates. This step can be done as an off-line preprocess before processing any query.

3.1 Inference over spatial integrity constraints

In this section, we describe different types of inferences over combinations of RDs, TDs, and key constraints.

A) A first type of inference is possible when combining RDs and foreign-key conditions. Consider a RD of the form

$$\forall \bar{u}\bar{x}g_1(\mathbf{P}(\bar{u},\bar{x},g_1) \rightarrow \exists \bar{v}\bar{y}g_2\mathbf{R}(\bar{v},\bar{y},g_2) \wedge \bar{x} = \bar{v} \wedge \mathsf{T}(g_1,g_2)),$$

which expresses that if there exists a tuple in the relational predicate **P**, there must be a tuple in **R** where the thematic condition $\bar{x} = \bar{v}$ and the topological relation $\mathsf{T}(g_1, g_2)$ hold.

The RD alone does not ensure that the tuple in **R** such that $\bar{x} = \bar{v}$ holds is unique. However, combining the RD with the foreign-key (FK) condition such that attributes in $\bar{v}$ are a super key of **R**, we can derive a new TD that imposes that, whenever $\bar{x} = \bar{v}$ holds, the topological relation $\mathsf{T}(g_1, g_2)$ between geometries in tuples $\mathbf{P}(\bar{u}, \bar{x}, g_1)$ and $\mathbf{R}(\bar{v}, \bar{y}, g_2)$ will also hold. This is formalized with the following proposition.

Proposition 1 Given a database schema $\mathcal{R}$, a set of SSICs Σ and a topological relation T, if the RD $\forall \bar{u}\bar{x}g_1(\mathbf{P}(\bar{u},\bar{x},g_1) \rightarrow \exists \bar{v}\bar{y}g_2\mathbf{R}(\bar{v},\bar{y},g_2) \wedge \bar{x} = \bar{v} \wedge \mathsf{T}(g_1,g_2)) \in \Sigma$ and $\bar{v}$ is a super key of **R**, then the TD $\forall(\mathbf{P}(\bar{u},\bar{x},g_1) \wedge \mathbf{R}(\bar{v},\bar{y},g_2) \wedge \bar{x} = \bar{v} \rightarrow \mathsf{T}(g_1,g_2))$ is implied by Σ.

Proof $\forall \bar{u}\bar{x}g_1(\mathbf{P}(\bar{u},\bar{x},g_1) \rightarrow \exists \bar{v}\bar{y}g_2\mathbf{R}(\bar{v},\bar{y},g_2) \wedge \bar{x} = \bar{v} \wedge \mathsf{T}(g_1,g_2))$ with $\bar{v}$ a super key of **R**. Assume now that $\forall \bar{u}\bar{x}g_1\bar{v}\bar{y}g_2(\mathbf{P}(\bar{u},\bar{x},g_1) \wedge \mathbf{R}(\bar{v},\bar{y},g_2) \wedge \bar{x} = \bar{v} \rightarrow \mathsf{T}(g_1,g_2))$ is not true. Then, there must be tuples in **P** and **R** with $\bar{x} = \bar{v}$ such that $\mathsf{T}(g_1,g_2)$ doesn't hold. However, because $\bar{v}$ is a super key of **R**, there is only one tuple in **R** that holds $\bar{x} = \bar{v}$ and satifies $\mathsf{T}(g_1,g_2)$. We reach a contradiction. □

Notice that if we derive a TD involving predicates **P** and **R** from a RD as in Proposition 1 with thematic conditions $\bar{x} = \bar{v}$, and we have another TD with the negation of $\bar{x} = \bar{v}$, expressed as $\neg(\bar{x} = \bar{v})$, we have the complete knowledge about the topological relation between spatial attributes in any pair of tuples of **P** and **R**.

Example 6 (Example 3, 4 cont.) Considering the RD (6) in Example (3), primary key of **Co** in Example (4), and Proposition 1, we can infer the following constraint:

$$\forall x_1y_1z_1g_1x_2y_2z_2g_2(\ \mathbf{Pl}(x_1,y_1,z_1,g_1) \wedge\ \mathbf{Co}(x_2,y_2,z_2,g_2) \wedge y_1 = x_2 \rightarrow \mathsf{Within}(g_1,g_2)).$$

Similarly, from the RD (7), the primary key of **St**, it is possible to infer:

$$\forall x_1y_1z_1g_1x_2y_2z_2g_2(\ \mathbf{Co}(x_1,y_1,z_1,g_1) \wedge\ \mathbf{St}(x_2,y_2,g_2) \wedge y_1 = x_2 \rightarrow \mathsf{Within}(g_1,g_2)).$$

B) A second type of inference extends the notion of composition of topological relations [9] to the composition of TDs, which is formalized in the following axiom.

Proposition 2 Given a database schema $\mathcal{R}$, a set of SSICs Σ and topological relations T_1 and T_2, if the TDs $\forall \bar{x}\bar{y}g_1g_2(\mathbf{P}(\bar{x},g_1) \wedge \mathbf{Q}(\bar{y},g_2) \wedge \varphi_1 \rightarrow \mathsf{T}_1(g_1,g_2)) \in \Sigma$ and $\forall \bar{y}\bar{z}g_2g_3(\mathbf{Q}(\bar{y},g_2) \wedge \mathbf{R}(\bar{z},g_3) \wedge \varphi_2 \rightarrow \mathsf{T}_2(g_2,g_3)) \in \Sigma$, then the TD $\bar{\forall}(\mathbf{P}(\bar{x},g_1) \wedge \mathbf{Q}(\bar{y},g_2) \wedge \mathbf{R}(\bar{z},g_3) \wedge \varphi_1 \wedge \varphi_2 \rightarrow \mathsf{T}(g_1,g_3))$, with $\mathsf{T} = \mathsf{T}_1 \otimes \mathsf{T}_2$ is implied by Σ.

Proof Direct from the composition of topological relations. □

Notice that the result of the composition produces a rule with more than two predicates and, therefore, we refer to this constraint as an *extended topological dependency* (ETD). This additional predicate is necessary to keep the conditions on the thematic attributes that involve all three predicates.

Given an ETD or RD ϕ, we will denote by $\mathsf{TP}(\phi)$ the pair of atoms whose spatial attributes are compared by the topological relation in ϕ. For example, for ϕ_1:$(\mathbf{P}(x_1,x_2,g_1) \wedge \mathbf{Q}(y_1,y_2,y_3,g_2) \wedge \mathbf{R}(z_1,g_3) \wedge x_1 = y_1 \wedge y_2 \neq z_1 \rightarrow \mathsf{Overlap}(g_1,g_3))$, and ϕ_2 : $\mathbf{P}(x_1,x_2,g_1) \rightarrow \exists y_1,y_2,y_3,g_2(\mathbf{Q}(y_1,y_2,y_3,g_2) \wedge x_2 = y_1 \wedge \mathsf{Within}(g_1,g_2))$ we have $\mathsf{TP}(\phi_1) = [\mathbf{P}(x_1,x_2,g_1), \mathbf{R}(z_1,g_3)]$ and $\mathsf{TP}(\phi_2) = [\mathbf{P}(x_1,x_2,g_1), \mathbf{Q}(y_1,y_2,y_3,g_2)]$. Note that for a RD ϕ, $\mathsf{TP}(\phi)$ is exactly the only two atoms of ϕ.

Example 7 (Example 6 cont.) Let us consider the TDs inferred in Example 6.

$$\forall x_1y_1z_1g_1x_2y_2z_2g_2(\ \mathbf{Pl}(x_1,y_1,z_1,g_1) \wedge\ \mathbf{Co}(x_2,y_2,z_2,g_2) \wedge y_1 = x_2 \rightarrow \mathsf{Within}(g_1,g_2)).$$

$$\forall x_1y_1z_1g_1x_2y_2z_2g_2(\ \mathbf{Co}(x_1,y_1,z_1,g_1) \wedge\ \mathbf{St}(x_2,y_2,g_2) \wedge y_1 = x_2 \rightarrow \mathsf{Within}(g_1,g_2)).$$

Based on Proposition 2 and the composition of topological relation shown in Table 2, a new ETD ϕ that constraints the topological relation between geometries in **Pl** and **St** through predicate **Co** is:

$$\forall x_1y_1z_1g_1x_2y_2z_2g_2x_3y_3z_3g_3(\ \mathbf{Pl}(x_1,y_1,z_1,g_1) \wedge\ \mathbf{Co}(x_2,y_2,z_2,g_2) \wedge\ \mathbf{St}(x_3,y_3,g_3)$$
$$\wedge y_1 = x_2 \wedge y_2 = x_3 \rightarrow \mathsf{Within}(g_1,g_3)).$$

In this case, $\mathsf{TP}(\phi) = [\ \mathbf{Pl}(x_1, y_1, z_1, g_1),\ \mathbf{St}(x_3, y_3, g_3)]$.

We can generalize the previous derivation to consider the composition of a sequence of topological dependencies, which is formalized with the following proposition.

Proposition 3 Given a database schema $\mathcal{R}$ and a set of SSICs Σ if all the following hold:

$\forall \bar{x}_1 \bar{x}_2 g_1 g_2 (\mathbf{P}_1(\bar{u}_1 \bar{x}_1, g_1) \wedge \mathbf{P}_2(\bar{x}_2, g_2) \wedge \varphi_1 \rightarrow \mathsf{T}_1(g_1, g_2)) \in \Sigma$

$\vdots$

$\forall \bar{x}_i \bar{x}_{i+1} g_i g_{i+1} (\mathbf{P}_i(\bar{u}_i \bar{x}_i, g_i) \wedge \mathbf{P}_{i+1}(\bar{x}_{i+1}, g_{i+1}) \wedge \varphi_i \rightarrow \mathsf{T}_i(g_i, g_{i+1})) \in \Sigma$

$\vdots$

$\forall \bar{x}_{n-1} \bar{x}_n g_{n-1} g_n (\mathbf{P}_{n-1}(\bar{x}_{n-1}, g_{n-1}) \wedge \mathbf{P}_n(\bar{x}_n, g_n) \wedge \varphi_{n-1} \rightarrow \mathsf{T}_{n-1}(g_{n-1}, g_n)) \in \Sigma$

then the ETD $\overline{\forall}(\mathbf{P}_1(\bar{x}_1, g_1) \wedge \ldots \wedge \mathbf{P}_n(\bar{x}_n, g_n) \wedge \varphi_1 \wedge \ldots \wedge \varphi_{n-1} \wedge \mathsf{T}(g_1, g_n)$, with $\mathsf{T} = \mathsf{T}_1 \otimes \ldots \otimes \mathsf{T}_i \otimes \ldots \otimes \mathsf{T}_{n-1}$ implied by Σ.

The proof is a straight forward extension of the proof for Proposition 2.

It is easy to see that this proposition can be also generalized to consider the composition of ETDs.

C) Because RDs can infer TDs (see Proposition 1), and from this TDs we can infer ETDs, it is also possible to have propositions to infer ETDs from the composition of RDs. We are particularly interested in these ETDs because we can ensure that they can be applied to tuples that are enforced to exist due to the referential dependencies. This is better explained with the following example.

Example 8 Let us consider example 7 where the following ETD is implied:

$$\forall x_1 y_1 z_1 g_1 x_2 y_2 z_2 g_2 x_3 y_3 z_3 g_3 (\ \mathbf{Pl}(x_1, y_1, z_1, g_1) \wedge\ \mathbf{Co}(x_2, y_2, z_2, g_2) \wedge\ \mathbf{St}(x_3, y_3, g_3) \wedge y_1 = x_2 \wedge y_2 = x_3 \rightarrow \mathsf{Within}(g_1, g_3)).$$

This constraint is relevant to optimize join queries over predicates **Pl** and **St**. However, only if there exists a tuple in **Co** that satisfies the given thematic conditions $y_1 = x_2 \wedge y_2 = x_3$, the topological relation can be used to optimize the query. The existence of RDs (6) and (7) in Example 3 makes it possible to ensure that such tuple does indeed exist.

As the previous example shows, we will be particularly interested in ETDs that are derived from a sequence of RDs so that we can be sure there exist tuples for each atom in the ETDs. Keeping this in mind, and using the propositions described in this section, we now introduce the notion of a composition-dependency graph in a constructive fashion. This graph is an operational structure that stores the information of the TDs and ETDs that are derived during the preprocess of constraints and are useful for the optimization process.

3.2 Composition-dependency graph

Definition 1 Given a schema $\mathcal{R}$ and a set of SSICs Σ, let :

(i) $\Psi_{PK} = \{\psi'_1, \psi'_2, \ldots, \psi'_k\}$ be the set of TDs in Σ derived from primary keys.

(ii) $\Psi_{RD} = \{\phi_1, \phi_2, \ldots, \phi_n\}$ be the set of RDs in Σ of the form $\forall \bar{u} \bar{x} g_1 (\mathbf{P}(\bar{u}, \bar{x}, g_1) \rightarrow \exists \bar{v} \bar{y} g_2 (\mathbf{Q}(\bar{v}, \bar{y}, g_2) \wedge \bar{x} = \bar{v} \wedge \mathsf{T}(g_1, g_2)))$, with $\bar{v}$ a super key of **Q** and $\mathbf{P} \neq \mathbf{Q}$;

(iii) $\Psi_{TD} = \{\psi_1, \psi_2, \ldots, \psi_m\}$ be the set of TDs in Σ that are not derived from primary keys.

Without loss of generality we will assume that for every predicate $\mathbf{P} \in \mathcal{R}$ its atoms in $\Psi_{PK} \cup \Psi_{RD} \cup \Psi_{TD}$ use the same variables.

The *composition-dependency graph* (CDG) $\langle V, E, \lambda_r, \lambda_p, \lambda_s, \lambda_c, \lambda_t \rangle$ contains the relational predicates in Σ as vertices in set V, a set E of directed edges that represent TDs/ETDs between those vertices, and four labeling functions for the edges. Given an edge e, $\lambda_r(e)$ returns the ordered pair of endpoints in V including their variables, $\lambda_p(e)$ returns the conjunction of predicates in V that form part of the TD/ETDs excluding the endpoints (or True if there is no such predicates or conditions), $\lambda_s(e)$ labels the edge with the way in which the edge was created, which is codified by colors in the following description, $\lambda_c(e)$ returns the thematic conditions under which the constraint is triggered (or True if there is no such condition), and $\lambda_t(e)$ returns the topological relation associated with the edge.

The CDG is constructed iteratively with the following steps.

(G1) For every $\phi \in \Psi'_{PK}$ with $\phi : \forall \bar{x}_1 \bar{y}_1 \bar{x}_2 \bar{y}_2 g_1 g_2 (\mathbf{P}(\bar{x}_1, \bar{y}_1, g_1) \wedge \mathbf{P}(\bar{x}_2, \bar{y}_2, g_2) \wedge \varphi \rightarrow \mathsf{Equal}(g_1, g_2))$, let $e_\phi \in E$ with $\lambda_r(e_\phi) = \mathsf{TP}(\phi) = [\mathbf{P}(\bar{x}_1, \bar{y}_1, g_1), \mathbf{P}(\bar{x}_2, \bar{y}_2, g_2)]$, $\lambda_p(e_\phi) = \mathsf{True}$, $\lambda_s(e_\phi) = red$ (solid line), $\lambda_c(e_\phi) = \varphi$ and $\lambda_t(e_\phi) = \mathsf{Equal}(g_1, g_2)$.

(G2) For every $\phi \in \Psi_{RD}$ with $\phi : \forall \bar{x}_1 \bar{x}_2 g_1 g_2 (\mathbf{P}_1(\bar{x}_1, g_1) \rightarrow \exists \bar{x}_2, g_2 (\mathbf{P}_2(\bar{x}_2, g_2) \wedge \varphi_1 \wedge \mathsf{T}_1(g_1, g_2)))$ and φ_1 a condition of the primary key of $\mathbf{P}_2$, we can deduce by Proposition 1 a TD $\phi' : \forall \bar{x}_1 \bar{x}_2 g_1 g_2 (\mathbf{P}_1(\bar{x}_1, g_1) \wedge \mathbf{P}_2(\bar{x}_2, g_2) \wedge \varphi_1 \rightarrow \mathsf{T}_1(g_1, g_2))$. For each of these ϕ', let $e_{\phi'} \in E$ with $\lambda_r(e'_\phi) = \mathsf{TP}(\phi') = [\mathbf{P}_1(\bar{x}_1, g_1), \mathbf{P}_2(\bar{x}_2, g_2)]$, $\lambda_p(e'_\phi) = \mathsf{True}$, $\lambda_s(e'_\phi) = green$ (dashed line), $\lambda_c(e'_\phi) = \varphi_1$ and $\lambda_t(e'_\phi) = \mathsf{T}_1(g_1, g_2)$

(G3) For every $\phi \in \Psi_{TD}$ with $\phi : \forall \bar{x}_1 \bar{x}_2 g_1 g_2 (\mathbf{P}_1(\bar{x}_1, g_1) \wedge \mathbf{P}_2(\bar{x}_2, g_2) \wedge \varphi \rightarrow \mathsf{T}(g_1, g_2))$, let $e_\phi \in E$ with $\lambda_r(e_\phi) = \mathsf{TP}(\phi) = [\mathbf{P}_1(\bar{x}_1, g_1), \mathbf{P}_2(\bar{x}_2, g_2)]$, $\lambda_p(e_\phi) = \mathsf{True}$, $\lambda_s(e_\phi) = blue$ (dotted line), $\lambda_c(e_\phi) = \varphi$ and $\lambda_t(e_\phi) = \mathsf{T}(g_1, g_2)$.

(G4) For every $e_1, e_2 \in E$ such that $\lambda_s(e_1) = green$, $\lambda_s(e_2) = green$, $\lambda_r(e_1) = [\mathbf{P}_1(\bar{x}_1, g_1), \mathbf{P}_2(\bar{x}_2, g_2)]$, $\lambda_r(e_2) = [\mathbf{P}_2(\bar{x}_2, g_2), \mathbf{P}_3(\bar{x}_3, g_3)]$, and using Proposition 3 add, if it does not exist, an edge e_3 to E such that $\lambda_r(e_3)$=$[\mathbf{P}_1(\bar{x}_1, g_1), \mathbf{P}_3(\bar{x}_3, g_3)]$, $\lambda_p(e_3) = \lambda_p(e_1) \wedge \mathbf{P}_2(\bar{x}_2, g_2) \wedge \lambda_p(e_2)$, $\lambda_s(e_3) = green$, $\lambda_c(e_3) = \lambda_c(e_1) \wedge \lambda_c(e_2)$ and $\lambda_t(e_3) = \lambda_t(e_1) \otimes \lambda_t(e_2)$.

Do this step until no new green edges are added.

(G5) For every $e_1, e_2 \in E$ such that $\lambda_s(e_1) = green$, $\lambda_s(e_2) = blue$, $\lambda_r(e_1) = [\mathbf{P}_1(\bar{x}_1, g_1), \mathbf{P}_2(\bar{x}_2, g_2)]$, $\lambda_r(e_2) = [\mathbf{P}_2(\bar{x}_2, g_2), \mathbf{P}_3(\bar{x}_3, g_3)]$ and $\mathbf{P}_1 \neq \mathbf{P}_3$, using Proposition 2 add e_3 to E with $\lambda_r(e_3) = [\mathbf{P}_1(\bar{x}_1, g_1), \mathbf{P}_3(\bar{x}_3, g_3)]$, $\lambda_p(e_3) = \lambda_p(e_1) \wedge \mathbf{P}_2(\bar{x}_2, g_2) \wedge \lambda_p(e_2)$, $\lambda_s(e_3) = red$, $\lambda_c(e_3) = \lambda_c(e_1) \wedge \lambda_c(e_2)$, $\lambda_t(e_3) = \lambda_t(e_1) \otimes \lambda_t(e_2)$.

Do this until no more red edges can be added

(G6) Do the following until no more changes can be made:

- (a) If an edge $e \in E$ has an inconsistent $\lambda_p(e) \wedge \lambda_c(e)$, then remove e from E.
- (b) If two parallel edges $e_1, e_2 \in E$ are such that $\lambda_c(e_1) \rightarrow \lambda_c(e_2)$ then let $\lambda_t(e_1) := \lambda_t(e_1) \cap \lambda_t(e_2)$.
- (c) If two edges have the same labels λ_r, λ_p, λ_s, λ_c and λ_t, remove one of the edges.
- (d) For each edge, we eliminate redundancy in the conditions and predicates that are not used in both thematic and topological conditions.

The CDG as defined above includes a set of edges that represent: (i) TDs derived from primary key constraints (as red edges). (ii) Original TDs (as blue edges). (ii) TDs derived

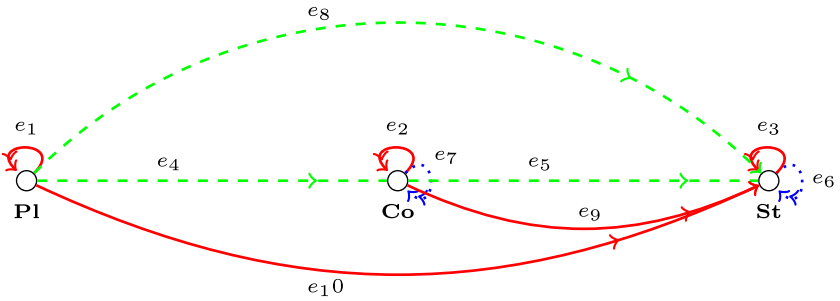


Fig. 2 Example of a composition-dependency graph (Red edges as solid lines, green edges as dashed lines and blue edges as dotted lines)

from RDs by Proposition 1 (as green edges). (iii) ETDs derived from composition of edges (Proposition 3) derived from (ii) (as green edges). (iv) ETDs derived from the composition of edges derived from (ii) plus an original TD (as red edges). When deriving new edges, red edges are not used for any further composition because they represent primary-key constraints that can lead to undesirable loops or because they are TDs whose composition with other edges cannot guarantee the existence of tuples that make possible the composition as we explained in the previous section.

Even though the composition-dependency graph does not have all the possible TDs/ETDs we can infer from a set of SSICs Σ, it includes the ones that are useful for the optimization process detailed in the next section.

Example 9 (Example 6 cont.) Figure 2 illustrates the composition-dependency graph derived from the constraints in Examples 2, 3, and 4. The graph is complemented with the Table 3 to describe the edges of the graph.

In this example, edges e_1 to e_3 can be automatically derived from primary-key conditions. Edges e_4 and e_5 add a spatial semantics to foreign-key constraints. Edges e_6 and e_7 are

Table 3 Description of edges of graph in Fig. 4

E	λ_r	λ_p	λ_s	λ_c	λ_t
e_1	[**Pl**(u, n, c, g_p), **Pl**(u', n', c', g'_p)]	True	red	$u = u'$	Equal(g_p, g'_p)
e_2	[**Co**(v, m, s, g_c), **Co**(v', m, s', g'_c)]	True	red	$v = v'$	Equal(g_c, g'_c)
e_3	[**St**(w, e, g_s), **St**(w', e', g'_s)]	True	red	$w = w'$	Equal(g_s, g'_s)
e_4	[**Pl**(u, c, n, g_p), **Co**(v, m, s, g_c)]	True	green	$c = v$	Within(g_p, g_c)
e_5	[**Co**(v, s, n, g_c), **St**(w, e, g_s)]	True	green	$w = s$	Within(g_p, g_c)
e_6	[**St**(v, s, g_c), **St**(w, e, g_s)]	True	blue	$v \neq w$	Meet$(g_c, g_s)\vee$ Disjoint(g_c, g_s)
e_7	[**Co**(v, s, n, g_c), **Co**(w, e, s, g_s)]	True	blue	$v \neq w$	Meet$(g_c, g_s)\vee$ Disjoint(g_c, g_s)
e_8	[**Pl**(u, c, n, g_p), **St**(w, e, g_s)]	**Co**(v, n, s, g_c)	green	$(c = v) \wedge (s = w)$	Within(g_p, g_s)
e_9	[**Co**(v, s, m, g_c), **St**(w, e, g_s)]	True	red	$s \neq w$	Meet$(g_c, g_s)\vee$ Disjoint(g_c, g_s)
e_{10}	[**Pl**(u, c, n, g_p), **St**(w, e, g_s)]	**Co**(v, s, n, g_c)	red	$(c = v) \wedge (s \neq w)$	Meet$(g_c, g_s)\vee$ Disjoint(g_c, g_s)

semantic constraints that should be specified by the database designer. Edges e_8 to e_{10} are derived automatically from our strategy.

Among the derived edges, a further process to eliminate redundancy was applied to edge e_9. Originally, by applying $(G5)$ we obtain a ETD of the form:

$$\overline{\forall}(\ \mathbf{Co}(v, s, m, g_c) \wedge\ \mathbf{St}(u, p, g_s') \wedge\ \mathbf{St}(w, e, g_s) \wedge s = u \wedge u \neq w \rightarrow \mathsf{Meet}(g_c, g_s) \vee \mathsf{Disjoint}(g_c, g_s)).$$

However, this ETD can be simplified by having that $s = u \wedge u \neq w \rightarrow s = w$ and topological conditions applied to attributes of only two predicates. So, we finally obtain:

$$\overline{\forall}(\ \mathbf{Co}(v, s, m, g_c) \wedge\ \mathbf{St}(w, e, g_s) \wedge s \neq w \rightarrow \mathsf{Meet}(g_c, g_s) \vee \mathsf{Disjoint}(g_c, g_s)).$$

3.3 Semantic query preprocessing

We describe the proposed optimization strategy based on a semantic query preprocessing in terms of a join query of the form

$$q(\bar{x}) : \exists \bar{z}(\mathbf{P}(\bar{y}_1, g_1) \wedge \mathbf{R}(\bar{y}_2, g_2) \wedge \psi_q \wedge \mathsf{T}_q(g_1, g_2)),$$

where $\bar{x}$ are free variables, this is $\bar{x} = (\bar{y}_1 \cup \bar{y}_2 \cup \{g_1, g_2\}) \setminus \bar{z}$, and ψ_q is a possibly empty formula of equalities or inequalities of variables in $\bar{y}_1 \cup \bar{y}_2$.

Given a set of constraints Σ that are satisfied by the data, the strategy works by matching the relational predicates in the query q with respect to relational predicates in TDs or ETDs in Σ. When the match occurs, the strategy analyzes query conditions (i.e., equalities or inequalities of thematic attributes and topological relations) to conclude if a query has empty results or it can be rewritten. The rewritten of a query can eliminate spatial-join conditions (i.e., topological relations) or add thematic conditions to reduce the number of tuples over which to check topological relations.

The optimization strategy uses integrity constraints to derive possible topological relations between spatial attributes and reduce, therefore, the computation cost of spatial joins. As we described in the previous section, if we have a TD of the form $\overline{\forall}(\mathbf{P}(\bar{x}, g_1) \wedge \mathbf{Q}(\bar{y}, g_2) \wedge \varphi_1 \rightarrow \mathsf{T}(g_1, g_2))$ and another TD of the form $\overline{\forall}(\mathbf{P}(\bar{x}, g_1) \wedge \mathbf{Q}(\bar{y}, g_2) \wedge \varphi_2 \rightarrow \mathsf{T}'(g_1, g_2))$, where $\varphi_2 = \neg\varphi_1$, we can completely characterize the topological relation between the geometry of any pair of tuples in $\mathbf{P}$ and $\mathbf{Q}$.

In general terms, an ETD can be seen as a formula of the form $\Phi \wedge \psi \rightarrow T(g, g')$, where Φ is a conjunction of relational predicates and thematic conditions, g and g' are variables of spatial attributes in predicates of Φ, and ψ is a equality or inequality of variables in Φ. Notice that there is a mapping from this expression to edges in a CDG G, such that for an edge $e \in G$, Φ is a conjunction of the predicates in $\lambda(e)$ and predicates and condition in $\lambda_p(e)$, ψ is $\lambda_c(e)$, and $T(g, g')$ is $\lambda_t(e)$. Now, if we have two ETDs of the form $\phi_1 : \Psi \wedge \psi \rightarrow T(g, g')$ and $\phi_2 : \Psi \wedge \neg\psi \rightarrow T'(g, g')$, where Ψ is known to be true, then ϕ_1 and ϕ_2 cover all possible cases to derive the topological relation between g and g'.

We are now in condition to introduce the semantic preprocessing to rewrite join queries.

Let us also consider a schema $\mathcal{R}$, a set of SSICs Σ, the CDG $G = \langle V, E, \lambda_r, \lambda_p, \lambda_s, \lambda_c, \lambda_t \rangle$ derived from Σ, and a join query $q(\bar{x}) : \exists \bar{z}(\mathbf{P}(\bar{y}_1, g_1) \wedge \mathbf{R}(\bar{y}_2, g_2) \wedge \psi_q \wedge \mathsf{T}_q(g_1, g_2))$. If there exists an edge $e \in E$ such that $\lambda_r(e) = [\mathbf{P}(\bar{y}_1, g_1), \mathbf{R}(\bar{y}_2, g_2)]$, it is possible to rewrite the query based on the three following cases, which are summarized in Table 4:

(C1) $\lambda_t(e) \cap \mathsf{T}_q = \emptyset$:

This means that under the condition imposed by $\lambda_p(e)$ and $\lambda_c(e)$, the topological relation of any pair of tuples does not satisfy T_q and, therefore, under this condition,

Table 4 TD/ETD-based optimization

Case	Condition	Rewritting
(C1)	$\lambda_t(e) \cap \mathsf{T}_q = \emptyset$	$\mathsf{q}_\emptyset$
(C2)	$\lambda_t(e) \subseteq \mathsf{T}_q$	$\bar{\exists}\mathbf{P}(x, g_1) \wedge \mathbf{R}(y, y_2) \wedge \lambda_p(e) \wedge \lambda_c(e) \wedge \psi_q$
(C3)	Otherwise	$\bar{\exists}\mathbf{P}(x, g_1) \wedge \mathbf{R}(y, y_2) \wedge \lambda_p(e) \wedge \lambda_c(e) \wedge \psi_q \wedge \mathsf{T}_q$

the result is empty, i.e., $\mathsf{q}_\emptyset$. Consequently, we have to check for pairs of tuples that satisfy ψ_q and $\lambda_p(e)$, but not $\lambda_c(e)$, that is, tuples that satisfy $\neg\lambda_c(e)$. As we stated in the previous section, ETDs included in the CDG guarantees that there exist tuples that satisfy $\lambda_p(e)$, which makes possible to evaluate conditions in $\lambda_c(e)$ or $\neg\lambda_c(e)$.

(C2) $\lambda_t(e) \subseteq \mathsf{T}_q$:

This means that under the condition imposed by $\lambda_p(e)$ and $\lambda_c(e)$, pairs of tuples satisfy not only $\lambda_t(e)$ but also T_q. Consequently, tuples that satisfy $\lambda_p(e)$ and $\lambda_c(e)$ and also ψ_q will satisfy the topological relation T_q. Notice, however, that $\lambda_c(e)$ and ψ_q may be in contradiction, (e.g., $\psi_q : a = b$ and $\psi_q : a \neq b$), in which case, the answer to this query will be empty. In addition, we have to check for those pairs of tuples that satisfy $\neg\lambda_c(e)$ but still may satisfy T_q.

(C3) Otherwise, this is, if $\lambda_t(e) \cap \mathsf{T}_q \neq \emptyset$:

In this case and because none of the previous conditions are satisfied, we run the query by adding $\lambda_p(e)$ and $\lambda_c(e)$, if any, to have a possibly more selective query with additional conditions that reduce the tuples over which to check topological relations. In this process, redundant conditions are eliminated.

We can describe the query preprocessing as a sequence of steps for a join query $q(\bar{x}) : \exists\bar{z}(\mathbf{P}(\bar{x}_1, g_1) \wedge \mathbf{R}(\bar{x}_2, g_2) \wedge \psi_q \wedge \mathsf{T}_q(g_1, g_2))$.

(S1) Construct the CDG G from integrity constraints. This is an off-line preprocessing.

(S2) Search for an edge e in G such that $\lambda_r(e) = [\mathbf{P}(\bar{x}_1, g_1), \mathbf{R}(\bar{x}_2, g_2)]$. If such edge e exists, go to the following step (3). If such edge does not exist, then check if there exists e' in G such that $\lambda_r(e') = [\mathbf{R}(\bar{x}_2, g_2), \mathbf{P}(\bar{x}_1, g_1)]$. If e' exists, go to the following step (S3) making the query $q(\bar{x}) : \exists\bar{z}(\mathbf{R}(\bar{x}_2, g_2) \wedge \mathbf{P}(\bar{x}_1, g_1) \wedge \psi_q \wedge \mathsf{T}_q^c(g_2, g_1))$, where T_q^c is the converse of T_q. If there is no such e', then return the original q.

(S3) Rewrite q into q_1 based on the comparison between $\lambda_t(e)$ and T_q using Table 4.

(S4) Check if there exists another e' in G such that $\lambda_r(e) = \lambda_r(e')$ and $\lambda_c(e) = \neg(\lambda_c(e'))$. If this e' exists, then create a new query q_2 based on the comparison between $\lambda_t(e')$ and T_q using Table 4. If e' does not exist, then make $q_2 : \bar{\exists}\mathbf{P}(x, g_1) \wedge \mathbf{R}(y, y_2) \wedge \lambda_p(e) \wedge \neg(\lambda_c(e)) \wedge \psi_q \wedge \mathsf{T}_q$.

(S5) Create query q^o as the union of q_1 and q_2. This is equivalente to a disjunction of q_1 and q_2.

(S6) Eliminate possible duplication of thematic conditions in q^o and detect any possible contradiction between thematic conditions. If there is any contradiction or if there is condition False in a subquery of q^o, the result of the subquery is the empty set.

This iterative process will result in a new query that can add relational predicates and thematic conditions as it happens when using a *Chase* and *Backchase* technique [1]. As we will see in the experimental evaluation, adding these conditions produces a query that can be processed more efficiently.

In the process described above, there may be many possible ETDs or TDs to apply in the query preprocessing. In such case, one could analyze which of the possible rewritings will produce queries whose process is more efficient and where the priority is to void the computation of the topological condition of the query.

Example 10 Let us consider the schema and constraints in Examples 2, 3, and 4, upon which the CDG G in Example 9 is derived. Also, let us have query $q(x_1, x_2) : \exists y_1 g_1 y_2 g_2\ \mathbf{St}(x_1, y_1, g_1) \wedge \mathbf{St}(x_2, y_2, g_2) \wedge \mathsf{Meet}(g_1, g_2)$, with ψ_q : True (i.e., an empty thematic condition). Edge e_3 matches predicates and, because $\{\mathsf{Meet}\} \cap \lambda_t(e_3) = \emptyset$, we can apply case (C1) in Table 4 giving as a first result $q_\emptyset$. Following step (S4) of the methodology, we find edge e_6 such that $\lambda_r(e_6) = \lambda_r(e_3)$ and apply case (C3) in Table 4. Then a second result q_2 is:

$$q_2(x_1, x_2) : \exists y_1 g_1 y_2 g_2(\ \mathbf{St}(x_1, y_1, g_1) \wedge\ \mathbf{St}(x_2, y_2, g_2) \wedge \neg(x_1 = x_2) \wedge \mathsf{Meet}(g_1, g_2)).$$

Then, $q(x_1, x_2)$ is rewritten into $q^o(x_1, x_2)$ by making the union of $q_\emptyset \cup q_2(x_1, x_2)$, which doesn't have contradictions, resulting in:

$$q^o(x_1, x_2) : \exists y_1 g_1 y_2 g_2(\ \mathbf{St}(x_1, y_1, g_1) \wedge\ \mathbf{St}(x_2, y_2, g_2) \wedge \neg(x_1 = x_2) \wedge \mathsf{Meet}(g_1, g_2)).$$

Let us consider now a second query $q(x_1x_2) : \exists y_1 z_1 g_1 y_2 g_2\ \mathbf{Pl}(x_1, y_1, z_1, g_1) \wedge \mathbf{St}(x_2, y_2, g_2) \wedge \mathsf{Meet}(g_1, g_2)$, with $\psi_{q'}$: True. From CDG G, we get the edge e_8 such that $\{\mathsf{Meet}\} \cap \lambda_t(e_8) = \{\mathsf{Meet}\} \cap \{\mathsf{Within}\} = \emptyset$ and a first result $q_\emptyset$. In (S4) the methodology finds a second edge e_{10} that matches relation predicate in query q' and satisfies case (C3) in Table 4, which produces a second query q_2:

$$q_2(x_1x_2) : \exists y_1 z_1 g_1 y_2 g_2 x_3 y_3 z_3 g_3(\ \mathbf{Pl}(x_1, y_1, z_1, g_1) \wedge\ \mathbf{Co}(x_3, y_3, z_3, g_3) \wedge\ \mathbf{St}(x_2, y_2, g_2) \wedge y_1 = x_3 \wedge y_3 \neq x_2 \rightarrow \mathsf{Meet}(g_1, g_2)).$$

Then, $q(x_1, x_2)$ is rewritten into $q^o(x_1, x_2)$ by making the union of $q_\emptyset \cup q_2(x_1, x_2)$ resulting in:

$$q^o(x_1x_2) : \exists y_1 z_1 g_1 y_2 g_2 x_3 y_3 z_3 g_3(\ \mathbf{Pl}(x_1, y_1, z_1, g_1) \wedge\ \mathbf{Co}(x_3, y_3, z_3, g_3) \wedge\ \mathbf{St}(x_2, y_2, g_2) \wedge y_1 = x_3 \wedge y_3 \neq x_2 \rightarrow \mathsf{Meet}(g_1, g_2)).$$

The preprocessing as defined above terminates. This can be guaranteed because there is a finite number of useful TDs and ETDs, and a finite number of cases in Table 4.

3.4 Algorithms

This section gives the algorithms for constructing a CDG and algorithms for the query rewriting.

Algorithm 1 constructs a CDG as described in Definition 1. The algorithm starts by creating nodes and adding edges derived from the set of topological dependencies Ψ_{TD}, a set denoted by Ψ'_{TD}, and referential dependencies Φ_{RD}; that is, adding loop edges on a single predicate of TDs in Ψ'_{TD} and edges between different relational predicates in TDs and RDs in Ψ_{TD} and Φ_{RD}, respectively. Then, the algorithm derives new edges by the composition of topological relations as it is described in steps (G4) and (G5) of Definition 1. Recall that by Definition 1, an iterative process can expand the composition of several *green* edges, but a composition between green and blue edges will end with a single composed red edge. We apply this composition in a breadth-search fashion, adding new green edges to the queue of previously existing edges.

Algorithm 1 Construction of a CDG

Input: Set Ψ_{RD}, Ψ_{TD}, and Ψ_{PK} as defined in Definition 1.
Output: A CDG $G = \langle V, E, \lambda_r, \lambda_p, \lambda_s, \lambda_c, \lambda_t \rangle$

```
CDG cdg ← newCDGraph(Ψ_TD, Ψ_PK, Ψ_RD) {Create a graph with nodes as predicates found in Ψ_PK ∪ Ψ_TD ∪ Ψ_RD}
for every ψ ∈ Ψ_PK do
    Edge e ← createEdge(ψ,'red') {Create a loop-red edge}
    cdg.addEdge(e) {Add edge to cdg}
for every ψ ∈ Ψ_TD do
    Edge e ← createEdge(ψ,'blue') {Create a blue edge from a TD}
    cdg.addEdge(e) {Add edge to cdg}
for every φ ∈ Φ_RD do
    Edge e ← createEdge(φ,'green') {Create a green edge from a RD}
    cdg.addEdge(e) {Add edge to cdg}
edgeComposition(cdg) {It adds edges by composition producing green edges (derived from RDs) or red edges (combination of RDs and TDs)}
refine(cdg) {Eliminate inconsistent thematic conditions and redundancy}
return
```

Algorithm 2 edgeComposition

Input: A CDG cdg.
Output: Graph cdg with new green edges by composition.

```
Queue q ← newQueque(cdg){Set a queue with green edges}
while q is not empty do
    Edge e ← q.front()
    for every edge e' ∈ cdg with λ_s(e') = 'green' such that λ_r(e) = [P(ū, g_p), R(v̄, g_r)] and λ(e') = [R(v̄, g_r), Q(w̄, g_q)] do
        Edge e'' ← composeEdge(e, e','green') {Create a green edge as the composition of edges e and e'}
        cdg.addEdge(e'')
        q.enqueue(e'')
for every edge e ∈ cdg with λ_s(e) = 'green' do
    for every edge e' ∈ cdg with λ_s(e') = 'blue' such that λ_r(e) = [P(ū, g_p), R(v̄, g_r)] and λ(e') = [R(v̄, g_r), Q(w̄, g_q)] do
        Edge e'' ← composeEdge(e, e','red') {Create a red edge as the composition of edges e and e'}
        cdg.addEdge(e'')
return g
```

Let n_1, n_2 and n_3 the number of constraints in Ψ_{RD}, Ψ_{TD}, and Ψ_{PK}, respectively. The complexity of the Algorithm 1 is divided into three steps: (i) Add the initial edges from Ψ_{PK}, Ψ_{TD} and Φ_{RD}. (ii) Add edges from the composition of green edges or between green and blue (Algorithm edgeComposition). (iii) Eliminate redundancies on thematic conditions. Step (i) has a cost of $O(n_1 + n_2 + n_3)$, assuming no duplication of constraints.

For the step (ii), only constraints in Ψ_{RD} and Ψ_{TD} are used. In the first part, the Algorithm edgeComposition creates new green edges from the composition of TDs derived from Ψ_{RD}. In the worse case, the algorithm will complete a graph, which has initially a

single path between any pair of nodes and, at the end, it has edges between any pair of nodes. For n_1 number of constraints, there are in the worse case, n_1 initial green edges. A first set of green edges is created by composing two adjacent edges in the initial graph. Using adjacency-matrix representation of the CDG with direct access to adjacent nodes, this implies $O(n_1 - 1)$ checks, producing at most $(n_1 - 1)$ edges. Notice that the nodes of these edges already exist in the graph. The new derived edges in the worst can then be composed with an original green, producing $(n_1 - 2)$, following a breadth search. Thus, the process is upper bound by $\Sigma_{i=1}^{n_1-1}(n_1 - i)$, which is in $O(n_1^2)$. For the second part of Algorithm edgeComposition, the composition needs just one iteration over green edges to find a blue edge with whom to be composed, given a cost equal to the total number of possible green edges, which in the worst case, it is in $O(n_1^2)$. Consequently, the total cost of (ii) is upper bound by $O(n_1^2)$.

Finally, the step (iii) refines the CDG by checking edges for redundancy. Using a structure with direct access to edges over the same pair of nodes gives us a linear cost with respect to the number of edges, which in the worst case it is upper bound by $O(n_1^2+n_2+n_3)$. Notice that in practice, this upper bound does not affect the cost of query processing because values of n_1, n_2, and n_3 are much lower than the number of tuples in the database instance.

Algorithm 3 applies the query preprocessing following the strategy defined in Section 3.3 using Table 4. In this algorithm, $\mathsf{q}_\emptyset$ is a query with empty results and ψ_q is, a possible empty, conjunction of thematic equalities or inequalities in the join query. Finally, the Algorithm 3 calls a subroutine cleanQuery to eliminate redundant conditions and to check contradiction of thematic conditions, in which case, the algorithm returns $\mathsf{q}_\emptyset$. Using direct access to edges in the CDG between two specific predicates, the algorithm run in linear time with respect to the number of edges in the CDG that match the predicates.

Algorithm 3 QueryPreprocessing

Input: A Join Query $q(\bar{x}) : \exists\bar{z}(\mathbf{P}(\bar{x}_1, g_1) \wedge \mathbf{R}(\bar{x}_2, g_2) \wedge \psi_q \wedge \mathsf{T}_q(g_1, g_2))$ and a CDG cdg.
Output: An rewriten query q_r built using the CDG which is equivalent to $q(\bar{x})$

Let $e \in cdg$ an edge such that $\lambda_r(e) = [\mathbf{P}(\bar{x}_1, g_1), \mathbf{R}(\bar{x}_2, g_2)]$; or let $e = null$ if such edge does not exist.
if $e = null$ **then**
  if $\exists e \in cdg$ such that $\lambda_r(e) = [\mathbf{R}(\bar{x}_2, g_2), \mathbf{P}(\bar{x}_1, g_1)]$ **then**
    $q^c(\bar{x}) \leftarrow \exists\bar{z}(\mathbf{R}(\bar{x}_2, g_2) \wedge \mathbf{P}(\bar{x}_1, g_1) \wedge \psi_q \wedge \mathsf{T}_q^c(g_2, g_1))$ **return** Optimizer($q^c(\bar{x})$, cdg)
  elsereturn q
else
  $q_1 \leftarrow$ rewriteQuery(q, e)
  Let $e' \in cdg$ an edge such that $\lambda_r(e') = [P(\bar{x}_1, g_1), \mathbf{R}(\bar{x}_2, g_2)]$ and $\lambda_c(e) = \neg\lambda_c(e')$; or let $e' = null$ if such edge does not exists.
  if $e' \neq null$ **then**
    $q_2 \leftarrow$ rewriteQuery(q, e')
  else
    $q_2 \leftarrow \bar{\exists} P(x, g_1) \wedge \mathbf{R}(y, y_2) \wedge \lambda_p(e) \wedge \neg(\lambda_c(e)) \wedge \psi_q \wedge \mathsf{T}_q$
$q_r \leftarrow$ cleanQuery$(q_1 \cup q_2)$ {*It returns $q_\emptyset$ if there exist contradictions in thematic conditions or it returns a query without redundant thematic conditions*}
return q_r

Algorithm 4 rewriteQuery

Input: A Join Query $q(\bar{x}) : \exists \bar{z}(P(\bar{y}_1, g_1) \wedge \mathbf{R}(\bar{y}_2, g_2) \wedge \psi_q \wedge \mathsf{T}_q(g_1, g_2))$ and an edge e of a CDG.
Output: An optimization of query $q(\bar{x})$ using edge e
if $(\mathsf{T}_q \cap \lambda_t(e) = \emptyset)$ **then**
  return $q_\emptyset$
else if$(\lambda_t(e) \subseteq \mathsf{T}_q)$ **then**
  return $\bar{\exists} P(x, g_1) \wedge \mathbf{R}(y, y_2) \wedge \lambda_p(e) \wedge \lambda_c(e) \wedge \psi_q$
else
  return $\bar{\exists} P(x, g_1) \wedge \mathbf{R}(y, y_2) \wedge \lambda_p(e) \wedge \lambda_c(e) \wedge \psi_q \wedge \mathsf{T}_q$

4 Experimental evaluation

We run an experiment to compare the time cost of processing join queries with and without rewriting. The experimental evaluation uses a real data set extracted from the TIGER/ Line Shapefile of the U.S. Census Bureau.[3] The data set consists of tables about states (**St**), counties (**Co**), places (**Pl**), landmarks (**Lm**), census tracks (**Ct**), census blocks (**Cb**), urban areas (**Ua**), and primary roads (**Rd**) of U.S.A, which cover data of the benchmark of spatial database performance in [20]. Table 5 shows the details of the data used in the experimental evaluation, where underlined attributes are primary keys, FKey are foreign keys, Size is the size of the tables in Postgres, and Index refers to the size of the indexed spatial attributes using R-tree. Note that in this table, the relation schema **Pl** has a foreign key to **St** and not to **Co** as in Example 1.

We include data about administrative boundaries in tables **St** and **Co** and spatial features of **Pl** and **Lm**, which have a hierarchical structure represented through foreign keys. The spatial semantics of these relations is then formally expressed by topological dependency and referential constraints.

We use subset of queries and data from an existing benchmark [20] to make the experimental evaluation replicable with respect to a benchmark found in the literature. In addition, we add **Ct**, **Cb**, **Ua**, and **Rd** to use larger data sets and features of type lines. As the experimental evaluation will show, using such common constraints as topological dependencies for primary keys results in a query rewritten that reduces the cost of processing spatial join queries.

To run the preprocessing, the data set is complemented with the set of spatial integrity constraints. Besides the traditional dependency constraints on thematic attributes that arise from primary and foreign key constraints, there are also topological dependencies that enforce primary keys on spatial attributes and topological and referential dependencies that capture the spatial semantics of the data.

For this experimental evaluation, the topological dependency constraints are:

(i) $\overline{\forall}(\mathbf{St}(x_1, y_1, z_1, g_1) \wedge \mathbf{St}(x_2, y_2, z_2, g_2) \wedge x_1 = x_2 \rightarrow \mathsf{Equal}(g_1, g_2))$
(ii) $\overline{\forall}\,\mathbf{Co}(x_1, y_1, z_1, g_1) \wedge \mathbf{Co}(x_2, y_2, z_2, g_2) \wedge x_1 = x_2 \rightarrow \mathsf{Equal}(g_1, g_2))$
(iii) $\overline{\forall}\,\mathbf{Pl}(x_1, y_1, z_1, g_1) \wedge \mathbf{Pl}(x_2, y_2, z_2, g_2) \wedge x_1 = x_2 \rightarrow \mathsf{Equal}(g', g))$
(iv) $\overline{\forall}(\mathbf{Lm}(x_1, y_1, z_1, g_1) \wedge \mathbf{Lm}(x_2, y_2, z_2, g_2) \wedge x_1 = x_2 \rightarrow \mathsf{Equal}(g_1, g_2))$

[3] http://www.census.gov/geo/maps-data/data/tiger.html

Table 5 Data of the experimental evaluation

Schema	FKeys	Tuples	Size (Mb)	Index (Mb)
St(sid,name,division,geom)	–	56	15	15
Co(cid,sid,name,geom)	sid	3234	129	128
Pl(pid, sid,name,geom)	sid	28556	206	134
Lm(lid,sid,name,geom)	sid	902664	141	51
Ct(ctid,sid,cid,name,geom)	cid,sid	74133	650	469
Cb(cbid,ctid,sid,cid,name,geom)	ctid,cid,sid	11709632	13000	2801
Ua(uid,name,geom)	–	3601	139	132
Rd(prid,name,geom)	–	11574	51	20

(v) $\overline{\forall}(\mathbf{Ct}(x_1, y_1, z_1, w, 1, g_1) \wedge \mathbf{Ct}(x_2, y_2, z_2, w2, g_2) \wedge x_1 = x_2 \rightarrow \mathsf{Equal}(g_1, g_2))$

(vi) $\overline{\forall}(\mathbf{Cb}(x_1, y_1, z_1, w, 1, v_1 g_1) \wedge \mathbf{Cb}(x_2, y_2, z_2, w_2, v_2, g_2) \wedge x_1 = x_2 \rightarrow \mathsf{Equal}(g_1, g_2))$

(vii) $\overline{\forall}(\mathbf{Ua}(x_1, y_1, g_1) \wedge \mathbf{Ua}(x_2, y_2, g_2) \wedge x_1 = x_2 \rightarrow \mathsf{Equal}(g_1, g_2))$

(viii) $\overline{\forall}(\mathbf{Rd}(x_1, y_1, g_1) \wedge \mathbf{Rd}(x_2, y_2, g_2) \wedge x_1 = x_2 \rightarrow \mathsf{Equal}(g_1, g_2))$

(ix) $\overline{\forall}(\mathbf{St}(x_1, y_1, z_1, g_1) \wedge \mathbf{St}(x_2, y_2, z_2, g_2) \wedge x_1 \neq x_2 \rightarrow \mathsf{Disjoint}(g_1, g_2) \vee \mathsf{Meet}(g_1, g_2))$

(x) $\overline{\forall}(\mathbf{Co}(x_1, y_1, z_1, g_1) \wedge \mathbf{Co}(x_2, y_2, z_2, g_2) \wedge x_1 \neq x_2 \rightarrow \mathsf{Disjoint}(g_1, g_2) \vee \mathsf{Meet}(g_1, g_2))$

(xi) $\overline{\forall}(\mathbf{Ct}(x_1, y_1, z_1, w_1, g_1) \wedge \mathbf{Ct}(x_2, y_2, z_2, w_2, g_2) \wedge x_1 \neq x_2 \rightarrow \mathsf{Disjoint}(g_1, g_2) \vee \mathsf{Meet}(g_1, g_2))$

(xii) $\overline{\forall}(\mathbf{Cb}(x_1, y_1, z_1, w_1, v_1, g_1) \wedge \mathbf{Cb}(x_2, y_2, z_2, w_2, v_2, g_2) \wedge x_1 \neq x_2 \rightarrow \mathsf{Disjoint}(g_1, g_2) \vee \mathsf{Meet}(g_1, g_2))$

Several of these constraints can be automatically derived from primary key conditions. Indeed, constraints (i) to (viii) are automatically derived. The designer is in charge of expressing additional topological constraints as the application requires, such as constraints (ix) to (xii).

Topological referential constraints are then used to impose a spatial semantics to the foreign key constraints. For example, if there is a county, then there exists a state that spatially contains the county. This also applies for places and landmarks with respect to states, and for census blocks with respect to census tracks, as the following constraints express:

(ix) $\forall x_1 y_1 z_1 g_1(\mathbf{Co}(x_1, y_1, z_1, g_1) \rightarrow \exists x_2 y_2 z_2 g_2(\mathbf{St}(x_2, y_2, z_2, g_2) \wedge y_1 = x_2 \wedge \mathsf{Within}(g_1, g_2)))$

(x) $\forall x_1 y_1 z_1 g_1(\mathbf{Pl}(x_1, y_1, z_1, g_1) \rightarrow \exists x_2 y_2 z_2 g_2(\mathbf{St}(x_2, y_2, g_2) \wedge y_1 = x_2 \wedge \mathsf{Within}(g_1, g_2)))$

(xi) $\forall x_1 y_1 z_1 g_1(\mathbf{Lm}(x_1, y_1, z_1, g_1) \rightarrow \exists x_2 y_2 z_2 g_2(\mathbf{St}(x_2, y_2, g_2) \wedge y_1 = x_2 \wedge \mathsf{Within}(g_1, g_2)))$

(xii) $\forall x_1 y_1 z_1 g_1(\mathbf{Cb}(x_1, y_1, z_1, w_1, v_1, g_1) \rightarrow \exists x_2 y_2 z_2 g_2(\mathbf{Ct}(x_2, y_2, z_2, w_2, g_2) \wedge y_1 = x_2 \wedge \mathsf{Within}(g_1, g_2)))$

(xiii) $\forall x_1 y_1 z_1 g_1(\mathbf{Ct}(x_1, y_1, z_1, w_1, g_1) \rightarrow \exists x_2 y_2 z_2 g_2(\mathbf{Co}(x_2, y_2, z_2, w_2, g_2) \wedge z_1 = x_2 \wedge \mathsf{Within}(g_1, g_2)))$

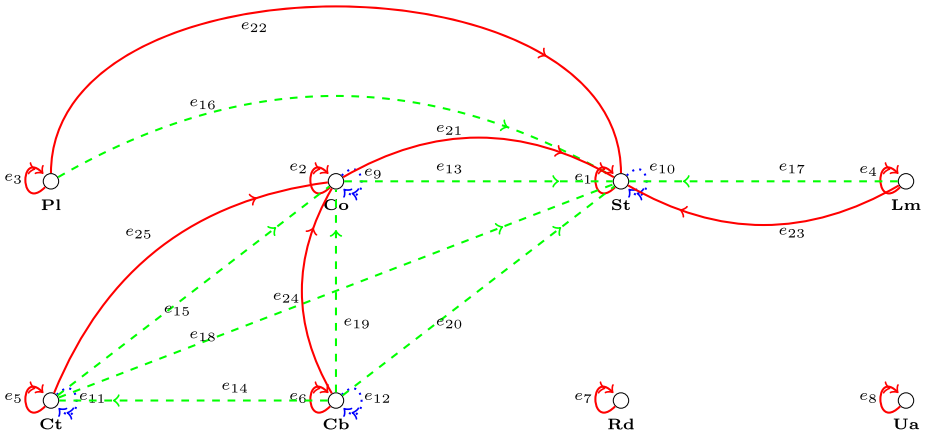


Fig. 3 The composition-dependency graph for the experimental evaluation (Red edges as solid lines, green edges as dashed lines and blue edges as dotted lines)

Figure 3 and Table 6 show the CDG with the useful TDs and ETDs derived from the constraints, where the first eight edges can be automatically derived from primary key conditions, edges e_9 to e_{17} are the 9 constraints that required to be specified and the last edges e_{18} to e_{25} are derived from the algorithm.

We use 15 join queries, where queries q_1 to q_8 are basic queries to analyze scalability in terms of number of tuples, and queries q_9 to g_{15} are queries that exploit different cases where the optimization process may have an effect. Because the data satisfy the integrity constraints, there should be no answers to q_{11} and q_{12}. Table 7 shows the original queries q_i and how they are rewritten into q_i^o using the preprocessing based on the integrity constraints and the derived CDG.

The experiments compared the time cost of processing the original query and the query after preprocessing or rewritten on a database management system. To distinguish the effect of spatial indexing used by the DBMS, we also tested processing cost with and without indexing.

In addition, and following a partition-based optimization [17], we compared our approach with a strategy that partitions a unique table in tables by states such that join queries apply over smaller number of tuples. This strategy can add additional cost by applying queries to several tables, but as we will see with the experimental results, it provides a basic optimization useful in several queries.

Notice that, although the time cost depends on the algorithm for join queries used by the SDBMS, by comparing the cost of answering the query with and without preprocessing we are just evaluating the effect of rewritten, where the spatial joins are solved by the same algorithm for the original and modified query.

The experiments ran in a computer with Intel Intel Xeon E3-1220 v5, 64GB DDR4 memory, and PostgreSQL 9.5.12 extended with PostGIS 2.2.1.

Table 8 shows the cost of processing queries in Table 7, where RW is the time of preprocessing plus query rewriting in ms, q_i is the time in ms of solving the original query, $q_{i_{rw}}$ is the time used to solve the rewritten query q_i, and $q_{i_{part}}$ is the time to solve the query q_i by the strategy of data partition to process the query.

To clarify how well the optimization strategy works with respect to the original query, Fig. 4 shows a graph of the time ratio between a query optimized by rewritten ($q_{i_{\{rew\}}}$) and by partitioning ($q_{i_{\{part\}}}$) with respect to the original query (q_i), without (no_index) or with ($index$) spatial indexing.

Table 6 Description of edges of graph in Fig. 3

E	λ_r	λ_p	λ_s	λ_c	λ_t
e_1	[**St**$(w, \bar{m}, g_s)$, **St**$(w', \bar{m'}, g'_s)$]	True	red	$w = w'$	Equal(g_s, g'_s)
e_2	[**Co**$(v, \bar{s}, g_c)$, **Co**$(v', \bar{s'}, g'_c)$]	True	red	$v = v'$	Equal(g_c, g'_c)
e_3	[**Pl**$(u, \bar{c}, g_p)$, **Pl**$(u', \bar{c'}, g'_p)$]	True	red	$u = u'$	Equal(g_p, g'_p)
e_4	[**Lm**$(w, \bar{m}, g_l)$, **Lm**$(w', \bar{m'}, g'_l)$]	True	red	$w = w'$	Equal(g_l, g'_l)
e_5	[**Ct**$(w, \bar{m}, g_l)$, **Ct**$(w', \bar{m'}, g'_l)$]	True	red	$w = w'$	Equal(g_l, g'_l)
e_6	[**Cb**$(w, \bar{m}, g_l)$, **Cb**$(w', \bar{m'}, g'_l)$]	True	red	$w = w'$	Equal(g_l, g'_l)
e_7	[**Rd**$(w, \bar{m}, g_l)$, **Rd**$(w', \bar{m'}, g'_l)$]	True	red	$w = w'$	Equal(g_l, g'_l)
e_8	[**Ua**$(w, \bar{m}, g_l)$, **Ua**$(w', \bar{m'}, g'_l)$]	True	red	$w = w'$	Equal(g_l, g'_l)
e_9	[**Co**$(v, \bar{s}, g_c)$, **Co**$(v', \bar{s'}, g'_c)$]	True	blue	$v \neq v'$	Meet$(g_c, g'_c) \vee$ Disjoint(g_p, g'_p)
e_{10}	[**St**$(w, \bar{m}, g_s)$, **St**$(w', \bar{m'}, g'_s)$]	True	blue	$w \neq w'$	Meet$(g_s, g'_s) \vee$ Disjoint(g_p, g'_p)
e_{11}	[**Ct**$(w, \bar{m}, g_s)$, **Ct**$(w', \bar{m'}, g'_s)$]	True	blue	$w \neq w'$	Meet$(g_s, g'_s) \vee$ Disjoint(g_p, g'_p)
e_{12}	[**Cb**$(w, \bar{m}, g_s)$, **Cb**$(w', \bar{m'}, g'_s)$]	True	blue	$w \neq w'$	Meet$(g_s, g'_s) \vee$ Disjoint(g_p, g'_p)
e_{13}	[**Co**$(c, s, \bar{v}, g_c)$, **St**$(w, \bar{m}, g_s)$]	True	green	$s = w$	Within(g_c, g_s)
e_{14}	[**Cb**$(w, t, \bar{v}, g_b)$, **Ct**$(v, \bar{n}, g_t)$]	True	green	$t = v$	Within(g_b, g_t)
e_{15}	[**Ct**$(u, v, w, \bar{c}, g_t)$, **Co**$(u', \bar{c'}, g'_c)$]	True	green	$w = u'$	Within(g_t, g_c)
e_{16}	[**Pl**(p, s, n, g_c), **St**$(w, \bar{m}, g_s)$]	True	green	$s = w$	Within(g_c, g_s)
e_{17}	[**Lm**(p, s, n, g_c), **St**$(w, \bar{m}, g_s)$]	True	green	$s = w$	Within(g_c, g_s)
e_{18}	[**Ct**(u, v, b, t, q_t), **St**(w, n, d, g_s)]	**Co**(c, s, m, g_c)	green	$b = c \wedge s = w$	Within(g_t, g_s)
e_{19}	[**Cb**(u, t, s, w, k, g_t), **Co**(c, p, n, g_c)]	**Ct**(v, m, b, q, g_c)	green	$t = v \wedge b = c$	Within(g_t, g_c)
e_{20}	[**Cb**(u, t, s, w, k, g_b), **St**(c, p, n, g_s)]	**Ct**$(v, m, b, q, g_t) \wedge$ **Co**(c', p', n', g_c)	green	$t = v \wedge b = c \wedge p' = c$	Within(g_b, g_s)
e_{21}	[**Co**(c, t, n, g_c), **St**(s, p, m, g_s)]	**St**(s', p', m', g'_s)	blue	$s \neq s' \wedge t = s'$	Meet$(g_c, g_s) \vee$ Disjoint(g_c, g_s)
e_{22}	[**Pl**(c, t, n, g_p), **St**(s, p, m, g_s)]	**St**(s', p', m', g'_s)	blue	$s \neq s' \wedge t = s'$	Meet$(g_p, g_s) \vee$ Disjoint(g_p, g_s)
e_{23}	[**Lm**(c, t, n, g_l), **St**(s, p, m, g_s)]	**St**(s', p', m', g'_s)	blue	$s \neq s' \wedge t = s'$	Meet$(g_l, g_s) \vee$ Disjoint(g_l, g_s)
e_{24}	[**Cb**(c, t, n, q, u, g_b), **Ct**(s, p, m, v, g_t)]	**Ct**(s', p', m', v', g'_t)	red	$s \neq s' \wedge t = s'$	Meet$(g_c, g_b) \vee$ Disjoint(g_c, g_t)
e_{25}	[**Ct**(c, t, n, q, g_t), **Co**(s, p, m, g_c)]	**Co**(s', p', m', g'_c)	red	$s \neq s' \wedge q = s'$	Meet$(g_t, g_c) \vee$ Disjoint(g_t, g_c)

Table 7 Reformulation of queries for experiments

Original q_i and final q_i^o queries
$q_1(i,i',g,g') : \exists(\ \mathbf{St}(i,n,d,g) \wedge\ \mathbf{St}(i',n',d',g') \wedge \mathsf{Meet}(g',g))$
$q_1^o(i,i',g,g') : \exists(\ \mathbf{St}(i,n,d,g) \wedge\ \mathbf{St}(i',n',d',g') \wedge i \neq i' \wedge \mathsf{Meet}(g',g))$
$q_2(i,i',g,g') : \exists(\ \mathbf{Rd}(i,n,g) \wedge\ \mathbf{Rd}(i',n',g') \wedge \mathsf{Meet}(g',g))$
$q_2^o(i,i',g,g') : \exists(\ \mathbf{Rd}(i,n,g) \wedge\ \mathbf{Rd}(i',n',g') \wedge i \neq i \wedge \mathsf{Meet}(g',g))$
$q_3(i,i',g,g') : \exists(\ \mathbf{Co}(i,n,d,g) \wedge\ \mathbf{Co}(i',n',d',g') \wedge \mathsf{Meet}(g',g))$
$q_3^o(i,i',g,g') : \exists(\ \mathbf{Co}(i,n,d,g) \wedge\ \mathbf{Co}(i',n',d',g') \wedge i \neq i' \wedge \mathsf{Meet}(g',g))$
$q_4(i,i',g,g') : \exists(\ \mathbf{Ua}(i,n,g) \wedge\ \mathbf{Ua}(i',n',g') \wedge \mathsf{Meet}(g',g))$
$q_4^o(i,i',g,g') : \exists(\ \mathbf{Ua}(i,n,g) \wedge\ \mathbf{Ua}(i',n',g') \wedge i \neq i' \wedge \mathsf{Meet}(g',g))$
$q_5(i,i',g,g') : \exists(\ \mathbf{Lm}(i,n,d,g) \wedge\ \mathbf{Lm}(i',n',d',g') \wedge \mathsf{Meet}(g',g))$
$q_5^o(i,i',g,g') : \exists(\ \mathbf{Lm}(i,n,d,g) \wedge\ \mathbf{Lm}(i',n',d',g') \wedge i \neq i' \wedge \mathsf{Meet}(g',g))$
$q_6(i,i',g,g') : \exists(\ \mathbf{Pl}(i,n,d,g) \wedge\ \mathbf{Pl}(i',n',d',g') \wedge \mathsf{Meet}(g',g))$
$q_6^o(i,i',g,g') : \exists(\ \mathbf{Pl}(i,n,d,g) \wedge\ \mathbf{Pl}(i',n',d',g') \wedge i \neq i' \wedge \mathsf{Meet}(g',g))$
$q_7(i,i',g,g') : \exists(\ \mathbf{Ct}(i,n,d,e,g) \wedge\ \mathbf{Ct}(i',n',d',e',g') \wedge \mathsf{Meet}(g',g))$
$q_7^o(i,i',g,g') : \exists(\ \mathbf{Ct}(i,n,d,e,g) \wedge\ \mathbf{Ct}(i',n',d',e',g') \wedge i \neq i' \wedge \mathsf{Meet}(g',g))$
$q_8(i,i',g,g') : \exists(\ \mathbf{Cb}(i,n,d,e,f,g) \wedge\ \mathbf{Cb}(i',n',d',e',f'g') \wedge \mathsf{Meet}(g',g))$
$q_8^o(i,i',g,g') : \exists(\ \mathbf{Cb}(i,n,d,e,f,g) \wedge\ \mathbf{Cb}(i',n',d',e',f',g') \wedge i \neq i' \wedge \mathsf{Meet}(g',g))$
$q_9(i,i',g,g') : \exists(\ \mathbf{St}(i,n,d,g) \wedge\ \mathbf{Co}(i',s,n',g') \wedge \mathsf{Within}(g',g))$
$q_9^o(i,i',g,g') : \exists(\ \mathbf{St}(i,n,d,g) \wedge\ \mathbf{Co}(i',s,n',g') \wedge i = s)$
$q_{10}(i,i',g,g') : \exists(\ \mathbf{St}(i,n,d,g) \wedge\ \mathbf{Pl}(i',c,n',g') \wedge \mathsf{Within}(g',g))$
$q_{10}^o(i,i',g,g') : \exists i''sn''g''(\ \mathbf{St}(i,n,d,g) \wedge\ \mathbf{Pl}(i',c,n',g') \wedge \mathbf{Co}(i''s,n'',g'') \wedge i = s \wedge i'' = c)$
$q_{11}(i,i',g,g') : \exists(\ \mathbf{Co}(i,s,n,g) \wedge\ \mathbf{Co}(i',s',n',g') \wedge \mathsf{Overlap}(g',g))$
$q_{11}^o(i,i',g,g') : \mathsf{q}_\emptyset$ (empty result)
$q_{12}(i,i',g,g') : \exists(\ \mathbf{St}(i,n,d,g) \wedge\ \mathbf{Pl}(i',n',d',g') \wedge \mathsf{Overlap}(g',g))$
$q_{12}^o(i,i',g,g') : \mathsf{q}_\emptyset$ (empty result)
$q_{13}(i,i',g,g') : \exists(\ \mathbf{St}(i,n,d,g) \wedge\ \mathbf{Ct}(i',n',d',g') \wedge \mathsf{Within}(g',g))$
$q_{13}^o(i,i',g,g') : \exists(\ \mathbf{St}(i,n,d,g) \wedge\ \mathbf{Ct}(i',n',d',g') \wedge\ \mathbf{Co}(i'',s',n'',g'') \wedge i'' = n' \wedge s' = i)$
$q_{14}(i,i',g,g') : \exists(\ \mathbf{St}(i,n,d,g) \wedge\ \mathbf{Ct}(i',n',d',e',g') \wedge \mathsf{Meet}(g',g))$
$q_{14}^o(i,i',g,g') : \exists(\ \mathbf{St}(i,n,d,g) \wedge\ \mathbf{Ct}(i',n',d',e',g') \wedge \mathbf{C}o(i'',s',n'',g'') \wedge i'' = n' \wedge s' = i$ $\wedge \mathsf{Meet}(g,g''))$
$q_{15}(i,i',g,g') : \exists(\ \mathbf{Co}(i,n,d,g) \wedge\ \mathbf{Cb}(i',n',d',e',f',g') \wedge \mathsf{Within}(g',g))$
$q_{15}^o(i,i',g,g') : \exists(\ \mathbf{Co}(i,n,d,g) \wedge\ \mathbf{Cb}(i',n',d',e',f',g') \wedge\ \mathbf{Ct}(i'',s',n'',d'',g'')$ $\wedge n'' = i \wedge n' = i'' \wedge \mathsf{Meet}(g,g''))$

The experiments show that by replacing the spatial by classical joins reduces significantly the processing cost of the query. The use of a spatial indexing structure plus the preprocessing strategy improves, or at least keeps, the same time cost for query processing. The experimental evaluation also shows that query rewriting outperforms the partition-based strategy in most cases. In cases when this does not happen (queries 7 and 8), both strategies outperform the cost of the original query. In these cases, the thematic constraints introduced by the query rewriting ($x \neq y$) are not able to filter out enough tuples to compensate the optimization of running spatial joins with less number of tuples.

From queries 9 to 15, the results highlight the advantage of the proposal. In these cases there is a direct or indirect relationship through composition that makes possible to filter out tuples based on a thematic constraint and an additional relational predicate. For example,

Table 8 Cost of query processing in *ms*

Query	RW	No Index			Index		
i		q_i	$q_{i_{rew}}$	$q_{i_{part}}$	q_i	$q_{i_{rew}}$	$q_{i_{part}}$
1	7	15987	8152	15987	14782	7512	14782
2	7	205846	172898	172898	82391	63851	82391
3	8	205205	171200	220445	123878	69189	130798
4	7	141440	71530	141440	103548	31316	103548
5	8	44721400	37469442	44721400	71884	65321	70648
6	9	1947167	1512667	1746215	144913	60884	99955
7	6	9425922	7260801	6421350	662093	377368	302666
8	8	164417336	125269852	90314947	84006668	26882090	19421200
9	9	15808	1	13	14390	1	10
10	9	163767	24	30	10378	28	41
11	6	192828	0	201346	36825	0	40379
12	6	207481	0	201647	203621	0	203004
13	43	302827	389	106	39617	416	107
14	45	386505	251329	380547	388283	255221	374017
15	44	17460515	3715014	13004515	14745648	3715647	10220999

the query that relates census blocks with counties, which can be optimized by introducing an additional condition that relates a census block with its census tract that is also related to a county. The original foreign-key constraints relate census blocks with census tracts, and census tracts with counties. Then, by composition of these constraints, we introduce additional conditions that reduce the tuples over which to apply spatial joins. This is a simple strategy that can be systematically applied without consideration of the data itself in the tables.

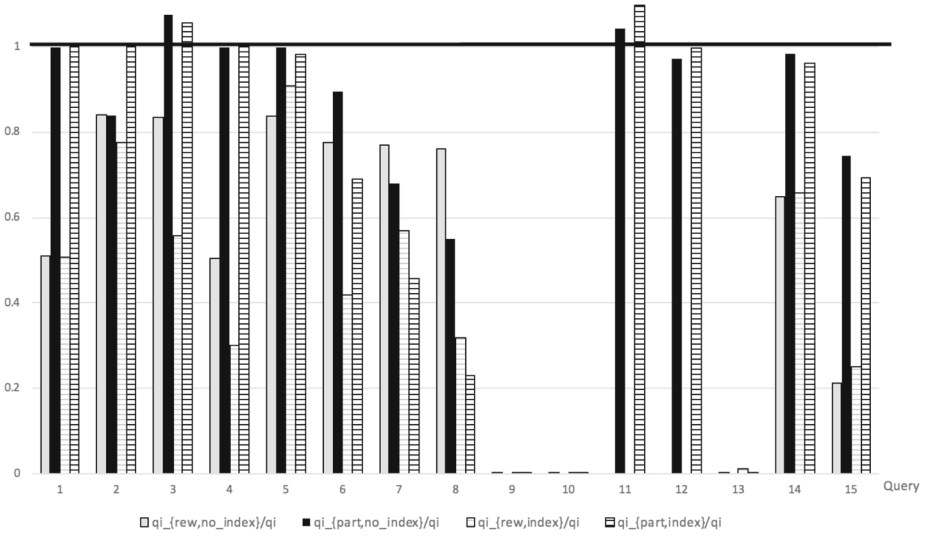


Fig. 4 Time ratio between rewritten- and partitioning-based optimization strategies with respect to the original query without and with spatial indexing

It is also worth to notice that using spatial indexing is not always the best option. In particular, when using a spatial join with overlap (queries 11 and 12), where the index is not able to filter out tuples and make the query more selective.

5 Conclusions

The work in this paper presents a strategy for rewriting spatial join queries in terms of the semantic information given by integrity constraints. The goal was to avoid the computation of spatial joins when possible by using topological dependencies and topological referential constraints to extract the topological relations that spatial attributes satisfy in a consistent database. This strategy can complement state-of-the-art methods for spatial joins using indexing structures. The work shows that a strategy like this can be useful even for case with simple constraints defined by primary and foreign keys.

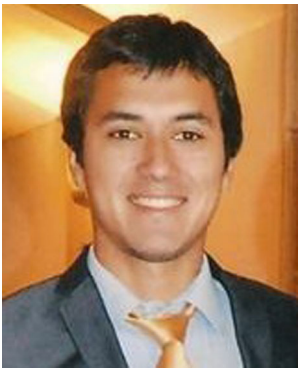
As future work, we would like to study the effect of null values in the query processing and include new integrity constraints, as check constraints defined in [4]. The consideration of other types of spatial queries is also possible; however, our preliminary results show that for range queries most of the optimization can be done within the algorithm of searching using the existing indexing structures.

Acknowledgements This work has been funded by Fondecyt 1170497 and by the Millennium Institute for Foundational Research on Data, Chile.

References

1. Popa L, Deutsch A, Tannen V (2005) Query Reformulation with Constraints
2. Aref WalidG, Samet Hanan (1991) Optimization for spatial query processing. In: 17th international conference on very large data bases, september 3-6, 1991, barcelona, catalonia, spain, proceedings, pp 81–90
3. Bogorny V, Engel PM, Alvares LOC (2005) Towards the reduction of spatial join for knowledge discovery in geographic databases using geo-ontologies and spatial integrity constraints. In: Proc of the ECML/PKDD second workshop on knowledge discovery and ontologies (KDO'2005), pp 51–58
4. Bravo L, Rodríguez AM (2012) Formalization and reasoning about spatial semantic integrity constraints. Data Knowl Eng 72:63–82
5. Clementini E, Sharma J, Egenhofer MJ (1994) Modelling topological spatial relations: strategies for query processing. Comput Graph 18(6):815–822
6. Deutsch A, Ludäscher B, Nash A (2007) Rewriting queries using views with access patterns under integrity constraints. Theor Comput Sci 371(3):200–226
7. Deutsch A, Popa L, Tannen V (1999) Physical data independence, constraints, and optimization with universal plans. In: VLDB, pp 459–470
8. Deutsch A, Popa L, Tannen V (2006) Query reformulation with constraints. SIGMOD Rec 35(1):65–73
9. Egenhofer M, Al-Taha K (1992) Reasoning about gradual change of topological relationships. In: Frank A, Campari I, Formentini U (eds) Theories and methods of spatio-temporal reasoning in geographic space, LNCS 636, Springer, pp 196–219
10. Egenhofer MJ, Franzosa RD (1991) Point-set topological spatial relations. Int J Geogr Inf Syst 5:161–174
11. Egenhofer MJ, Herring J (1990) Categorizing binary topological relations between regions, lines and points in geographic databases, the 9-intersection. Formalism and its Use for Natural Language Spatial Predicates Santa Barbara CA National Center for Geographic Information and Analysis Technical Report 94:1–28
12. Egenhofer MJ (1994) Deriving the composition of binary topological relations. J Vis Lang Comput 5(2):133–149

13. Grant J, Gryz J, Minker J, Raschid L (2000) Logic-based query optimization for object databases. IEEE Trans Knowl Data Eng 12(4):529–547
14. Jacox EH, Samet H (2007) Spatial join techniques. ACM Trans Database Syst 32(1):7
15. Lee SG, Henschen LJ, Chun J, Lee T (2000) Identifying relevant constraints for semantic query optimization. Inf Softw Technol 42(13):899–914
16. Mamoulis N, Theodoridis Y, Papadias D (2005) Spatial joins: algorithms, cost models and optimization techniques. In: Manolopoulos Y, Papadopoulos A, Vassilakopoulos M (eds) Spatial databases: technologies, techniques and trends, idea group, pp 155–184
17. Maher MJ, Wang J (2000) Optimizing queries in extended relational databases. In: DEXA, vol 1873 of lecture notes in computer science. Springer, pp 386–396
18. OpenGis (1999) Opengis simple features specification for sql. Technical report Open GIS Consortium
19. Randell DA, Cui Z, Cohn AG (1992) A spatial logic based on regions and connection. In: KR, pp 165–176
20. Ray S, Simion B, Brown AD (2011) Jackpine: a benchmark to evaluate spatial database performance. In: Proceedings of the 27th international conference on data engineering. ICDE 2011, April 11-16, 2011, Hannover, Germany, pp 1139–1150
21. Stock O (1997) Spatial and temporal reasoning. Kluwer Acaddemic Publishers
22. Worboys M (1992) A geometric model for planar geographical objects. Int J Geogr Inf Syst 6(5):353–372



Eduardo Mella received his MSc in Computer Science from the Universidad de Concepción in 2016. He now works as a software engineer for Woodtech MS Company, Chile.

M. Andrea Rodríguez is a full professor at the Department of Computer Science, Faculty of Engineering of the Universidad de Concepción, Chile. She received a MSc and PhD in Spatial Information Science and Engineering from the University of Maine in 1997, and 2000, respectively. Her research interests include spatio-temporal databases and information retrieval.

Loreto Bravo received her Engineering Degree in 2000 from P. Universidad Católica de Chile and PhD in Computer Science from Carleton University in 2007. She was a research fellow of the Database Group at University of Edinburgh from 2007 to 2008. She is currently director of the Institute of Data Science of the Universidad del Desarrollo in Santiago, Chile. Her research interests include database theory, database consistency, XML Databases and logic programming.

Diego Gatica is a student of the MSc in Computer Science of the Universidad de Concepción.