

SPATIAL QUERY PROCESSING USING GENERALIZED FILTER

XIAOFANG ZHOU

*Department of Computer Science and Electrical Engineering, University of Queensland
Brisbane, Queensland 4072, Australia*

XUEMIN LIN

*School of Computer Science and Engineering, University of New South Wales
Sydney, New South Wales 2052, Australia*

CHENGFEI LIU

*School of Computing Sciences, University of Technology Sydney
Sydney, New South Wales 2009, Australia*

JINLI CAO

*Department of Mathematics and Computing, University of Southern Queensland
Toowoomba, Queensland 4350, Australia*

Received (to be inserted
Revised by Publisher)

Spatial data, ranging from various land information data to different types of environmental data, are typically collected and used by different custodians. The full benefits of using spatial data can be achieved by combining the data from different sources covering a common region. Due to organizational, political and technical reasons, it is unrealistic to physically integrate the vast amount of spatial data managed by different systems in different organizations. A practical approach is to provide interoperability to support multi-site data queries. In this paper, we study the performance aspect of complex spatial query processing. We propose a framework for processing queries with multiple spatial and aspatial predicates using data from multiple sites. Using a new concept called generalized filter, a query is processed in three steps. First, an aspatial filter that incorporates some conditions derived from spatial predicates is used to find a set of candidates, which is a superset of the final query results. Then, the candidates are manipulated and a refinement step is executed following an optimized candidate sequence. Finally, a post-processing step is used to handle spatial expressions in query results. The focus of this paper is to generate enhanced filters in order to minimize the need of transferring and processing complex spatial data.

Keywords: Spatial databases; Query processing; Performance; Filter and refine; Complex queries; Predicate transformation.

1. Introduction

Government agencies and large organizations have invested a huge amount of effort in the past in collecting spatial data. Large amounts of spatial data, which are comprehensive and extensive in terms of geographical coverage, time and thematic layers, become an important corporate asset. Due to historical reasons, these data

are typically owned, managed and used by different groups. It has been realized that a much higher return for investment in collecting spatial data can be achieved through a horizontal approach². That is, an application can often be best served by using parts of spatial data from different collections for the same region. Comprehensive, up-to-date and consistent data are vital to better decision making. A new residential development project, for example, may need to access spatial data such as land titles and zoning information (from the Land Department), infrastructure information (from the Transport Department and utilities companies), demographic data (from the Bureau of Statistics) and environmental information (from several Departments such as Environment and Natural Resources).

One of the major barriers of spatial data sharing is heterogeneous data models and formats. This problem is the main motivation for spatial data standardization initiatives (e.g., OpenGIS, <http://www.opengis.org>). Progresses in this direction make spatial data from different sources searchable and exchangeable. A common practice in sharing spatial data at this stage is to use an Internet-based spatial data directory service to locate a dataset by some standard metadata attributes (such as the ANZLIC metadata, <http://www.anzlic.org.au/metaelem.htm>). Then, a copy of the dataset is obtained by ordering tapes or CDs, rather than downloading on the Internet due to the large sizes of spatial data sets. There are several obvious problems with installing a local copy of other people's data. These data cannot be fully utilized due to lack of knowledge and software that sometimes only the owner has. Given the very large sizes of spatial data (from gigabytes to terabyte), duplication is costly. It is also difficult to maintain data consistency when there are multiple copies around.

The ideal way of sharing spatial data, as in sharing other types of data, is to get the required data on-line and on-demand. In the previous urban planning example, it is a waste of time and money to buy data and software for the land information of the entire city. A much better way is to select land parcels when the application needs them. For example, if the land developer is interested in redeveloping a number of vacant blocks into medium-density residential blocks, s/he needs to check issues such as if these land blocks are classified as "agriculture A" (top grade agriculture land cannot be used for other purposes by legislation) or "polluted" (not suitable for residential purpose), and the distance to major arterial roads, major shopping centers, schools and recreational facilities. Instead of having to purchase many large and complex datasets and to fully understand the data to be able to use them, one may use a simple query in an SQL-like language (see Section 2) if such a query can be processed automatically and efficiently across the sites where relevant data are available. For this particular example, only the amalgamated boundary of the land blocks needs to be passed to other data sites, and the results returned from these sites are very simple, either Boolean or numerical in this case. The developer only needs to pay a small amount of money for this query². When this paradigm of spatial data sharing becomes possible, spatial data will reach more people and find more applications.

To make this happen, a number of technical issues need to be solved. Most of the problems faced by the multidatabase approach are also present here, such as the need for a global data model and query language, and the problems caused by heterogeneous local schemas and possible semantic conflicts⁹. We believe that the data integration problem for spatial databases is somewhat simpler than integrating other types of data, as there is a common underlying reference object for all spatial databases. That is, these spatial data are all about the same region, and their coordinate systems can be mapped from one to another (for example, using the longitude and latitude). In this paper, we assume that each data site uses a modern relational database management system (DBMS). Their capacity for spatial data processing can vary only supporting simple spatial object retrieval to a fully-fledged object-relational DBMS where a spatial data type is treated no differently to other data types such as numbers. Our objective in this paper is to take full advantage of the relational DBMS, to simplify spatial query processing and reduce the amount of spatial data to be exchanged among data sites. To get spatial data from remote sites is not only costly, but also forces a spatial DBMS designed to handle one type of spatial data to be able to deal with other types of spatial data. Therefore, when translating a global query into a sequence of local database operations (called execution plans), we attempt to avoid using spatial data from other sites as much as possible. When it is necessary to use remote spatial data, these data are used in a simple way in order to lower the entry requirement for a site's spatial processing capacity.

In this paper, we propose a framework for processing queries with multiple spatial and aspatial predicates using data from multiple sites. Using a new concept called generalized filter, a query is processed in three steps. First, an aspatial filter that incorporates some conditions derived from spatial predicates is used to find a set of candidates, which is a superset of the final query results. Then, the candidates are manipulated and a refinement step is executed following an optimized candidate sequence. Finally, a post-processing step is used to handle spatial expressions in the query results. The focus of this paper is to generate enhanced filters in order to minimize the need for transferring and processing complex spatial data. We achieve this by exploring rich semantic relationships among spatial predicates.

The rest of the paper is organized as follows. In Section 2, we describe the data model and query language to be used in this paper. Also in this Section, we introduce a concept called generalized filter and use it as a framework for complex spatial query optimization. In Section 3, we examine semantic relationships among spatial operations. Our findings are applied to spatial query transformation in Section 4. We conclude this paper in Section 5.

2. Preliminaries

2.1. System Architecture, Data Model and Query Language

In this section, we discuss the data model and the query language to be considered

in this paper. The purpose of this paper is to introduce new techniques for spatial query processing, rather than discussing spatial data models or spatial query languages. The data model and query language given in this section are simplified and generic.

An SDBMS supports storage and manipulation of spatial objects such as points, lines and polygons¹⁰. Several aspatial attributes and spatial attributes can be used to describe a spatial object. For example, a spatial object **property** can have aspatial attributes such as street address, owner's name and spatial attributes such as its boundary. In the object-relational model, a spatial data type, implemented as an abstract data type or a user-defined data type, is treated no differently to other conventional data types. A database table consists of a collection of attributes. An attribute is of either spatial or aspatial data type. Generic spatial data types include *point*, *line* and *polygon*. A *spatial table* is a database table with at least one spatial attribute. Below are two spatial tables, both have a spatial attribute **boundary** whose data type is polygon.

```
lake(name, boundary)
factory(name, address, boundary, type)
```

In general, a spatial attribute or spatial constant can be used in a query where an aspatial attribute or constant can appear.

A set of spatial operations can be defined on spatial data types. There is no standard algebra defined on spatial data, although several proposals have been made⁷. Thus, there is no standard set of spatial operations. Instead, what spatial operations to use and their exact semantic meanings depend heavily on the target application domain. In this paper we use the eight spatial relations: **disjoint**, **contains**, **inside**, **equals**, **meet**, **covers**, **covered_by** and **overlap** (the reasons of choosing these eight relations and their precise definition can be found in⁵). Note that **covers** (resp., **covered_by**) differs to **contains** (resp., **inside**) by the condition that the boundaries touch each other or not. These relations are illustrated in Figure 1. However, we do allow users to alter the meaning of these operations, and to define new operations.

We consider an SQL-like query language in a simplified form as shown below:

```
SELECT {target list}
FROM   {table list}
WHERE  {condition}
```

or equivalently

$$\pi_{\{target\ list\}}\sigma_{condition}(R_1 \times R_2 \times \cdots R_n)$$

The target list is a list of expressions, with attribute names, constants, built-in functions, arithmetic and other operations. The built-in functions include, in addition to those normal aspatial functions such as **min**, **max**, **average** and **count**, other functions which can take spatial attributes as parameters (e.g., **area**, **distance**, and **perimeter**) (**count** can be applied on both spatial and aspatial attributes). When a spatial operation is used in a target expression, the data type of the expression can be data-dependant. For example, the intersection of two polygons can

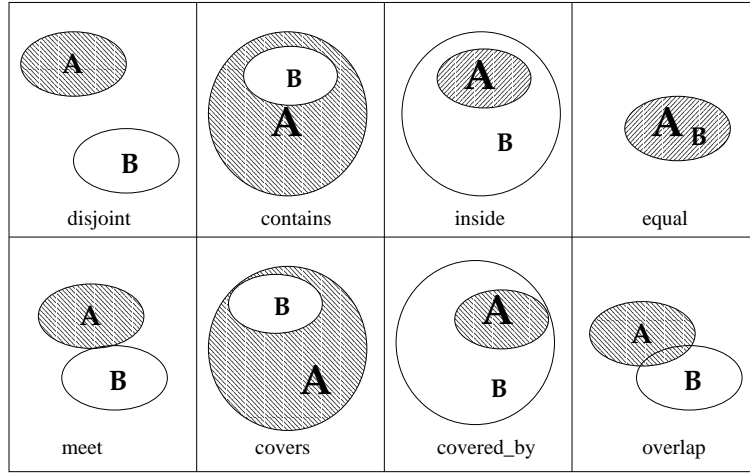


Fig. 1. Illustration of the eight spatial relations.

be any number of points, lines or polygons⁸. A query condition can be represented as a predicate expression, which is a set of atomic predicates connected by logical operations (i.e., ‘and’ (\wedge), ‘or’ (\vee) and ‘not’). An atomic predicate is of the form “*op1 operation op2*”, where the operation can be either spatial or aspatial, *op1* is an attribute name, and *op2* is either an attribute name (i.e., a join operation) or a constant (i.e., a select operation). A spatial query is where the condition involves at least one spatial operation.

Therefore, a spatial query to find all chemical factories that are adjacent to a lake can be expressed for the purpose of query transformation as

$$factory.type = 'chemical' \wedge lake.boundary \text{ meet } factory.boundary$$

We use θ to denote an aspatial operation, and Θ for a spatial operation.

2.2. Generalized Filter

Spatial operations are intrinsically time-consuming, because spatial objects are large and spatial relationships are complex to evaluate. A common strategy in spatial operation processing is the filter-and-refine approach^{1,13}. This approach uses some simpler operations to act as the necessary condition (i.e., the filtering condition) for a spatial operation. For example, a spatial object can be approximated using some simpler geometric shapes, such as the minimum bounding rectangles (MBRs). An MBR can be represented by the coordinates of its lower-left and upper-right corners (i.e., (x_{low}, y_{low}) and (x_{high}, y_{high})). As two polygons overlap only if their MBRs intersect, the operation of MBR intersection is a filtering condition for polygon intersection. MBR intersection can be tested, no matter how complex these objects are, as

$$A.x_{low} < B.x_{high} \wedge A.y_{low} < B.y_{high} \wedge B.x_{low} < A.x_{high} \wedge B.y_{low} < A.y_{high}$$

Note that this can be treated as an aspatial operation. Its processing cost is typically lower than the polygon intersection operation. Only those objects which satisfy the filtering condition need to be further tested using the spatial operation on full object geometry. By using such a filter-and-refine processing strategy, the cost of spatial operation processing can be reduced as the CPU cost is reduced for using simpler operations and the I/O cost can be reduced for using object approximations which are typically much smaller than the objects they represent. This idea of filter-and-refinement for a single spatial join operation can be extended to a distributed environment ³.

In this paper we extend this idea to complex query processing. In spatial decision support systems and new applications such as spatial on-line analytical processing and data mining, a query can be very complex, involving several spatial operations including spatial joins as well as some aspatial predicates. Such a query can be processed by decomposing it into two sets of tasks: one set for aspatial conditions and another set for spatial conditions. For a query with both spatial and aspatial components, the spatial conditions are often processed before the aspatial part ¹¹. However, it is not always efficient to process all aspatial operations first, as spatial predicates might be more selective thus should be executed first to reduce the sizes of interim data sets.

Now we propose to use *generalized filter* (or *g-filter* in short) for processing complex spatial queries. Let Q be a query condition where Q is defined as

$$Q = \underbrace{(A_1 \Theta_1 A'_1) \wedge \cdots \wedge (A_n \Theta_n A'_n)}_{\text{spatial predicates}} \wedge \underbrace{(B_1 \theta_1 B'_1) \wedge \cdots \wedge (B_m \theta_m B'_m)}_{\text{aspatial predicates}}$$

where Θ_i and θ_j can be either selection or join operations (i.e., A'_i and B'_j can be attribute names or constants), $1 \leq i \leq n$, $1 \leq j \leq m$. We consider mainly the conjunction of predicates in this paper, as disjunction of predicates is typically processed by a database system as a union of subqueries. The generalized filter of Q , denoted as q , is defined as

$$q = \underbrace{(A_1 \Delta_1 A'_1) \wedge \cdots \wedge (A_n \Delta_n A'_n)}_{\text{filtering conditions}} \wedge \underbrace{(B_1 \theta_1 B'_1) \wedge \cdots \wedge (B_m \theta_m B'_m)}_{\text{aspatial predicates}}$$

where Δ_i is an aspatial filtering condition for Θ_i . Note that the operands for Δ_i can be approximations of A_i and A'_i . When the MBR is used as object approximation, q can be an aspatial query. Clearly, the result of q is a superset of the result of Q . The result of Q can be obtained by refining the result of q by applying the refinement condition q' , which consists of the spatial predicates of Q , on the data set obtained from executing a task using q as the condition.

$$q' = A_1 \Theta_1 A'_1 \wedge A_2 \Theta_2 A'_2 \cdots \wedge A_n \Theta_n A'_n$$

Using the concept of g-filter, a complex spatial query can be processed in the following steps:

1. (*Generating a g-filter*) A generalized filter query q is constructed from all aspatial conditions as well as the filter conditions for each spatial operation. A corresponding refinement query q' is also constructed in this step.
2. (*Executing the g-filter to produce a candidate list*) The g-filter obtained from the previous step is optimized (using techniques such as those to be discussed later in the paper) and then executed to produce a list of candidates. A candidate consists of object identifiers for all spatial attributes which are used by the refinement query q' or appear in the target list.
3. (*Refinement using the candidate list*) To test the refinement condition (i.e., q') for each candidate, by requesting spatial objects by their identifiers as in the candidate from different sites.
4. (*Post-query processing*) Additional operations, such as spatial aggregate functions, are processed here.

Many optimizations can be done for the refinement step, such as eliminating duplicates and ordering candidates in a way such that total communication or I/O cost are minimized^{12,1}. However, to compliment all these optimizations, it is important to minimize the amount of spatial data to be used in query processing (some of them may need to be transferred from remote database sites). Thus, it is highly desirable to make a g-filter as selective as possible to produce the smallest possible candidate list. The rest of this paper will focus on the simplification and enhancement of g-filter based on semantic relationships among spatial operations.

3. Semantic Relationships

In comparison to relational operations, spatial operations have richer semantic relationships among them. Such relationships can be explored for query optimization. First, a spatial operation is typically implemented using some computational geometry algorithm, which often has multiple processing steps. For example, to test if a pair of polygons A and B intersect or not, a typical algorithm consists of three steps¹³: 1) if a point in polygon A is inside polygon B or not; 2) if a point in B is inside A or not; and 3) if there is a line segment in A crossing or touching a line segment from B . Each of these steps itself is implemented using a complex algorithm. Therefore, one spatial operation can be implemented as a sequence of other operations. Second, a set of spatial operations can have certain relationships between each other, such as *generalization*, *specialization* and *mutual-exclusiveness*. The eight relations in Figure 1 are mutually exclusive. A user can define a new spatial relation **intersect** as that there is a point in one object which is also inside or on the boundary of another object (this is a common definition seen in several SDBMSs). Then **intersect** is a generalization of all spatial relations in Figure 1 except **disjoint**. Both cases above imply that an apparently atomic spatial operation may be decomposed into other sub-operations. They differ, however, in how

these sub-operations are related to the final results. When Θ is decomposed into $\Theta_1, \Theta_2 \dots$, we use the following two notations:

1. $\Theta \equiv \langle \Theta_1, \Theta_2, \dots, \Theta_n \rangle$ when Θ_{i+1} needs to be applied to the resultant data set produced by Θ_i . That is, Θ_i produces a *superset* of the final results, which need to be further examined by Θ_{i+1} . The final results are generated after Θ_n is applied. For example, $\text{contains} \equiv \langle \text{intersect}, \text{contains} \rangle$.
2. $\Theta \equiv \{\Theta_1, \Theta_2, \dots, \Theta_n\}$ where Θ_i is applied to original data sets to produce a *subset* of the final results. The final results are the union of results from applying each Θ_i . For example, $A \text{ intersect } B \equiv \{A[0] \text{ inside } B, B[0] \text{ inside } A, A \text{ line_intersect } B\}$ where $A[0]$ is the first point of A .

These two strategies can be used together. Before we apply the semantic relationships of spatial operations to query optimization, we discuss the underlying foundation of semantic relationships among spatial operations. In order to precisely describe the meaning of a spatial operation, Egenhofer proposes a formal description of binary topological relationships between spatial objects^{4,5,6}. For a spatial object A , one can define its boundary (A_b), exterior (A_e) and interior (A_i). The topological relationship between two spatial objects A and B can be characterized by the following 9-intersection matrix:

$$\begin{pmatrix} A_b \cap B_b & A_b \cap B_i & A_b \cap B_e \\ A_i \cap B_b & A_i \cap B_i & A_i \cap B_e \\ A_e \cap B_b & A_e \cap B_i & A_e \cap B_e \end{pmatrix}$$

A combination of whether each of these 9 intersections is empty or not ('0' for empty and '1' for nonempty) defines a unique spatial relation between two objects. The spatial relations in Figure 1 are defined as follows:

$$\begin{aligned} I_{disjoint} &= \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} & I_{contains} &= \begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \\ I_{inside} &= \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} & I_{equal} &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \\ I_{meet} &= \begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} & I_{covers} &= \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \\ I_{covered_by} &= \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} & I_{overlap} &= \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \end{aligned}$$

This matrix is a fundamental instrument for defining spatial relations precisely. They are used, for example, in the SQL/MM standard and Oracle 8. While a total of $2^9 = 512$ types of spatial relation can be defined using the 9-intersection matrix, many of them cannot be realized or the difference among them is not of concern

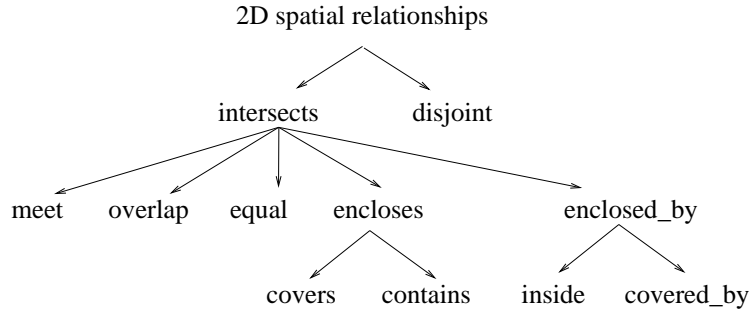


Fig. 2. A hierarchy of spatial operations.

to the user. Only these eight relations are useful for a continuous vector space, according to several researchers⁴.

The 9-intersection notation is useful for query optimization. The property of mutual-exclusiveness has been exploited in⁴. The basic idea is that, when the relationship between any two spatial objects is one and only one from a set of pre-defined relations, the algorithm to determine if a particular relationship holds or not can be simplified. For example, in order to see if two objects are equal or not, no more than 3 intersections need to be tested to exclude the other seven relations (for example, using the 3 intersections on the first row in I_{equal}).

Some applications may not require such strict mutual-exclusiveness among spatial operations. The 9-intersection notation can be extended to accommodate non-exclusive operations to some extent. For example, when there is no need to distinguish *contains* and *covers* (similarly, *inside* and *covered_by*), one can introduce *encloses* (and *enclosed_by*) for *covers* or *contains* (for *inside* or *covered_by* respectively). If we introduce “*” to represent *do-not-care*, then we have the following definitions where the condition of boundary intersection is ignored.

$$I_{encloses} = \begin{pmatrix} * & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \quad I_{enclosed_by} = \begin{pmatrix} * & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

The 9-intersection matrix is not able to define the arbitrary spatial operations an SDBMS (or a user if the system allows users to define new operations) wishes to define. For example, it cannot describe *intersect* as defined before which covers all non-disjoint relations. For the purpose of spatial query optimization, it is important to understand semantic relationships among spatial operations. Therefore, we introduce a concept called the *semantic tree* for spatial operations. The relationship among a set of spatial operations can be presented as a tree, where a child node is a specialized form of the parent node. Figure 2 is the semantic tree for the eight relations plus *intersect*, *encloses* and *enclosed_by* as defined previously. The semantic tree is a piece of easy-to-define information to capture semantic relationships among spatial operations. Such a tree can be derived from the 9-intersection matrix with *do-not-care* elements (i.e., an operation with a spec-

ified value is a specialized form of the one with corresponding ‘*’). A semantic tree can also be modified manually when an operation cannot be described using the 9-intersection matrix notation (e.g., `intersect`). Given a semantic tree of spatial operations, two predicates Θ_1 and Θ_2 are in one and only one of the following relationships:

1. $\Theta_1 \leftarrow \Theta_2$: Θ_1 is a *generalized* operation of Θ_2 if Θ_1 is an ancestor of Θ_2 in the semantic tree;
2. $\Theta_1 \rightarrow \Theta_2$: Θ_1 is a *specialized* form of Θ_2 when $\Theta_2 \leftarrow \Theta_1$ (that is, Θ_1 is in a sub-tree of Θ_2); or
3. $\Theta_1 \# \Theta_2$: Θ_1 and Θ_2 are *non-relevant* if there exists one node in the tree such that Θ_1 and Θ_2 belong to different sub-trees of that node.

4. Predicate Transformation

In this section we revisit those techniques used in relational query optimization for predicate transformation and examine new aspects for spatial queries based on semantic relationships of spatial operations. These techniques can be used for g-filter optimization.

4.1. Predicate Simplification

When a complex query condition is given to a query optimizer, the condition will be partially evaluated by the optimizer whenever possible. Any redundant or conflicting predicates will be identified in this step. For example, “`A inside $R_1 \wedge A$ inside R_2` ” can be simplified to “`A inside R_1` ” if R_1 is a region fully enclosed by R_2 . This type of simplification is data-dependant (i.e., the containment relationship between two objects, R_1 and R_2 need to be known by the query optimizer). With the information captured by a semantic tree, further simplification, which is data-independent, can be made with spatial expressions. Table 1 lists the rules for simplifying “ `$A\Theta_1B$ op $A\Theta_2B$` ”. “N/A” in the table means no simplification is made.

Table 1. Simplification of spatial operations.

Original Predicates	Simplified Predicates		Original Predicates	Simplified Predicates	
	$\Theta_1 \rightarrow \Theta_2$	$\Theta_1 \# \Theta_2$		$\Theta_1 \rightarrow \Theta_2$	$\Theta_1 \# \Theta_2$
$\Theta_1 \wedge \Theta_2$	Θ_1	False	$\Theta_1 \vee \Theta_2$	Θ_2	N/A
$\neg\Theta_1 \wedge \Theta_2$	N/A	Θ_2	$\neg\Theta_1 \vee \Theta_2$	True	$\neg\Theta_1$
$\Theta_1 \wedge \neg\Theta_2$	False	Θ_1	$\Theta_1 \vee \neg\Theta_2$	N/A	$\neg\Theta_2$
$\neg\Theta_1 \wedge \neg\Theta_2$	$\neg\Theta_2$	N/A	$\neg\Theta_1 \vee \neg\Theta_2$	$\neg\Theta_1$	True

Another type of simplification can be achieved based on the 9-intersection matrices. For two spatial relations Θ_1 and Θ_2 , whose 9-intersection matrices are $I_{\Theta_1} = \{a_{ij} : 1 \leq i, j \leq 3\}$ and $I_{\Theta_2} = \{b_{ij} : 1 \leq i, j \leq 3\}$, we can have a matrix for $\Theta_1 \wedge \Theta_2$, which is simply defined as $I_{\Theta_1 \wedge \Theta_2} = \{c_{ij} = a_{ij} \wedge b_{ij} : 1 \leq i, j \leq 3\}$.

Then we can simplify $a_{ij} \wedge b_{ij}$ using the following rules (a_{ij} and b_{ij} are symmetric):

$$c_{ij} = \begin{cases} a_{ij} & \text{if } b_{ij} = '*' \\ a_{ij} & \text{if } b_{ij} \neq '*' \wedge a_{ij} = b_{ij} \\ FALSE & \text{if } b_{ij} \neq '*' \wedge a_{ij} \neq '*' \wedge a_{ij} \neq b_{ij} \end{cases} \quad (4.1)$$

For $\Theta_1 \vee \Theta_2$, the simplification is more complex as we do not have a simple property such as $I_{\Theta_1 \vee \Theta_2} = \{a_{ij} \vee b_{ij} : 1 \leq i, j \leq 3\}$.

4.2. Predicate Decomposition

An apparently atomic spatial predicate can be decomposed into smaller granularity. For example, a spatial relation between two objects can be checked by finding whether each of the 9 intersections is empty or not (the query optimization work in ⁴ is based on this assumption). Further, because of the hierarchical semantic relationships among spatial operations, new operations can be *introduced*. When $\Theta_1 \rightarrow \Theta_2$, by definition we know that $A\Theta_1 B \subseteq A\Theta_2 B$ and $A\Theta_1 B \equiv \langle A\Theta_2 B, A\Theta_1 B \rangle$ (here we use a predicate to denote both the operation and the resultant data set). When an operation Θ is replaced by a sequence of predicates, we say that Θ is decomposed. Θ can be decomposed in two ways: 1) used in conjunction with one or several operations which are its generalized operations; and 2) replaced by a sequence of operations which implement Θ . The former can be determined using the semantic tree information, and the latter requires the knowledge of the spatial operation implementation.

One purpose of predicate decomposition is to introduce a filtering step. When Θ_1 is replaced by $\langle \Theta_2, \Theta_1 \rangle$, the filtering step is to apply Θ_2 on the input data sets before the second step (the refinement step) applies Θ_1 on the results of the filtering step. This transformation is worthwhile if Θ_2 is less resource-consuming than Θ_1 and produces interim data sets which are smaller than the original data sets such that the total cost of processing the two operations in that order is smaller than that of processing Θ_1 only. As mentioned before, for many spatial operations one particularly important generalized operation is `mbr_intersect`.

With a semantic tree, the filter-and-refine approach is not limited to using `mbr_intersect` as the filtering condition. As long as there is a generalized operation that is less expensive to evaluate, it can be used in the filter step. A generalized operation can be used as a filter to identify a superset of the final query results. At the same time, a specialized operation can also be used as filter to identify a subset of the final query results. A specialized operation can also reduce the total cost by reducing the data set to be 'refined' using more expensive operations. For example, by using certain types of conservative approximation such as an internal rectangle to approximate a spatial object, intersection of such internal rectangles means that the polygons must intersect. Thus, a pair of polygons whose internal rectangles intersect is recognized as overlapping without any further check. Both types of optimization techniques have been explored by several previous researches.

Another application of predicate decomposition can enhance a well-known op-

timization technique in relational query optimization for identifying *common sub-expressions* for spatial query processing. For a complex query, there may exist some common sub-expressions (i.e., applying the same operations on same data). For example, $(\sigma_{\theta_1 \wedge \theta_2} R) \bowtie (\sigma_{\theta_1 \wedge \theta_3} R)$ has a common sub-expression $\sigma_{\theta_1} R$ in both operands of the join operation. These common sub-expressions need to be evaluated only once, with the results stored temporarily for other references of the sub-expression. Identification of common sub-expressions can avoid redundant processing. It is important for a query optimizer to consider forming common sub-expressions using predicate transformation. While this rule is applicable in a straightforward way to spatial predicates, the semantic hierarchy of spatial predicates introduces a new dimension: a common spatial sub-expression can be derived from apparently irrelevant predicates by predicate decomposition.

Consider the following example to find the golf courses that have water features. Let G and W be two data sets for golf courses and water features respectively, the query condition can be expressed as

$$W \text{ inside } G \vee W \text{ meet } G \vee W \text{ overlap } G$$

One execution plan is to have three separate sub-queries followed by a union operation. A more efficient alternative is to decompose these predicates using the filter-and-refine strategy and execute only once $W \text{ mbr_intersect } G$, which is a common sub-expression for the three operations. That is,

$$\langle W \text{ mbr_intersect } G, \{W \text{ inside } G, W \text{ meet } G, W \text{ overlap } G\} \rangle$$

Further, the 9-intersection matrices for these three operations have a common component. This common component consists of a number of intersection tests, which can be treated as a common sub-expression. This leads to the following execution plan:

$$\langle W \text{ mbr_intersect } G, I_{com}, \{I'_{inside}, I'_{meet}, I'_{overlap}\} \rangle$$

where I_{com} contains only the bottom three elements in the 9-intersection matrix (which are the common elements of the matrices of the three spatial operations), and I'_{inside} is I_{inside} less than the three elements in I_{com} , and so on.

4.3. Predicate Composition

Let Θ_1 and Θ_2 be two spatial predicates, then a condition $A\Theta_1B \wedge B\Theta_2C$ may imply a new predicate Θ_3 such that $A\Theta_1B \wedge B\Theta_2C$ implies $A\Theta_3C$. Obviously, this new relationship $A\Theta_3C$ is redundant. There is no need to check $A\Theta_3C$ if the original condition is true. However, it can be used to form a more selective filter. For example, “ A encloses $B \wedge B$ intersects C ” implies “ A intersects C ”. Therefore, the filter condition for “ A encloses $B \wedge B$ intersects C ” becomes “ $A \text{ mbr_intersect } B \wedge B \text{ mbr_intersect } C \wedge A \text{ mbr_intersect } C$ ”. In this case, polygons A , B , and C in Figure 3 are not to be processed by the refinement task

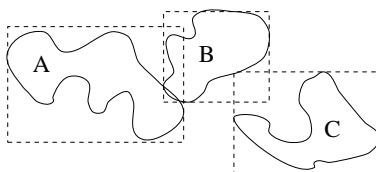


Fig. 3. An example of useful predicate composition.

because of a new filtering condition derived from “*A intersects C*”. In this example, spatial predicates are used to derive an aspatial predicate (i.e., *mbr_intersect*) which can be used to make a filter more selective. Obviously, this derived condition will be ignored by the refinement step.

The problem of spatial predicate composition has been investigated in ⁷. For the set of spatial operations considered in this paper, the rule for identifying useful predicate composition, in terms of producing a derived predicate to enhance filter, is that at least one of the predicates is *encloses* or *enclosed_by*. In other words, Θ_3 can be derived as below (note that Θ_1 and Θ_2 are symmetric):

$$\Theta_3 = \begin{cases} \text{encloses} & \text{if } \Theta_1 = \text{encloses} \wedge \Theta_2 = \text{encloses} \\ \text{intersects} & \text{if } \Theta_1 = \text{encloses} \wedge \Theta_2 \neq \text{encloses} \end{cases} \quad (4.2)$$

For the purpose of filter enhancement, these two rules are not different as they both imply *A mbr_intersect C*. However, the difference can be useful if the costs to evaluate different spatial operations are taken in to account ⁴. This technique is more useful when a query optimizer has statistical information about the database. For example, if the maximum size of spatial objects in a table is known, then “*A intersect B*” and “*B intersect C*” will lead to a condition about the distance between *A* and *C*. This in turn leads to a filtering condition that the MBR of one object, after being enlarged properly by considering the maximum object size, must intersect with the MBR of another object.

5. Conclusions

Spatial query optimization is a complex issue which requires much more research before an SDBMS can handle complex queries with similar performance to that which a relational DBMS can achieve. Previous research on this issue largely focused on processing single spatial operations using spatial indices. We approached this problem from a new angle. In this paper, we have proposed the generalized filter as a framework for processing complex spatial queries. This framework works for a single SDBMS to process a complex spatial query. More importantly, it can also be used for processing spatial queries involving multiple sites. We have studied relationships among spatial operations, and introduced a new mechanism, the semantic tree, to describe such relationships. A semantic tree can capture information that is useful for spatial query optimization but cannot be conveyed using traditional methods. We have discussed applications of the semantic tree and the

9-intersectin matrix in spatial query transformation. Such transformation can be used to simplify spatial queries, generate more selective filters, and generate more useful candidate execution plans. This paper has considered a wide range of optimization techniques for complex spatial queries. It is an important contribution towards integrated query processing for spatial and aspatial predicates.

Acknowledgements

The first author conducted part of this research while in the CSIRO. His work is also supported by a UQ NSRSF grant. The second and third authors' research are partially supported by a small ARC grant from UNSW and the Faculty Research Initiative Fund from UTS respectively.

References

1. D. J. Abel, V. Gaede, R. A. Power, and X. Zhou. Caching strategies for spatial joins. *GeoInformatica*, 3(1):33–59, 1999.
2. D. J. Abel, V. Gaede, K. Taylor, and X. Zhou. SMART: Towards a Spatial Internet Marketplace. *GeoInformatica*, 3(2):141–164, 1999.
3. D. J. Abel, B. C. Ooi, K.-L. Tan, R. Power, and J. X. Yu. Spatial join strategies in distributed spatial dbms. In *LNCS 951: Proceedings of 4th Int. Symp. on Spatial Databases (SSD'95)*, pages 346 – 367. Springer-Verlag, 1995.
4. E. Clementini, J. Sharma, and M. J. Egenhofer. Modeling topological spatial relations: strategies for query processing. *Cpmp. and Graphics*, 18(6):815–822, 1994.
5. M. J. Egenhofer. Reasoning about binary topological relationships. In *LNCS 552: Proceedings of 2nd Int. Symp. on Spatial Databases (SSD'91)*, pages 143–160. Springer-Verlag, 1991.
6. M. J. Egenhofer. Topological relations between regions in R^2 and Z^2 . In *LNCS 692: Proceedings of 3rd Int. Symp. on Spatial Databases (SSD'93)*, pages 316–331. Springer-Verlag, 1993.
7. M. J. Egenhofer. Spatial SQL: a query and presentation language. *IEEE transactions on knowledge and data engineering*, 6(1):86–95, 1994.
8. V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.
9. A. Gupta. *Integration of Information Systems: Bridging Heterogenous Databases*. IEEE Computer Press, 1989.
10. R. H. Güting. An introduction to spatial database systems. *VLDB Journal*, 3(4):357–399, 1994.
11. B. C. Ooi. *LNCS 471: Efficient query processing in GIS*. Springer-Verlag, 1990.
12. Y. Zhang, J. Xiao, and X. Zhou. A declustering algorithm for minimizing spatial join cost. In *LNCS 1276: Proc. of 3rd International Conference on Computing and Combinatorics (COCOON'97)*, pages 363–372. Springer-Verlag, 1997.
13. X. Zhou, D. J. Abel, and D. Truffet. Data partitioning for parallel spatial join processing. In *LNCS 1262: Proc. 5th Int. Symp. on Spatial Databases (SSD'97)*, pages 178–196. Springer-Verlag, 1997.