Indexing Vague Regions in Spatial Data Warehouses

Thiago Luís Lopes Siqueira[1,2], João Celso Santos de Oliveira[1], Valéria Cesário Times[3] Cristina Dutra de Aguiar Ciferri[4], Ricardo Rodrigues Ciferri[1]

[1] Department of Computer Science, Federal University of São Carlos, UFSCar, 13.565-905, São Carlos, SP, Brazil

[2] São Paulo Federal Institute of Education, Science and Technology, IFSP, 13.565-905, São Carlos, SP, Brazil.

[3] Informatics Center, Federal University of Pernambuco, UFPE, 50.670-901, Recife, PE, Brazil

[4]Department of Computer Science, University of São Paulo, USP, 13.560-970, São Carlos, SP, Brazil

prof.thiago@ifsp.edu.br, joaocelso@comp.ufscar.br, vct@cin.ufpe.br cdac@icmc.usp.br, ricardo@dc.ufscar.br

Abstract. *A vague spatial data warehouse allows multidimensional queries with spatial predicates to support the analysis of business scores related to vague spatial data, addressing real world phenomena characterized by inexact locations or indeterminate boundaries. However, vague spatial data are usually represented and stored as multiple geometries and impair the query processing performance. In this paper, we introduce an index called VSB-index to improve the query processing performance in vague spatial data warehouses, focusing on range queries and vague regions. We also conduct an experimental evaluation demonstrating that our VSB-index provided remarkable performance gains up to 97% over existing solutions.*

1. Introduction

Decision-making support has gained the attention of researchers of Geographic Information System (GIS), Data Warehouse (DW) and Online Analytical Processing (OLAP). Fast, flexible, and multidimensional ways for spatial data analysis are provided by Spatial OLAP tools that query a Spatial Data Warehouse (SDW), which is a subject-oriented, integrated, time-variant, voluminous, non-volatile and multidimensional database that mainly stores crisp spatial data as vector (e.g. political boundaries) and their descriptive attributes (conventional data) [Bimonte et al. 2010]. A fact denotes the scores of business activities through numeric measures or spatial measures, while dimensions hold conventional attributes and spatial attributes that contextualize values of measures. Usually, spatial range queries concerning ad hoc spatial query windows select specific spatial objects stored in the SDW, e.g. intersection range query (IRQ) [Siqueira et al. 2012a]. The performance of query processing is a critical issue in SDW and motivates the design of indices to reduce the elapsed time to join huge tables, process spatial predicates and aggregate voluminous data [Papadias et al. 2001; Siqueira et al. 2012a].

Mainly, SDWs store crisp spatial data. On the other hand, several real world phenomena are affected by spatial vagueness, which is one kind of spatial data imperfection concerning the difficulty of distinguishing an object shape from its neighborhood. As a result, it is not possible to be sure if the parts of a spatial object belong completely or partially to it or not. Exact models based on objects represent spatial vagueness reusing well-known crisp spatial data models and extending the theory of spatial data types and spatial predicates [Pauly and Schneider 2010].

A vague region *r* has a known extent and a broad boundary comprising a two-dimensional zone surrounding the known extent, instead of a one-dimensional line with minimal thickness. According to Pauly and Schneider (2010), *r* is a pair of crisp regions: the *kernel* and the *conjecture*. The *kernel* represents the known extent and is the determinate part of *r*, while the *conjecture* represents the broad boundary and is the vague part of *r*. The interior of the kernel and the interior of the conjecture are disjoint. If *p* is a point and belongs to the kernel, then it *certainly* belongs to *r*. However, if *p* belongs to the conjecture, then *p possibly* belongs to *r*. If *p* does not belong to the kernel and neither to the conjecture, then p does not belong to *r*.

Although vague SDWs store both vague spatial data and crisp spatial data, and allow multidimensional and spatial analysis of business scores regarding spatial vagueness, the design and use of SDWs supporting vague spatial data, i.e. vague SDWs, are still in infancy [Siqueira et al. 2012b; Edoh-Alove et al. 2013]. Furthermore, little attention has been devoted to the experimental evaluation of query processing performance in vague SDWs and to the investigation of the cost to process spatial predicates against vague spatial data represented as multiple geometries. Such investigation could aid designers to improve the performance of their systems.

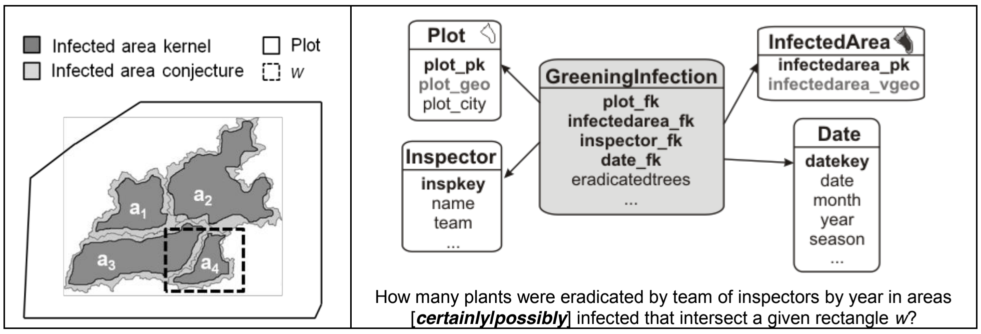
Motivated by the challenge of improving the query processing performance in vague SDWs, in this paper, we provide the following contributions: (i) an experimental evaluation of indices implemented by DBMSs and developed for SDWs to verify if they offer a reasonable query processing performance in vague SDW; (ii) a progressive approximation called MIP, which drastically reduces the cost of the spatial predicate resolution; and (iii) the proposal of an index called Vague Spatial Bitmap Index (VSB-index) to efficiently process multidimensional queries extended with spatial predicates concerning vague regions in SDWs.

This paper is organized as follows. Section 2 summarizes a case study, Section 3 surveys related work, Section 4 reports a performance evaluation of DBMS and Indices for SDW, Section 5 describes the VSB-index, Section 6 reports the experimental evaluation of the VSB-index and Section 7 concludes the paper.

2. Case Study: Greening

The following case study of a real problem in agriculture increases the motivation of this work. Greening is a serious disease that infects citrus and impairs the industry. It is caused by a bacterium transmitted by an insect. As there is not a cure so far, its control is done by visual inspection and immediate eradication of the infected plant by the roots. Temporal and spatial patterns of distribution of greening in the field at different scales, as plots and cities, are crucial to reduce the rate of failures in visual inspections [Silva et al. 2011]. Every area infected by greening is a vague region with broad boundaries as shown in Figure 1a. The kernel is the extent where plants were infected, while the conjecture is the broad boundary where the insect possibly transmitted the bacterium to plants.

Therefore, the vague SDW depicted in Figure 1b was built according to Siqueira et al. (2012b). *GreeningInfection* is a fact table referencing dimension tables and holding the numeric measure of eradicated plants. *Inspector* and *Date* are conventional dimension tables. *Plot* is a crisp spatial dimension table with the crisp spatial attribute *plot_geo* of type polygon. *InfectedArea* is a vague spatial measure pushed in a vague spatial dimension table with the vague spatial attribute *infectedarea_vgeo* of type multipolygon. Areas *certainly* infected are fetched as their kernels intersect the spatial query window *w*, e.g. a_3 and a_4 in Figure 1a. More coarsely, areas *possibly* infected are fetched as their conjectures intersect *w*, e.g. a_1, a_2, a_3 and a_4 in Figure 1a. These queries are more relevant to aid reducing failures in visual inspections than queries specified by exact models, e.g. Pauly and Schneider (2010), which compare a vague region to other vague regions of the dataset.



(a) Vague SDW schema **(b) Vague spatial data distribution**

Figure 1. A vague SDW on greening infection

3. Related Work

Currently, spatial predicates are solved in vague SDWs using indices of DBMSs, which are suitable for crisp spatial data and use the MBR as conservative approximation, e.g. R-tree [Guttman 1984] and GiST [Aoki 1997]. The filter step of the spatial predicate resolution uses strictly one conservative approximation: the MBR. Differently from our VSB-index, they do not perform a multistep spatial predicate resolution [Brinkhoff et al. 1993], i.e. they do not use a progressive approximation in addition to the MBR in order to identify answers already in the filter step to reduce the cost of the refinement step.

A bitmap join index on the attribute *C* of a dimension table indicates the set of rows in the fact table to be joined with a certain value of *C* [O'Neil and Graefe 1995]. Although the bitmap join index avoids joining huge tables in DWs, it cannot solve spatial predicates. In SDWs, the aR-tree [Papadias et al. 2001], the SB-index and the HSB-index [Siqueira 2012b] are capable to process spatial predicates, conventional predicates, aggregation and sorting. They use a single conservative approximation on crisp spatial data: the MBR. As a result, they produce identical sets of candidates to be processed in the refinement step. The aR-tree and the HSB-index have hierarchical data structures and a tree-based search in the filter step (similar to the R-tree's), while the SB-index has a sequential data structure and a sequential search in the filter step (similar to our VSB-index'). Conventional predicates, aggregation and sorting are processed by the aR-tree manipulating multidimensional arrays, while the SB-index, the HSB-index and our VSB-index reuse bitmap join indices to process them.

Some indices for vague spatial data focus on probability density functions and field data [Tao et al. 2005; Zinn et al. 2007]. On the other hand, the VSB-index addresses vector data. The vague R-tree is an index for vague regions based on the R-tree [Petry et al. 2007]. Its intermediate nodes maintain a pair of entries per cluster of vague regions. Entry *O* holds a MBR circumscribing the MBRs of the clustered vague regions, while entry *I* holds a MBR circumscribing the MBRs of kernels of the clustered vague regions. Progressive approximations were not addressed and only algorithms for point queries were designed, differently from our VSB-index that uses progressive approximations and tackles range queries. Besides, the vague R-tree was not assessed through an experimental evaluation, differently from the VSB-index.

4. Performance Evaluation of DBMS and Indices for SDW

In this section we conduct an experimental evaluation to demonstrate that indices implemented in DBMSs and indices for SDW are not suitable to manipulate vague regions. Section 4.1 addresses the experimental setup and Section 4.2 describes results.

4.1 Experimental Setup

Regarding the dataset, we processed real polygons of the rural census of the Brazilian Institute of Geography and Statistics to create the vague SDW shown in Figure 1b with 302,357 multipolygons in the attribute *infectedarea_vgeo*. The kernel was a negative buffer on the real polygon, while the conjecture was the convex hull of the real polygon minus the real polygon. The data generator of the Star Schema Benchmark produced conventional data for the other tables with scale factor 10 (60 million facts).

The workload was based on the query shown in Figure 2 (adapted from the Star Schema Benchmark) that assesses IRQ as spatial predicate, testing the rectangular ad hoc spatial query window *w* against a high cardinality attribute (e.g. infectedarea_vgeo). We performed 10 consecutive queries using disjoint spatial query windows, flushing the system cache between them, and gathered the average elapsed time. Given a spatial attribute of cardinality *c*, and a spatial query window *w* that retrieves *n* objects, the selectivity was $n \div c$. We used the following selectivity values for the spatial predicate: 0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007, 0.008, 0.009, 0.01, 0.02, 0.03 and 0.04. We assume that a user of SOLAP tool would hardly select more than 12,000 vague regions to be retrieved and displayed. Time reduction measured how much a configuration was more efficient than another. Configurations are described as follows.

Complying with the logical design methods of Siqueira et al. (2012b), configuration *DBMS1* assessed the DBMS processing the query transcribed in Figure 2 with *TABLE=InfectedArea* and *ATTRIBUTE=infectedarea_vgeo* over the schema shown in Figure 1b. A GiST index was built on infectedarea_vgeo to aid the processing of the spatial predicate on multiple geometries (multipolygons).

The SB-index and the aR-tree were evaluated due to their efficiency in SDWs. They were implemented in C/C++ and the disk page size was set to 8 KB. The filter step comprised index scan, while the refinement step accessed multipolygons of *infectedarea_vgeo*. The SB-index built bitmap join indices on attributes *team*, *year*, *infectedarea_pk* and *eradicatedtrees* using FastBit (https://sdm.lbl.gov/fastbit/), while the aR-tree built a 1,000×7×302,357 3-dimensional array based on the cardinalities of *team*, *year*, *infectedarea_pk*, respectively, to store values of the measure *eradicatedtrees*.

In contrast to the logical design methods of Siqueira et al. (2012b), we created configuration *DBMS2* to assess the DBMS using a schema similar to that shown in Figure 1b, but replacing the table InfectedArea by table *InfectedArea2*, whose attributes were *infectedarea_pk*, *infectedarea_ker_geo* of type polygon storing the kernel of the vague region and *infectedarea_outb_geo* of type polygon storing the outer boundary of the conjecture of the vague region. Note that the outer boundary of the conjecture encompasses the vague region. The query transcribed in Figure 2 had the parameters set to *TABLE=InfectedArea2* and *ATTRIBUTE=infectedarea_outb_geo*. GiST indices were built on *infectedarea_ker_geo* and *infectedarea_outb_geo* to aid processing the spatial predicate on simple polygons instead of multiple polygons (DBMS1).

The platform was a computer with a 3.2 GHz Pentium D processor, 8 GB of main memory, a 7200 RPM SATA 320 GB hard disk with 8 MB of cache, Linux CentOS 6, PostgreSQL 9.2 and PostGIS 2.0.1.

```
SELECT team, year, SUM(eradicatedtrees) FROM Inspector, Date, GreeningInfection, TABLE
WHERE inspkey=inspector_fk AND datekey=date_fk AND infectedarea_fk=infectedarea_pk
AND team='XY' AND INTERSECTS(ATTRIBUTE,w)
GROUP BY team, year ORDER BY team, year
```

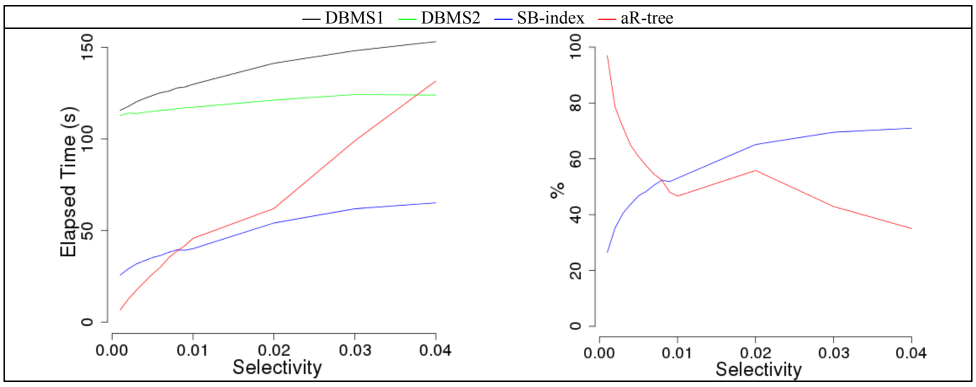
Figure 2. Querying the vague SDW shown in Figure 1b

4.2 Results

Figure 3a reports the elapsed time to process queries using the configurations DBMS1, DBMS2 and SB-index previously described. Clearly, as higher the selectivity value was, greater was the time spent to process queries in all configurations. Furthermore, the separation of the vague spatial attribute of type multipolygon in two attributes of type polygon benefited the query processing performance, since DBMS2 spent less time than DBMS1 to process queries. Such performance finding opposes the logical design methods stated by Siqueira et al. (2012b) and indicates that designers should separate vague regions in a vague SDW stored in the DBMS as we have done in Section 4.1.

However, both DBMS1 and DBMS2 had prohibitive query response times and indicated the necessity of using indices to provide a better performance. The aR-tree greatly overcame the other configurations for selectivity values less than 0.01, while the SB-index overcame the other configurations for higher selectivity values. Yet, both indices spent prohibitive times to process queries with increasing values of selectivity.

To identify the bottleneck in the query processing performance of the indices for SDW, we measured the time spent on each phase of its query processing algorithm. The fraction to resolve the spatial predicate is reported in Figure 3b. We concluded that the resolution of spatial predicates against vague regions was a costly step to process queries in vague SDWs. For the SB-index, such cost augmented as the selectivity of the spatial predicate increased. For instance, it was less than 30% for the selectivity 0.001 and greater than 70% for the selectivity 0.04. Clearly, the SB-index does not offer mechanisms to reduce the cost of the spatial predicate resolution against vague regions. Conversely, such cost decreased in the aR-tree for increasing values of selectivity since the manipulation of the multidimensional array imposed a larger overhead.



(a) Total elapsed time **(b) Fraction of the spatial predicate**

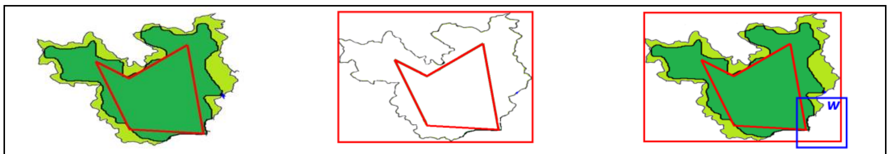
Figure 3. Results of DBMS and Indices for SDW

5. The Vague Spatial Bitmap Index

To propose the VSB-index, some design choices were made as follows, based on the previous discussions. We prioritized a multistep resolution of the spatial predicate and then created a specific progressive approximation to be used in the filter step and reduce the cost of the refinement step. We have chosen range queries as the spatial predicates to be supported by the VSB-index, initially, to satisfy the case study requirements. Since queries issued on a vague SDW process not only spatial predicates, but also conventional predicates, aggregation and sorting, the latter three are processed by bitmap join indices that are commonly used in DWs. This section details the proposal of the VSB-index and is organized as follows. Section 4.1 introduces the progressive approximation MIP. Section 4.2 defines the data structure of the VSB-index. Section 4.3 addresses the building operation of the VSB-index. Section 4.4 focuses on the VSB-index query processing.

5.1. Maximum Area Inscribed Polygon

The Maximum Area Inscribed Polygon (MIP) is a progressive approximation consisting of a polygon with *x* vertices. The number of vertices is the suffix, e.g. MIP5 for *x*=5. We define MIP to be applied specially on vague regions, to improve the resolution of spatial predicates in query processing. Figure 4a shows a vague region, its conjecture (light green), its kernel (dark green) and a MIP5 on its kernel (red contour). The outer boundary of the conjecture encompasses the vague region and therefore a MIP5 on the kernel is also a subset of the outer boundary of the conjecture, as shown in Figure 4b.



(a) MIP5 on the kernel **(b) MBR on outer boundary and MIP5 on kernel** **(c) IRQ on $O_{\supseteq}K_{\subseteq}$**

Figure 4. Vague region, approximations and query

5.2. Data Structure

The VSB-index for vague regions is an array whose entries have the type *vrbitvector* (vague region bit-vector), comprising: (i) one key value *pk*; (ii) one mandatory conservative approximation $O_{\supseteq}$ on the outer boundary of the conjecture; (iii) one optional progressive approximation $O_{\subseteq}$ on the outer boundary of the conjecture; (iv) one optional conservative approximation $K_{\supseteq}$ on the kernel; (v) one optional progressive approximation $K_{\subseteq}$ on the kernel; and (vi) a pointer *ptr* to the bitvector of the key value in a bitmap join index. The nomenclature considers the conservative approximation $_{\supseteq}$ as a superset of the vague region, and the progressive approximation $_{\subseteq}$ as a subset of the vague region. All feasible configurations for the VSB-index are listed in Table 1. The conservative approximation $O_{\supseteq}$ is mandatory to enable the query processing since the outer boundary of the conjecture encompasses the vague region. Except $O_{\supseteq}$, the other approximations are optional and allow flexible data structures and query processing algorithms (Section 5.4). We encourage using MIP as progressive approximations $O_{\subseteq}$ and $K_{\subseteq}$.

5.3. Building Operation

The building operation of the VSB-index issues a SQL query selecting the primary key and the spatial attribute from the spatial dimension table and sorting the results in ascending order based on the primary key. For each row retrieved, the approximations of the vague region are calculated and copied together with the key value into one entry of an array in the main memory. When the array becomes full, it is written to a disk page of the index file. After processing all rows and writing all disk pages, a bitmap join index is built on primary key values. Since the entries of the VSB-index are sorted by the primary key values, VSB-index[i] refers to the bitvector B[i] of the bitmap join index.

The size of a VSB-index entry, in bytes, is $s = sizeof(int) + sizeof(O_{\supseteq}) + sizeof(O_{\subseteq}) + sizeof(K_{\supseteq}) + sizeof(K_{\subseteq})$. Each disk page with l bytes maintains $L = l\ DIV\ s$ index entries. Some unused bytes $U = c\ MOD\ L$, where c is the cardinality of the indexed vague spatial attribute, are left between different disk pages to avoid fragmented entries and prevent two disk accesses to obtain a single entry. There is also a header disk page to store metadata. Then, $A = 1 + c\ DIV\ L + y$ disk pages are required to store the VSB-index, where y=0, if $c\ MOD\ L = 0$; and y=1, otherwise. Besides, A disk accesses are required to build the index file. Table 1 exemplifies values of s, L and A for MIP5, l=8192 bytes and c=302,357.

Table 1. Index entry sizes in bytes (*s*), number of entries per disk page (*L*) and number of disk pages used to store the index file (*A*).

	$O_{\supseteq}O_{\subseteq}K_{\supseteq}K_{\subseteq}$	$O_{\supseteq}O_{\subseteq}K_{\supseteq}$	$O_{\supseteq}O_{\subseteq}K_{\subseteq}$	$O_{\supseteq}O_{\subseteq}$	$O_{\supseteq}K_{\supseteq}K_{\subseteq}$	$O_{\supseteq}K_{\supseteq}$	$O_{\supseteq}K_{\subseteq}$	$O_{\supseteq}$
s	228	148	196	116	148	68	116	36
L	35	55	41	70	55	120	70	227
A	8640	5499	7376	4321	5499	2521	4321	1333

5.4. Query Processing

The range queries supported by the VSB-index are the following. Let w be an iso-oriented rectangle called ad hoc spatial query window and S be a set of vague regions. An $IRQ_{poss}(w,S)$ concerns an intersection that is *possibly* true and retrieves vague regions in S whose outer boundary of the conjecture intersects w. Conversely, an $IRQ_{cert}(w,S)$ concerns an intersection that is *certainly* true and retrieves vague regions in *S* whose kernel intersects *w*. CRQ_{poss} and CRQ_{cert} are defined analogously for the relationship of containment. ERQ_{poss} and ERQ_{cert} are defined analogously for enclosure ("inside of").

The VSB-index query processing firstly performs a filter step as a sequential scan on the index file which requires *A* disk accesses (Section 5.3). The functions that execute such filter step are function *f1* detailed in Algorithm 1 and function *f2* detailed in Algorithm 2. They produce candidates and answers of the spatial predicate and store them in their proper sets in the main memory, i.e. *setCandidates* and *setAnswers*, respectively.

Function *f1* performs a sequential scan over the index file (lines 2-7), which retrieves each disk page (line 3) and temporarily stores it in the main memory (line 4). Function *get* obtains the conservative approximation of every entry transferred to main memory (line 6). Such conservative approximation is $O_{\supseteq}$ or $K_{\supseteq}$, depending on the parameter passed, and is tested against the ad hoc spatial query window (line 6). If the spatial relationship is satisfied, the entry's primary key value is appended to a set (line 7). Finally, the index file is closed (line 8). The aforementioned set might be the set of candidates or the set of answers, depending on the parameter passed.

To identify answers already in the filter step, function *f2* performs a sequential scan that firstly tests the conservative approximation and secondly tests the progressive approximation. For each entry (lines 5-10), if the spatial relationship is satisfied for both the conservative and progressive approximations, the entry is considered an answer and its primary key value is stored in the set of answers (lines 6-8). However, if only the conservative approximation satisfies the spatial relationship, the entry is considered a candidate and its primary key value is stored in the set of candidates (line 10). According to the calls, *f2* is particularly useful for IRQ_{poss} when $O_{\supseteq}$ and $O_{\subseteq}$ are available, and for IRQ_{cert} when $K_{\supseteq}$ and $K_{\subseteq}$ are available. Also, $K_{\subseteq}$ can be used to fetch results when querying IRQ_{poss}, as well as $O_{\supseteq}$ can be used to indicate candidates when querying IRQ_{cert}.

```
Algorithm 1: f1(R, w, conservative, set, idx, L)

Input: parameters described in Table 2
Declarations: page, array
Output: a set of candidates or a set of answers of the spatial predicate

1  open (idx)
2  while not (eof(idx)) do
3      read (idx, page)
4      copy (page, array)
5      for i ← 0 to L do
6          if R(w, get(array[i],conservative))
7              append(set, array[i].pk)
8  close(idx)
```

```
Algorithm 2: f2(R, w, conservative, progressive, setCandidates, setAnswers, idx, L)

Input: parameters described in Table 2
Declarations: page, array
Output: the set of candidates and the set of answers of the spatial predicate

1   open (idx)
2     while not (eof(idx)) do
3       read (idx, page)
4       copy (page, array)
5       for i ← 0 to L do
6         if R(w, get(array[i],conservative))
7           if R(w, get(array[i],progressive))
8             append(setAnswers, array[i].pk)
9           else
10            append(setCandidates, array[i].pk)
11  close(idx)
```

The filter step is a call to an adequate function based on a decision regarding the spatial predicate to evaluate and which approximations are available among $O_{\supseteq}$, $O_{\subseteq}$, $K_{\supseteq}$

and $K_{\subseteq}$. There are a total of 48 situations, but in this paper we focus on IRQ$_{\text{cert}}$ and IRQ$_{\text{poss}}$ which are solved by *f1* or *f2*. The 16 calls to process IRQ$_{\text{cert}}$ and IRQ$_{\text{poss}}$ are listed in Table 3. For instance, *f2* is called to process both IRQ$_{\text{cert}}$ and IRQ$_{\text{poss}}$ for $O_{\supseteq}K_{\subseteq}$ and adds the vague region shown in Figure 4c to the set of answers already in the filter step.

After the filter step, the refinement step is performed using the DBMS and its results are recorded in *setAnswers*. Further, a key-matching produces a string with a conventional predicate based on primary key values of those vague regions that satisfy the spatial predicate. Such string replaces the spatial predicate of the query submitted to the vague SDW. For instance, "INTERSECTS…" in Figure 2 is replaced by "(infectedarea_ pk=10 OR infectedarea_pk=15)", where 10 and 15 are key values of vague regions that satisfy the spatial predicate. Finally, the rewritten query is solved by efficient bitmap join indices that avoid joining huge SDW tables and provide the query answer.

Table 2. Parameters of Algorithms 1 and 2.

Parameter	Description
conservative	Indicator for $O_{\supseteq}$ or $K_{\supseteq}$
conservativeK	Indicator for $K_{\supseteq}$
conservativeO	Indicator for $O_{\subseteq}$
idx	The VSB-index file
L	The maximum number of index entries that a disk page can hold
progressive	Indicator for $O_{\subseteq}$ or $K_{\subseteq}$
progressiveK	Indicator for $K_{\subseteq}$
progressiveO	Indicator for $O_{\subseteq}$
pk	The primary key attribute of *table*
R	The spatial relationship (intersection, containment or enclosure)
setAnswers	The set of answers of the spatial predicate
setCandidates	The set of candidates (possible answers) of the spatial predicate
table	The vague spatial dimension table queried
vsa	The vague spatial attribute of *table*
w	The ad hoc spatial query window

Table 3. Calls made to functions *f1* and *f2* by configuration of the VSB-index.

Configuration	Function calls of IRQ$_{\text{cert}}$ and IRQ$_{\text{poss}}$ (R=intersection)

$O_{\supseteq}O_{\subseteq}K_{\supseteq}K_{\subseteq}$	`IRQcert: f2 (R, w, conservativeK, progressiveK, setCandidates, setAnswers, idx, L)`
	`IRQposs: f2 (R, w, conservativeO, progressiveO, setCandidates, setAnswers, idx, L)`

$O_{\supseteq}O_{\subseteq}K_{\supseteq}$	`IRQcert: f1(R, w, conservativeK, setCandidates, idx, L)`
	`IRQposs: f2 (R, w, conservativeO, progressiveO, setCandidates, setAnswers, idx, L)`

$O_{\supseteq}O_{\subseteq}K_{\subseteq}$	`IRQcert: f2 (R, w, conservativeO, progressiveK, setCandidates, setAnswers, idx, L)`
	`IRQposs: f2 (R, w, conservativeO, progressiveO, setCandidates, setAnswers, idx, L)`

$O_{\supseteq}O_{\subseteq}$	`IRQcert: f1(R, w, conservativeO, setCandidates, idx, L)`
	`IRQposs: f2 (R, w, conservativeO, progressiveO, setCandidates, setAnswers, idx, L)`

$O_{\supseteq}K_{\supseteq}K_{\subseteq}$	`IRQcert: f2 (R, w, conservativeK, progressiveK, setCandidates, setAnswers, idx, L)`
	`IRQposs: f2 (R, w, conservativeO, progressiveK, setCandidates, setAnswers, idx, L)`

$O_{\supseteq}K_{\supseteq}$	`IRQcert: f1(R, w, conservativeK, setCandidates, idx, L)`
	`IRQposs: f1(R, w, conservativeO, setCandidates, idx, L)`

$O_{\supseteq}K_{\subseteq}$	`IRQcert: f2 (R, w, conservativeO, progressiveK, setCandidates, setAnswers, idx, L)`
	`IRQposs: f2 (R, w, conservativeO, progressiveK, setCandidates, setAnswers, idx, L)`

$O_{\supseteq}$	`IRQcert: f1(R, w, conservativeO, setCandidates, idx, L)`
	`IRQposs: f1(R, w, conservativeO, setCandidates, idx, L)`

6. Experimental Evaluation of the VSB-index

This section reports the remarkable performance of the VSB-index. The experimental setup is described in Section 5.1, the building operation and the storage requirements are discussed in Section 5.2, and IRQ_{poss} and IRQ_{cert} are tackled in Section 5.3.

6.1 Experimental Setup

We extended the experimental setup described in Section 4.1 as follows. Five configurations of the VSB-index were reported because their results were more notable: $O_{\supseteq}$, $O_{\supseteq}K_{\supseteq}$, $O_{\supseteq}K_{\subseteq}$, $O_{\supseteq}O_{\subseteq}$ and $O_{\supseteq}O_{\subseteq}K_{\supseteq}K_{\subseteq}$. We used MBR as conservative approximation and MIP5 as progressive approximation – 5 vertices by analogy with 5C from Brinkhoff et al. (1993). The VSB-index was implemented in C/C++ and the disk page size was set to 8 KB. MIP5 was built using the CGAL, Computational Geometry Algorithms Library (http://www.cgal.org) version 4.0.2 and the method CGAL::maximum_area_inscribed _k_gon_2. The method uses monotone matrix search [Aggarwal et al. 1987] and has a worst case running time of $O(x \times n + n \times \log n)$, where n is the number of vertices provided as input to build the MIP and x is the number of vertices of the output.

6.2 Time spent and storage requirements to build

Figure 5a reports the elapsed time to build the sequential file of the VSB-index for each configuration and separates: (i) the time spent to extract the boundary of the kernel or the outer boundary of the conjecture; and (ii) the time spent to build the approximations and store them in disk. The configurations that hold only MBRs were built in shorter time, i.e. $O_{\supseteq}$ and $O_{\supseteq}K_{\supseteq}$. Also, the overhead to build the MIP5 on the outer boundary of the conjecture was significantly lower than the overhead to build the MIP5 on the kernel. Then, the time spent to build the configuration $O_{\supseteq}O_{\subseteq}$ was shorter than to build the configurations $O_{\supseteq}K_{\subseteq}$ and $O_{\supseteq}O_{\subseteq}K_{\supseteq}K_{\subseteq}$. In fact, the boundary of the kernel is a negative buffer and has more vertices than the outer boundary of the conjecture that is a convex hull (Section 4.1). Therefore, the high number of vertices of the kernel impaired the calculation of the MIP5 for configurations $O_{\supseteq}K_{\subseteq}$ and $O_{\supseteq}O_{\subseteq}K_{\supseteq}K_{\subseteq}$. The storage requirements for the VSB-index are detailed in Figure 5b. As expected, configurations that hold more approximations also require more storage space (Table 1). Considering that bitmap join indices occupied 4GB (Section 4.1), then the VSB-index' sequential file added at least 0.25% ($O_{\supseteq}$) up to at most 1.7% ($O_{\supseteq}O_{\subseteq}K_{\supseteq}K_{\subseteq}$) to storage requirements.

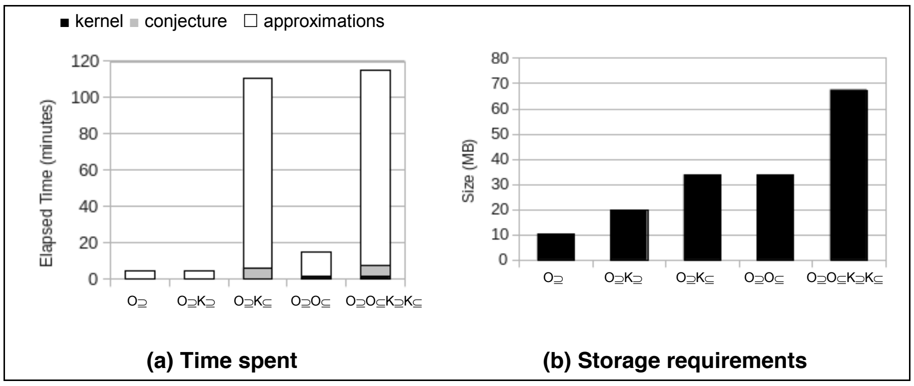


Figure 5. Results to build the VSB-index.

6.3 IRQ_{poss} and IRQ_{cert}

Instead of examining the time to process a complete query in the vague SDW (Section 3), this section focuses the resolution of the spatial predicate, motivated by the results discussed in Section 4.2. As SB-index' and aR-tree's performances were similar to the results achieved by the configuration $O_{\supseteq}$, we reported them together. A very low selectivity value of 0.0001 was included. Figure 6a shows the results of IRQ_{poss}. The configuration $O_{\supseteq}O_{\subseteq}$ outperformed the other configurations because since MIP5 allows identifying answers of the spatial predicate in the filter step. Although the configuration $O_{\supseteq}O_{\subseteq}K_{\supseteq}K_{\subseteq}$ produces the same set of candidates, given by the same call to function *f2*, more disk accesses are performed in the filter step due to a larger index entry size in bytes required to store four approximations (Table 1). Even though the query assessed the outer boundary of the conjecture, the progressive approximation MIP5 on the kernel of $O_{\supseteq}K_{\subseteq}$ provided a shorter query response time than configuration $O_{\supseteq}O_{\subseteq}K_{\supseteq}K_{\subseteq}$ that has the MIP5 on the outer boundary of the conjecture. Both configurations $O_{\supseteq}$ and $O_{\supseteq}K_{\supseteq}$ do not maintain a progressive approximation and therefore were severely impaired because they did not identify answers in the filter step and had a costly refinement step. Although the IRQ was issued over the outer boundary of the conjecture, a progressive approximation on the kernel improved the query processing performance, as the configuration $O_{\supseteq}K_{\subseteq}$ provided a time reduction of at least 23% and at most 97% over SB-index and aR-tree ($O_{\supseteq}$), for selectivity values 0.0001 and 0.04, respectively.

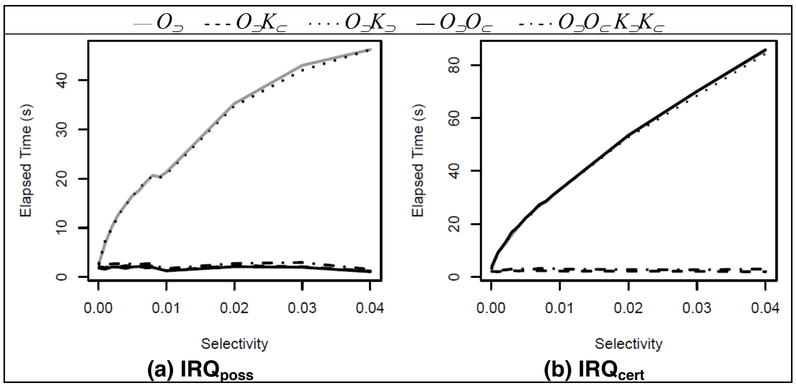


Figure 6. Results for IRQ_{poss} and IRQ_{cert}.

As for the VSB-index, Figure 6b shows the results of IRQ_{cert}. Curves for configurations $O_{\supseteq}$, $O_{\supseteq}K_{\supseteq}$ and $O_{\supseteq}O_{\subseteq}$ overlap each other. These configurations do not hold a MIP5 on the kernel and therefore had worst performances. As for the configuration $O_{\supseteq}O_{\subseteq}$, the progressive approximation the outer boundary of the conjecture could not improve the query processing performance of IRQ_{cert}, since it is not possible to obtain answers by calling the function *f1*. On the other hand, configurations $O_{\supseteq}O_{\subseteq}K_{\supseteq}K_{\subseteq}$ and $O_{\supseteq}K_{\subseteq}$ provided shorter query response times because they hold a MIP5 on the kernel and therefore can identify answers in the filter step. Again, the configuration $O_{\supseteq}O_{\subseteq}K_{\supseteq}K_{\subseteq}$ was impaired by its larger entry size that lead to more disk accesses to perform the filter step than the configuration $O_{\supseteq}K_{\subseteq}$. The configuration $O_{\supseteq}K_{\subseteq}$ provided a time reduction of at least 36% and at most 97% over SB-index and aR-tree ($O_{\supseteq}$), for selectivity values 0.0001 and 0.04, respectively. Notably, configuration $O_{\supseteq}K_{\subseteq}$ efficiently processed IRQ_{cert} and IRQ_{poss}.

7. Conclusions and Future Work

In this paper, we identified the lack of an index for vague SDW and the bottleneck to process vague spatial data using indices designed for crisp SDW. We also have gone one step forward and introduced the VSB-index to provide efficient processing of multidimensional queries extended with range queries against vague regions in vague SDWs. The VSB-index has a flexible data structure that enables the use of multiple approximations such that its query processing algorithm fits according to them. We also presented the progressive approximation MIP used by the VSB-index to reduce the cost of the refinement step in the spatial predicate resolution. An experimental evaluation corroborated the efficiency of the VSB-index that had remarkable performance gains up to 97% over existing solutions. Also, a study case was described to reinforce the feasibility of using our VSB-index in real applications. We are currently evaluating containment range queries and enclosure range queries. As future work, we intend to extend the VSB-index to support nearest neighbor queries and spatial joins, to index other data types as vague points and vague lines, and to enable SOLAP operations as roll-up and drill-down.

Acknowledgements. This work has been supported by FAPESP, CNPq, CAPES, INEP and FINEP. The 2nd author thanks PIBIC/CNPq/UFSCar for the undergraduate scholarship.

References

Aggarwal, A., Klawe, M. M., Moran, S., Shor, P. W., Wilber, R. 1987. Geometric applications of a matrix-searching algorithm. Algorithmica, 2, 195-208

Aoki, P.M. 1997. Generalizing "Search" in Generalized Search Trees. In ICDE, 380-389

Bimonte, S., Tchounikine, A., Miquel, M., Pinet, F. 2010. When Spatial Analysis Meets OLAP: Multidimensional Model and Operators. IJDWM, 6, 4, 33-60

Brinkhoff, T., Kriegel, H. P., Schneider, R. 1993. Comparison of Approximations of Complex Objects Used for Approximation-based Query Processing in Spatial Database Systems. In ICDE, 40-49

Edoh-Alove, E., Bimonte, S. Pinet, F., Bédard, Y. 2013. Exploiting Spatial Vagueness in Spatial OLAP: Towards a New Hybrid Risk-Aware Design Approach. In AGILE, 4p.

Guttman, A. 1984. R-Trees: A Dynamic Index Structure for Spatial Searching. ACM SIGMOD Record, 14, 2, 47-57

O'Neil, P., Graefe, G. 1995. Multi-Table Joins Through Bitmapped Join Indices. ACM SIGMOD Record, 24, 3, 8-11

Papadias, D., Kalnis, P., Zhang, J., Tao, Y. 2001. Efficient OLAP Operations in Spatial Data Warehouses. In SSTD, 443-459

Pauly, A., Schneider, M. 2010. VASA: An algebra for vague spatial data in databases. Inf. Syst., 35, 1, 111-138

Petry, F., Ladner, R., Somodevilla, M. 2007. Indexing Implementation for Vague Spatial Regions with R-trees and Grid Files. In: A. Morris, S. Kokhan, Geographic Uncertainty in Environmental Security, 187-199

Siqueira, T. L. L., Ciferri, C. D. A., Times, V. C., Ciferri, R. R. 2012a. The SB-index and the HSB-Index: efficient indices for spatial data warehouses. Geoinformatica, 16, 1, 165-205

Siqueira, T.L.L., Ciferri, C.D.A., Times, V.C., Ciferri, R. 2012b. Towards Vague Geographic Data Warehouses. In GIScience, 173-186

Silva, D.C.P., Posadas A., Jorge, L.A.C., Inamasu, R.Y. Paiva, M.S.V. 2011. Geração de mapas de incidência de greening utilizando técnicas Wavelets-Multifractais. In: R.Y. Inamasu et al. (Eds), Agricultura de Precisão: um novo olhar, EMBRAPA Instrumentação, 82-86 (Portuguese).

Tao, Y., Cheng, R., Xiao, X., Ngai, W., Kao, B., Prabhakar, S. 2005. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In VLDB, 922-933

Zinn, D., Bosch, J., Gertz, M. 2007. Modeling and Querying Vague Spatial Objects Using Shapelets. In VLDB, 567-578