

UNIVERSIDADE FEDERAL DE SÃO CARLOS
UNIVERSITÉ LIBRE DE BRUXELLES

JOINT SUPERVISION OF DOCTORAL THESIS

**THE DESIGN OF VAGUE SPATIAL DATA
WAREHOUSES**

THIAGO LUÍS LOPES SIQUEIRA

SUPERVISORS:
RICARDO RODRIGUES CIFERRI
ESTEBAN ZIMÁNYI

São Carlos – SP

December/2015

UNIVERSIDADE FEDERAL DE SÃO CARLOS
UNIVERSITÉ LIBRE DE BRUXELLES

JOINT SUPERVISION OF DOCTORAL THESIS

**THE DESIGN OF VAGUE SPATIAL DATA
WAREHOUSES**

THIAGO LUÍS LOPES SIQUEIRA

A thesis presented in partial fulfilment of the requirements for the degree of *Doutor em Ciência da Computação* from *Universidade Federal de São Carlos* and the degree of *Docteur en Sciences de l'ingénieur et technologie* from *Université libre de Bruxelles*

Supervisors:

Ricardo Rodrigues Ciferri

Esteban Zimányi

São Carlos – SP

December/2015

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária UFSCar
Processamento Técnico
com os dados fornecidos pelo(a) autor(a)

S618dv Siqueira, Thiago Luís Lopes
The design of vague spatial data warehouses /
Thiago Luís Lopes Siqueira. -- São Carlos : UFSCar,
2015.
325 p.

Tese (Doutorado) -- Universidade Federal de São
Carlos, 2015.

1. Spatial data warehouses. 2. Spatial vagueness.
3. Conceptual modeling. 4. Logical design. 5.
Indexing. I. Título.



UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a Defesa de Tese de Doutorado do candidato Thiago Luis Lopes Siqueira, realizada em 07/12/2015:

Prof. Dr. Ricardo Rodrigues Ciferri
UFSCar

Profa. Dra. Marcela Xavier Ribeiro
UFSCar

Profa. Dra. Marilde Terezinha Prado Santos
UFSCar

Prof. Dr. Esteban Zimanyi
ULB

Prof. Dr. Stijn Vansummeren
ULB

Prof. Dr. Alejandro Vaisman
ITBA

To Luísa and Cris.

ACKNOWLEDGEMENTS

I am grateful to the following institutions, whose administrative, financial, and technical supports were essential for the development of this thesis. Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (IFSP) provided me a leave of absence to attend a doctoral program (project #23315500316/2013-81). Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) funded a scholarship aimed at doctoral split fellowship program (grant #229675/2013-1). Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) and Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) funded a scholarship aimed at doctoral program (grant #2014/14103-9, São Paulo Research Foundation). Empresa Brasileira de Pesquisa Agropecuária (Embrapa) kindly provided a real dataset and allowed its use. Universidade Federal de São Carlos (UFSCar) and Université libre de Bruxelles (ULB) agreed on a joint supervision of doctoral thesis.

I thank Ricardo R. Ciferri and Esteban Zimányi for dedicating themselves to supervise me as a doctoral candidate and to contribute with the development of my professional and academic skills. I also thank Cristina D. A. Ciferri, João Celso S. Oliveira, Rodrigo C. Mateus, and Valéria C. Times, for the interesting discussions and fruitful collaborations we had.

Last but not least, I thank my daughter Luísa and my wife Cris for their love; my parents Silvia and Geraldo and my sister Thayse for their unconditional support and encouraging words; and Jesus for being lamp to my feet and for offering a source of living water and inspiration.

*Lead me on, lead me on
To a place where the river runs into your keeping
Oh, lead me on, lead me on
The awaited deliverance comforts the seeking, lead on.*

A. Grant, W. Kirkpatrick, and M. W. Smith

RESUMO

O *data warehouse* espacial (DWE) é um banco de dados multidimensional integrado e volumoso que armazena dados espaciais e dados convencionais. Já o processamento analítico-espacial *online* (SOLAP) permite consultar o DWE, tanto pela seleção de dados espaciais que satisfazem um relacionamento topológico, quanto pela agregação dos dados espaciais. Deste modo, DWE e SOLAP beneficiam o suporte a tomada de decisão. As aplicações de DWE e SOLAP abordam majoritariamente fenômenos representados por dados espaciais exatos, ou seja, que assumem localizações e fronteiras bem definidas. Contudo, tais aplicações negligenciam dados espaciais afetados por imperfeições, tais como a vagueza espacial, a qual interfere na identificação precisa de um objeto e de seus vizinhos. Um objeto espacial vago não tem sua fronteira ou seu interior precisamente definidos. Além disso, é composto por partes que certamente pertencem a ele e partes que possivelmente pertencem a ele. Apesar de inúmeros fenômenos do mundo real serem caracterizados pela vagueza espacial, na literatura consultada não se identificaram trabalhos que considerassem a vagueza espacial no projeto de DWE e nem para consultar o DWE. Tal limitação motivou a elaboração desta tese de doutorado, a qual introduz os conceitos de DWE vago e de SOLAP vago. Um DWE vago é um DWE que armazena dados espaciais vagos, enquanto que SOLAP vago provê os meios para consultar o DWE vago. Nesta tese, o projeto de DWE vago é abordado e as principais contribuições providas são: (i) o modelo conceitual VSCube que viabiliza a criação de um cubos de dados multidimensional para representar o esquema conceitual de um DWE vago; (ii) o modelo conceitual VSMultiDim que permite criar um diagrama para representar o esquema conceitual de um DWE vago; (iii) diretrizes para o projeto lógico do DWE vago e de suas restrições de integridade, e para estender a linguagem SQL visando processar as consultas de SOLAP vago no DWE vago; e (iv) o índice VSB-index que aprimora o desempenho do processamento de consultas no DWE vago. A aplicabilidade dessas contribuições é demonstrada em dois estudos de caso no domínio da agricultura, por meio da criação de esquemas conceituais de DWE vago, da transformação dos esquemas conceituais em esquemas lógicos de DWE vago, e do processamento de consultas envolvendo as regiões vagas do DWE vago.

Palavras-chave: data warehouse espacial, vagueza espacial, modelo conceitual, projeto lógico, indexação

ABSTRACT

Spatial data warehouses (SDW) and spatial online analytical processing (SOLAP) enhance decision making by enabling spatial analysis combined with multidimensional analytical queries. A SDW is an integrated and voluminous multidimensional database containing both conventional and spatial data. SOLAP allows querying SDWs with multidimensional queries that select spatial data that satisfy a given topological relationship and that aggregate spatial data. Existing SDW and SOLAP applications mostly consider phenomena represented by spatial data having exact locations and sharp boundaries. They neglect the fact that spatial data may be affected by imperfections, such as spatial vagueness, which prevents distinguishing an object from its neighborhood. A vague spatial object does not have a precisely defined boundary and/or interior. Thus, it may have a broad boundary and a blurred interior, and is composed of parts that certainly belong to it and parts that possibly belong to it. Although several real-world phenomena are characterized by spatial vagueness, no approach in the literature addresses both spatial vagueness and the design of SDWs nor provides multidimensional analysis over vague spatial data. These shortcomings motivated the elaboration of this doctoral thesis, which addresses both vague spatial data warehouses (vague SDWs) and vague spatial online analytical processing (vague SOLAP). A vague SDW is a SDW that comprises vague spatial data, while vague SOLAP allows querying vague SDWs. The major contributions of this doctoral thesis are: (i) the Vague Spatial Cube (VSCube) conceptual model, which enables the creation of conceptual schemata for vague SDWs using data cubes; (ii) the Vague Spatial MultiDim (VSMultiDim) conceptual model, which enables the creation of conceptual schemata for vague SDWs using diagrams; (iii) guidelines for designing relational schemata and integrity constraints for vague SDWs, and for extending the SQL language to enable vague SOLAP; (iv) the Vague Spatial Bitmap Index (VSB-index), which improves the performance to process queries against vague SDWs. The applicability of these contributions is demonstrated in two applications of the agricultural domain, by creating conceptual schemata for vague SDWs, transforming these conceptual schemata into logical schemata for vague SDWs, and efficiently processing queries over vague SDWs.

Keywords: spatial data warehouses, spatial vagueness, conceptual modeling, logical design, indexing

RÉSUMÉ

Les entrepôts de données spatiales (EDS) et l'analyse en ligne spatiale (ALS) améliorent la prise de décision en permettant l'analyse spatiale combinée avec des requêtes analytiques multidimensionnelles. Un EDS est une base de données multidimensionnelle intégrée et volumineuse qui contient des données classiques et des données spatiales. L'ALS permet l'interrogation des EDS avec des requêtes multidimensionnelles qui sélectionnent des données spatiales qui satisfont une relation topologique donnée et qui agrègent les données spatiales. Les EDS et l'ALS considèrent essentiellement des phénomènes représentés par des données spatiales ayant une localisation exacte et des frontières précises. Ils négligent que les données spatiales peuvent être affectées par des imperfections, comme l'imprécision spatiale, ce qui empêche de distinguer précisément un objet de son entourage. Un objet spatial vague n'a pas de frontière et/ou un intérieur précisément définis. Ainsi, il peut avoir une frontière large et un intérieur flou, et est composé de parties qui lui appartiennent certainement et des parties qui lui appartiennent éventuellement. Bien que plusieurs phénomènes du monde réel sont caractérisés par l'imprécision spatiale, il n'y a pas dans la littérature des approches qui adressent en même temps l'imprécision spatiale et la conception d'EDS ni qui fournissent une analyse multidimensionnelle des données spatiales vagues. Ces lacunes ont motivé l'élaboration de cette thèse de doctorat, qui adresse à la fois les entrepôts de données spatiales vagues (EDS vagues) et l'analyse en ligne spatiale vague (ALS vague). Un EDS vague est un EDS qui comprend des données spatiales vagues, tandis que l'ALS vague permet d'interroger des EDS vagues. Les contributions majeures de cette thèse de doctorat sont: (i) le modèle conceptuel Vague Spatial Cube (VSCube), qui permet la création de schémas conceptuels pour des EDS vagues à l'aide de cubes de données; (ii) le modèle conceptuel Vague Spatial MultiDim (VSMultiDim), qui permet la création de schémas conceptuels pour des EDS vagues à l'aide de diagrammes; (iii) des directives pour la conception de schémas relationnels et des contraintes d'intégrité pour des EDS vagues, et pour l'extension du langage SQL pour permettre l'ALS vague; (iv) l'indice Vague Spatial Bitmap (VSB-index) qui améliore la performance pour traiter les requêtes adressées à des EDS vagues. L'applicabilité de ces contributions est démontrée dans deux applications dans le domaine agricole, en créant des schémas conceptuels des EDS vagues, la transformation de ces schémas conceptuels en schémas logiques pour des EDS vagues, et le traitement efficace des requêtes sur des EDS vagues.

Mots-clés: entrepôts de données spatiales, données spatiales vagues, modèle conceptuel, modèle logique, index

LIST OF FIGURES

1.1	Crops in remote sensed image and after identification. (a) A remote sensed image (adapted from Crop Circles in Kansas, by NASA Image of the Day Gallery). (b) Identified crops.	33
1.2	An applied area of pesticide A_1 over the crop C_1 . (a) A_1 over C_1 . (b) A_1 over C_{11} . (c) A_1 over C_{13} . (d) Subsets of A_1	33
1.3	HLB infection. (a) Plots. (b) Two infected trees. (c) Two infected groups. (d) Two groups and a continuous representation for an infected region. (e) Two groups and a discrete representation for an infected region.	36
2.1	The relational schema of a SDW regarding a retail application (adapted from (SIQUEIRA et al., 2010)).	51
2.2	A table and its columns indexed by bitmap join indices.	54
2.3	A region, its approximations and spatial range queries. (a) A region and its MBR. (b) A region and its MER. (c) A region and its MBR and MER. (d) Spatial query windows w and v	56
2.4	The multi-step resolution of spatial predicates (adapted from Brinkhoff, Kriegel & Schneider (1993)).	56
2.5	An R-tree built using cities of customers from the SDW described in Figure 2.1. (a) Cities, their MBRs and the R-tree's space partitioning method. (b) The R-tree data structure.	58
2.6	Probabilistic modeling of uncertain objects: the location of a vehicle (extracted from Tao, Xiao & Cheng (2007)).	62
2.7	A conceptual model of uncertainty in spatial data (adapted from Fisher (1999) and Jadidi et al. (2014)).	65

2.8	Examples of vague spatial objects designed according to existing exact models for spatial vagueness.	69
2.9	Vague regions A and B and the topological relationships between them according to existing exact models.	73
2.10	Example of vague spatial objects designed according to existing fuzzy models for spatial vagueness.	74
2.11	A simple fuzzy region and its membership function (VERSTRAETE; HALLEZ; TRÉ, 2006).	76
2.12	Evaluating the fuzzy topological relationship between two fuzzy regions. (a) Pairs of reference α -cuts in black, and $\tilde{A}_{\alpha_0}, \tilde{B}_{\alpha_0}$ in gray. (b) Membership functions and the linguistic labels they assume.	78
2.13	Examples of implemented fuzzy point, fuzzy lines and fuzzy regions.	79
3.1	The graphical notation of the MultiDim conceptual model: (a) Level. (b) Hierarchy. (c) Cardinalities. (d) Fact with measures and related levels. (e) Hierarchy name. (f) Spatial data types. (g) Topological relationships. Adapted from Vaisman & Zimányi (2014a).	86
3.2	A SDW regarding the maintenance of highways modeled according to the MultiDim model (extracted from Vaisman & Zimányi (2014a)).	88
3.3	A SDW regarding the maintenance of highways modeled according to the UML profile proposed by Boulil, Bimonte & Pinet (2015).	90
3.4	The logical design according to the Fuzzy SDW (adapted from Somodevilla & Petry (2003), Perez, Somodevilla & Pineda (2007)). (a) Sample points and a studied area. (b) Original schema. (c) The fuzzy MBR. (d) Modified schema.	93
3.5	A conceptual schema of SDW for CERA (adapted from Jadidi et al. (2013)).	94
3.6	Creating fuzzy regions for an arbitrary indicator (adapted from Jadidi et al. (2014)). (a) A magnified cell and its center. (b) The Gaussian function. (c) A grid and its vague regions defined over multiple cells.	95
3.7	Flow of fertilizer by material by month by spreading region. (a) A spreading region. (b) The SDW's fact table.	97
3.8	An example of aR-tree. (a) Cities, MBRs and a spatial query window. (b) SDW data. (c) Structure of the bi-dimensional array. (d) Data structure of the aR-tree.	100

3.9	An example of SB-index. (a) Cities, their MBRs and a spatial query window. (b) SDW data. (c) Bitmap join indices. (d) Data structure of the SB-index.	102
3.10	An example of vague R-tree. (a) Vague regions and their pair of MBRs. (b) Vague R-tree and three point queries. (c) Vague R-tree's data structure.	105
3.11	An example of FMBR R-tree (adapted from Petry, Ladner & Somodevilla (2007)). (a) A fuzzy region and points within it. (b) Data structure.	106
4.1	The types of attributes defined in the VSCube Conceptual Model.	116
4.2	Instances of crisp spatial attributes: (a) An address. (b) A watershed.	117
4.3	The vague spatial attribute v is a composite attribute with a multivalued certitude denoted by geometries and a multivalued dubiety denoted by geometries and their membership values.	118
4.4	Vague spatial attributes and instances for the pest control case study. (a) An applied area. (b) A crop.	119
4.5	The topological relationships among agricultural lands and crops.	121
4.6	The topological relationships among watersheds and agricultural lands.	123
4.7	A watershed, an agricultural land and crops.	125
4.8	The multidimensional and geographic views provided by the cuboid c_0 , and the fact f	130
4.9	The matrix of a vague spatial fact.	132
4.10	Lattice of cuboids $L_{PesticideApplication}$, represented according to the bottom-up notation of Ciferri et al. (2013).	134
4.11	A subset of the lattice $L_{PesticideApplication}$. (a) Cuboids. (b) The multidimensional view. (c) The geographic view. (d) Data aggregation.	135
4.12	The vague spatial object z and the spatial query window q	136
4.13	The results for $IRQ_{certitude_elements}$, $IRQ_{dubiety_elements}$ and $IRQ_{dubiety_elements-mval}$. (a) $IRQ_{certitude_elements}$. (b) $IRQ_{dubiety_elements}$. (c) $IRQ_{dubiety_elements-mval}$	138
4.14	A $VSRQ_{object}$ against crisp watersheds. (a) Watersheds and spatial query windows. (b) Result set.	139

4.15	The vague spatial union applied to the vague spatial objects x and y . (a) x and y . (b) $z = VSUnion(x,y)$	141
4.16	The vague spatial intersection applied to the vague spatial objects x and y . (a) x and y . (b) $z = VSIntersection(x,y)$	142
4.17	The vague spatial difference applied to the vague spatial objects x and y . (a) x and y . (b) $z = VSDifference(x,y)$	143
4.18	A pivoted cuboid.	145
4.19	A spatial range query against a TIN. (a) A TIN and a spatial query window w . (b) Triangles t_{black} and t_{gray} of the TIN.	149
4.20	Pictograms denoting data types supported by the VSMultiDim conceptual model.	152
4.21	Pictograms denoting topological relationships of the VSMultiDim conceptual model.	155
4.22	Vague spatial levels: (a) Obtained from a dimension of the pest control case study. (b) Modeled to comply with the HLB case study.	156
4.23	A hierarchy of the pest control case study represented according to both the VSCube model and the VSMultiDim model.	158
4.24	A hierarchy of the HLB case study represented according to the VSMultiDim model.	159
4.25	Hierarchies of the HLB case study represented according to the VSMultiDim model.	160
4.26	A conceptual schema of vague SDW for the pest control case study.	161
4.27	A conceptual schema of vague SDW for the HLB case study.	162
5.1	Separate Tables for Certitude and Dubiety.	172
5.2	A vague region and its storage in separate Tables for Certitude and Dubiety. . .	173
5.3	A vague region and its storage in separate Tables for Certitude and Dubiety considering a vague spatial attribute whose both the certitude and the dubiety are monovalued.	174
5.4	The vague spatial attribute represented by a single table.	175

5.5	An example of vague region according to the vague spatial attribute represented by a single table.	175
5.6	A vague region, its MBR and the MBRs of its elements.	176
5.7	A single table with a composite primary key.	177
5.8	The classes VSElementType and VSAttributeType and the table T.	178
5.9	A vague region and its representation as an array.	178
5.10	The vague spatial attribute represented by a pair of arrays.	180
5.11	A vague region and its representation as a pair of arrays.	180
5.12	The vague spatial attribute designed as a column for multiple geometries and a column for the array of membership values.	182
5.13	A vague region stored as a multiple geometry and an array of membership values.	183
5.14	The logical design for the vague spatial attribute with monovalued certitude and monovalued dubiety.	185
5.15	The storage of a vague region whose both certitude and dubiety are monovalued.	185
5.16	Shortcomings of both the 2D geometry type with measure and the 3D geometry type: (a) A vague point set. (b) A vague line. (c) Two elements of a vague region.	186
5.17	Examples of the application of Rule 2VS to obtain the vague spatial level tables Crop and Infection.	192
5.18	Mapping rules applied to a pair of related vague spatial levels.	195
5.19	The logical schema of the vague SDW regarding the HLB disease.	197
5.20	A vague spatial fact table. (a) The schema. (b) A vague spatial fact set.	201
5.21	A vague spatial fact table with a vague spatial measure for areas where pesticides were applied to.	203
5.22	Intersections among elements of A_1 and C_1 . (a) A_1 and C_1 . (b) Intersection among elements of A_1 and C_{13} . (c) Merging the geometries from (b). (d) Intersection among elements of A_1 and C_{11} . (e) Merging the geometries from (d).	204
5.23	A vague spatial fact set involving one numeric measure, one vague spatial measure and one vague spatial level table.	205

6.1	The vague SDW storing infected groups as vague point sets.	238
6.2	Creating vague point sets: (a) Processing the real polygon. (b) The vague point set.	239
6.3	The spatial query window w containing two vague point sets.	240
6.4	Performance results of the DBMS for the vague SDW containing vague point sets: (a) Storage requirements. (b) Query processing.	241
6.5	The schema of the vague SDW storing vague regions: (a) Incomplete schema. (b) Tables to complete the schema.	242
6.6	From real polygons to vague regions: (a) Processing the real polygon. (b) The obtained vague region.	243
6.7	The spatial query window w intersecting several vague regions of the vague SDW.	245
6.8	Results for the DBMS and indices for SDWs: (a) Average elapsed time. (b) Fraction of the vague spatial predicate.	246
6.9	The vague SDW storing vague regions in a separate table.	248
6.10	Elapsed time to process VSRQobject over the vague SDW containing vague regions.	250
6.11	A vague region and its approximations: (a) The MIP5 on the certitude. (b) The MBR on the vague region and the MIP5 on the certitude.	252
6.12	A vague region in tones of gray, its approximations with black continuous contour and spatial range queries with dashed rectangles: (a) A CRQ_{object} against O_{MBR} . (b) A $CRQ_{certitude}$ against $O_{MBR}C_{MBR}$. (c) An IRQ_{object} against $O_{MBR}C_{MIP5}$. (d) Another IRQ_{object} against $O_{MBR}C_{MIP5}$. (e) An $IRQ_{certitude}$ against $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$	259
6.13	A vague region, its elements and vague spatial predicates regarding the dubiety: (a) A $CRQ_{dubiety}$. (b) An $IRQ_{dubiety}$. (c) A query window into the holes.	262
6.14	Vague spatial range queries. (a) Against MBRs. (b) Against MBRs and MIP5s.	268
6.15	The schema of both the synthetic vague SDW and the real vague SDW.	273
6.16	Two vague regions and one spatial query window w	273
6.17	Results for IRQ_{object} and $IRQ_{dubiety}$ issued over the real vague SDW: (a) Average elapsed time. (b) Average number of candidates.	277

6.18	Results for IRQcertitude issued over the real vague SDW: (a) Average elapsed time. (b) Average number of candidates.	279
6.19	The performance to process IRQobject and IRQdubiety over the synthetic vague SDW.	281
6.20	The performance to process the IRQcertitude over the synthetic vague SDW. . .	283
6.21	Average elapsed time to process CRQobject and CRQdubity over the synthetic vague SDW.	284
6.22	Average elapsed time to process CRQcertitude over the synthetic vague SDW. .	286
6.23	Average elapsed time to process VSRQobject over the synthetic vague SDW. .	288
6.24	Time spent to build different configurations of the VSB-index.	290
6.25	Storage requirements for different configurations of the VSB-index.	290

LIST OF TABLES

1.1	Examples of queries for the analysis of the pest control activities.	34
1.2	Examples of queries for the analysis of of HLB control.	37
2.1	Multidimensional queries extended with spatial predicates.	47
2.2	Nomenclatures adopted by exact models for components of a vague spatial object.	68
2.3	Comparing exact models, fuzzy models and implementations for fuzzy models.	83
3.1	Comparing existing work on SDW design to conceptual modeling and logical design of vague SDWs introduced in this thesis.	108
3.2	Comparing existing work on design of SDWs characterized by spatial vagueness to the design of vague SDWs described in this thesis.	109
3.3	Comparing indices for SDW and indices for vague regions to the VSB-index. .	110
4.1	Types that the attributes x_i and x_{i+1} can assume.	122
4.2	Categories of hierarchies of the VSCube conceptual model.	126
5.1	The approaches described in Sections 5.2.1 to 5.2.6 and the goals they achieve.	187
5.2	Operations on vague spatial attributes.	219
5.3	Accessors: methods' signatures and descriptions.	220
5.4	Vague spatial predicates: methods' signatures and descriptions.	223
5.5	Vague spatial aggregation functions: methods' signatures and descriptions. . . .	228
5.6	Classes of slice and dice operations.	230
5.7	Classes of roll-up and drill-down operations.	230
5.8	The logical design of vague SDWs and the corresponding guidelines or implementations.	232

- 6.1 Entry size in bytes (s), number of entries per disk page (L) and number of disk pages to store the index file (A) considering different cardinalities (c). 253
- 6.2 Spatial range queries of the workload. 275
- 6.3 Summary of the results obtained with the experimental evaluation of the VSB-index. 292

TABLE OF CONTENTS

CHAPTER 1 – INTRODUCTION	29
1.1 The Pest Control Case Study	32
1.2 The HLB Case Study	34
1.3 Thesis Organization	37
CHAPTER 2 – THEORETICAL FOUNDATIONS	41
2.1 Spatial Data Warehouse Design	41
2.1.1 Conceptual Modeling	41
2.1.1.1 Entity-Relationship Model and Unified Modeling Language	42
2.1.1.2 Multidimensional Modeling	43
2.1.1.3 Spatial Data	44
2.1.1.4 Multidimensional and Spatial Data Cubes	45
2.1.2 Logical Design	46
2.1.2.1 Relational Databases	47
2.1.2.2 Spatial Extensions of Database Management Systems	48
2.1.2.3 Relational Representation of Spatial Data Warehouses	49
2.1.3 Physical Design	52
2.1.3.1 Bitmap Index and Bitmap Join Index	52
2.1.3.2 Spatial Indices	54
2.2 Uncertain Data Management	59

2.2.1	Probabilistic Data	60
2.2.1.1	Probabilistic Data Warehouses	61
2.2.1.2	Probabilistic Spatial Data	61
2.2.2	Fuzzy Data	63
2.2.2.1	Fuzzy Data Warehouses	63
2.3	Spatial Vagueness	65
2.3.1	Exact Models for Spatial Vagueness	67
2.3.1.1	Vague Spatial Data Types	68
2.3.1.2	Vague Spatial Geometric Set Operators	70
2.3.1.3	Vague Topological Relationships	71
2.3.2	Fuzzy Models for Spatial Vagueness	72
2.3.2.1	Fuzzy Spatial Data Types	72
2.3.2.2	Fuzzy Spatial Operators	76
2.3.2.3	Fuzzy Spatial Topological Relationships	77
2.3.3	Implementations for Fuzzy Models	78
2.3.3.1	Spatial Plateau Objects	79
2.3.3.2	Lines with Gradual Transitions	80
2.3.3.3	Bitmaps	80
2.3.3.4	Triangulated Irregular Networks	81
2.3.3.5	Fuzzy Minimum Bounding Rectangles	81
2.3.4	Summary	82

CHAPTER 3 – RELATED WORK 85

3.1	Conceptual Modeling and Logical Design of Spatial Data Warehouses	85
3.1.1	The MultiDim Conceptual Model	86
3.1.2	UML Profiles	88
3.1.3	The Spatial Data Warehouse Metamodel	90

3.1.4	Discussion	91
3.2	Spatial Vagueness in Spatial Data Warehouses	92
3.2.1	The Fuzzy Spatial Data Warehouse	92
3.2.2	Conceptual Frameworks for Risk Assessment	94
3.2.3	The RADSOLAP Method	96
3.2.4	Discussion	98
3.3	Indices for Spatial Data Warehouses	99
3.3.1	aR-tree	99
3.3.2	SB-index	101
3.3.3	Discussion	103
3.4	Indices for Vague Regions	104
3.4.1	Vague R-tree	104
3.4.2	FMBR R-tree	105
3.4.3	Discussion	107
3.5	Summary	107

CHAPTER 4 – CONCEPTUAL DESIGN OF VAGUE SPATIAL DATA WAREHOUSES111

4.1	Conceptual Modeling of Vague Spatial Data Warehouses	112
4.2	Attributes	115
4.3	Hierarchies	120
4.3.1	Hierarchy Operator	120
4.3.2	Properties of Hierarchies	123
4.3.3	Categories of Hierarchies	126
4.4	Multidimensional Cube with Vague Spatial Data	127
4.4.1	Dimensions	127
4.4.2	Measures	128
4.4.3	Cube	128

4.4.4	Vague Spatial Fact	130
4.4.5	Lattice of Cuboids	132
4.5	Vague Spatial Predicates	134
4.5.1	Spatial Range Queries	135
4.5.2	The Vague Spatial Range Query	138
4.6	Vague Spatial Aggregation Functions	140
4.6.1	Vague Spatial Union	140
4.6.2	Vague Spatial Intersection	141
4.6.3	Vague Spatial Difference	142
4.7	Vague Spatial Online Analytical Processing	144
4.8	Reusing Existing Models and Implementations	146
4.8.1	Arbitrary Geometries	146
4.8.2	Bitmaps	147
4.8.3	Triangulations	147
4.8.4	Lines with Gradual Transitions	150
4.9	The Vague Spatial MultiDim Conceptual Model	151
4.9.1	Fundamentals	152
4.9.1.1	Data Types	152
4.9.1.2	Certitude and Dubiety	154
4.9.1.3	Vague Topological Constraints	154
4.9.2	Attributes, Levels and Members	155
4.9.3	Dimensions and Hierarchies	157
4.9.4	Fact and Measures	160
4.10	Summary	164

CHAPTER 5 – LOGICAL DESIGN OF VAGUE SPATIAL DATA WAREHOUSES 167

5.1	Relational Representation of Vague Spatial Data Warehouses	168
-----	--	-----

5.2	Implementations for the Vague Spatial Attribute	170
5.2.1	Separate Tables for Certitude and Dubiety	171
5.2.2	A Single Table for Certitude and Dubiety	174
5.2.3	User Defined Types	177
5.2.4	A Pair of Arrays	179
5.2.5	One Multiple Geometry And One Array of Membership Values	181
5.2.6	Monovalued Certitude and Monovalued Dubiety	184
5.2.7	2D Geometry With Measure or 3D Geometry	184
5.2.8	Discussion	186
5.3	Vague Spatial Attribute	187
5.4	Vague Spatial Level and Vague Spatial Member	191
5.5	Hierarchies	194
5.6	Fact and Vague Spatial Measure	195
5.7	Vague Spatial Fact	197
5.7.1	Relational Representation	198
5.7.2	Numeric Measures and Crisp Spatial Measures	199
5.7.3	Vague Spatial Measures	201
5.7.4	Loading a Vague Spatial Fact Table	205
5.7.5	Discussion	206
5.8	Vague Topological Constraints	208
5.8.1	Pairwise Evaluation of Sets of Topological Relationships	208
5.8.2	Hierarchy	211
5.8.3	Spatial Fact	214
5.8.4	Intra Level and Intra Fact	217
5.9	Vague Spatial Online Analytical Processing	219
5.9.1	Accessors	219

5.9.2	Vague Spatial Predicates	222
5.9.3	Vague Spatial Aggregation Functions	228
5.9.4	Slice-and-Dice, Roll-Up, and Drill-Down	229
5.10	Summary	231

CHAPTER 6 – PHYSICAL DESIGN OF VAGUE SPATIAL DATA WAREHOUSES 235

6.1	Indexing Vague Spatial Data Warehouses	235
6.2	Evaluation of a DBMS and of Indices for Spatial Data Warehouses	237
6.2.1	Containment Range Queries against Vague Point Sets	237
6.2.1.1	Workbench and Platforms	238
6.2.1.2	Workload	240
6.2.1.3	Results	240
6.2.2	Intersection Range Queries against Vague Regions	241
6.2.2.1	Workbench and Platforms	242
6.2.2.2	Workload	243
6.2.2.3	Results	245
6.2.3	Vague Spatial Range Queries against Vague Regions	246
6.2.3.1	Workbench and Platforms	247
6.2.3.2	Workload	247
6.2.3.3	Extending the SB-index	249
6.2.3.4	Results	249
6.2.4	Discussion	250
6.3	The Vague Spatial Bitmap Index	251
6.3.1	Maximum Area Inscribed Polygon	252
6.3.2	Data Structure	252
6.3.3	Building Operation	253
6.3.4	Processing Queries containing Spatial Range Queries	256

6.3.4.1	Filtering with a Conservative Approximation	258
6.3.4.2	Filtering with a Conservative Approximation and a Progressive Approximation	259
6.3.4.3	Particularities of Querying the Dubiety	261
6.3.4.4	Calling the Procedures	262
6.3.5	Processing Queries Containing a Vague Spatial Range Query	264
6.3.5.1	Filtering with a Conservative Approximation	266
6.3.5.2	Filtering with a Conservative Approximation and a Progressive Approximation	268
6.3.5.3	Calling the Procedures	270
6.4	Evaluation of the VSB-index	271
6.4.1	Experimental Setup	272
6.4.1.1	Workbench and Platforms	272
6.4.1.2	Workload	275
6.4.2	Intersection Range Queries over the Real Vague SDW	276
6.4.2.1	IRQobject and IRQdubiety	277
6.4.2.2	IRQcertitude	278
6.4.3	Intersection Range Queries over the Synthetic Vague SDW	280
6.4.3.1	IRQobject and IRQdubiety	280
6.4.3.2	IRQcertitude	282
6.4.4	Containment Range Queries over the Synthetic Vague SDW	282
6.4.4.1	CRQobject and CRQdubiety	283
6.4.4.2	CRQcertitude	285
6.4.5	Vague Spatial Range Queries over the Synthetic Vague SDW	285
6.4.5.1	Test Configurations	286
6.4.5.2	Results	287
6.4.6	Building Costs and Storage Requirements	288

6.5 Summary	290
CHAPTER 7 – CONCLUSION AND FUTURE WORK	295
REFERENCES	301
GLOSSARY	317
APPENDIX A – USER-DEFINED FUNCTIONS	319
APPENDIX B – PROCEDURES OF THE VSB-INDEX	323

Chapter 1

INTRODUCTION

Decision-making support has gained the attention of researchers of data warehouse (DW), online analytical processing (OLAP), and geographic information system (GIS). Spatial data warehouse (SDW) and spatial online analytical processing (SOLAP) are remarkable achievements that resulted from their research. A SDW is a subject-oriented, integrated, time-variant, voluminous, non-volatile and multidimensional database that stores spatial data and conventional data (numeric and alphanumeric) (STEFANOVIC; HAN; KOPERSKI, 2000; FIDALGO et al., 2004; SILVA et al., 2010; VAISMAN; ZIMÁNYI, 2014b). Fact, dimension, and hierarchy are core concepts of a SDW. A fact denotes the scores of business activities through numeric measures or spatial measures, while dimensions hold conventional attributes and spatial attributes that contextualize the values of the measures. Hierarchies associate dimension's attributes of coarser and finer granularity. When these attributes are spatial, the association is described by a topological relationship (e.g. within).

SOLAP aids the decision-making process by providing spatial analysis combined with multidimensional analytical queries that run over the SDW (BÉDARD; MERRETT; HAN, 2001; BIMONTE; TCHOUNIKINE; MIQUEL, 2005). SOLAP operations process both conventional and spatial data. For instance, a roll-up operation aggregates not only numeric data with functions such as sum, but also aggregates spatial data using the geometric union (STEFANOVIC; HAN; KOPERSKI, 2000; SILVA et al., 2008; GOMEZ et al., 2009; BOULIL; BIMONTE; PINET, 2015). In addition, spatial range queries select spatial data that satisfy a given topological relationship (e.g. intersects and within) with respect to an ad hoc spatial query window (PAPADIAS et al., 2001; SIQUEIRA et al., 2010).

In SDWs, spatial data are modeled either as spatial objects or as continuous fields (STEFANOVIC; HAN; KOPERSKI, 2000; VAISMAN; ZIMÁNYI, 2009; BIMONTE; KANG, 2010; GASCUEÑA; GUADALUPE, 2009; VAISMAN; ZIMÁNYI, 2014b; BOULIL; BIMONTE; PINET, 2015). Spatial ob-

jects are crisp in the sense that they assume definite extent, boundary, and shape, e.g. the territory of a city. They are commonly implemented using vector geometries provided by the underlying DBMS (HOEL, 2008). Continuous fields represent phenomena that change continuously in space, e.g. elevation (CÂMARA; FREITAS; CASANOVA, 1995; CÂMARA et al., 2014). They are often implemented as raster and triangulated irregular networks (HOEL, 2008).

However, spatial data are susceptible to imperfections. Spatial vagueness is one kind of spatial data imperfection concerning the difficulty of distinguishing an object shape from its neighborhood, since the concepts used to describe the spatial information are not defined precisely (WORBOYS, 1998; BEJAOU, 2009). As a result, it is not possible to be sure if the parts of a spatial object belong completely to it, or possibly belong to it, or simply do not belong to it (PAULY; SCHNEIDER, 2010). A vague spatial object has an extent, but it inherently cannot or does not have a precisely definable boundary and/or interior. Consequently, it is not rigorously bounded by a sharp line and might have a blurred interior (BURROUGH, 1996; SCHNEIDER, 2014). Spatial vagueness has been considered in applications for agriculture (BEJAOU et al., 2009; EDOH-ALOVE et al., 2014), coastal erosion (DILO, 2006; JADIDI et al., 2014), volcanic crisis management (PEREZ; SOMODEVILLA; PINEDA, 2010), among others (BURROUGH; FRANK, 1996).

Spatial vagueness has been addressed by distinct models that assign membership degrees to vague concepts: exact models (COHN; GOTTS, 1996; CLEMENTINI; FELICE, 1996; BEJAOU et al., 2009, 2010; PAULY; SCHNEIDER, 2010) and fuzzy models (DILO; BY; STEIN, 2007; SCHNEIDER, 2008; TANG; KAINZ; WANG, 2010; HAZARIKA; HAZARIKA, 2012; SCHNEIDER, 2014). Exact models are mostly based on crisp spatial data types, while fuzzy models adopt the fuzzy set theory as their mathematical fundamentals. There is not a commonly accepted terminology to define concepts related to imperfections of spatial data (DEVILLERS et al., 2010). In this thesis, the terms vague and fuzzy differ exclusively in the approach used to define the model. Therefore, vague spatial data can be modeled either according to a vague spatial data type from an exact model or to a fuzzy spatial data type from a fuzzy model.

Existing models for representing spatial vagueness do not provide multidimensional analysis combined with spatial analysis. However, such combination is essential for a SOLAP tool to query a SDW. Spatial vagueness was tackled exclusively in data cleansing and integration tasks, before the design of the SDW schema, for natural phenomena whose indeterminate boundaries need to be re-measured at different epochs (BÉDARD; MERRETT; HAN, 2001). Since the design of SDWs typically considers crisp spatial objects, it prevents some types of decision-support analysis to be carried out by SOLAP (BÉDARD; MERRETT; HAN, 2001). These shortcomings

gained the attention of researchers and recent work in the literature consider spatial vagueness in the design of SDWs (PEREZ; SOMODEVILLA; PINEDA, 2010; JADIDI et al., 2013; EDOH-ALOVE; BIMONTE; BÉDARD, 2014; EDOH-ALOVE et al., 2014). Those work rely upon the advances on multidimensional modeling (VAISMAN; ZIMÁNYI, 2014b; BOULIL; BIMONTE; PINET, 2015), systems of both vague and fuzzy spatial data types (BURROUGH; FRANK, 1996; DILO; BY; STEIN, 2007; PAULY; SCHNEIDER, 2010), and implementations for these systems (VERSTRAETE et al., 2005; SCHNEIDER, 2014).

Nevertheless, existing work focus exclusively on conceptual design of SDWs (JADIDI et al., 2013), or on logical design of SDWs (PEREZ; SOMODEVILLA; PINEDA, 2010), or do not consider different models to represent spatial vagueness in the design of SDWs (EDOH-ALOVE; BIMONTE; BÉDARD, 2014; EDOH-ALOVE et al., 2014). Furthermore, the physical design of a SDW does not comprise an index to improve the performance of multidimensional queries extended with spatial predicates involving vague spatial data. Rather, existing indices process either multidimensional queries (WU et al., 2009; LANE; POTINENI, 2014), or spatial predicates involving crisp spatial data (GUTTMAN, 1984; AOKI, 1998; ORACLE..., 2014; OBE; HSU, 2015), or spatial predicates involving vague spatial data (PETRY; LADNER; SOMODEVILLA, 2007), or multidimensional queries extended with spatial predicates involving crisp spatial data (PAPADIAS et al., 2001; SIQUEIRA et al., 2012b).

On the other hand, in this thesis, spatial vagueness is considered to design and query the SDW. Consequently, the vague spatial data warehouse (vague SDW) and the vague spatial on-line analytical processing (vague SOLAP) are obtained. The results of the investigations on conceptual modeling, logical design, and physical design of the vague SDW have provided the following contributions:

- the Vague Spatial Cube (VSCube) conceptual model that enables the creation of conceptual schemata for vague SDWs using data cubes and provides vague spatial aggregation functions and vague spatial predicates to query vague spatial data from measures and dimensions;
- the Vague Spatial MultiDim (VSMultiDim) conceptual model that provides visual representations for the concepts and enables the creation of conceptual schemata for vague SDWs using diagrams;
- guidelines for designing relational schemata and integrity constraints for vague SDWs, and for extending the SQL language to enable vague SOLAP;
- the Vague Spatial Bitmap Index (VSB-index), which improves the performance to process

queries against vague SDWs.

In order to illustrate the applicability of the vague SDW and the aforementioned contributions, two case studies concerning agriculture are described and used throughout this thesis to exemplify concepts. The pest control case study described in Section 1.1 addresses a real problem whose existing solutions mostly neglect spatial vagueness. The huanglongbing (HLB) case study detailed in Section 1.2 tackles a real problem in citriculture that has been investigated by the Brazilian Agricultural Research Corporation (Embrapa), but that had not yet been addressed using a vague SDW. Section 1.3 outlines the chapters of this thesis and summarizes the aforementioned contributions.

1.1 The Pest Control Case Study

Precision agriculture has been adopted in the management of farms to optimize returns on inputs while preserving resources. Spatial and temporal characteristics of crops are processed in decision support systems (INAMASU et al., 2011). For instance, remote sensing images are provided as input to classification algorithms, e.g. k-means, for identification of crops and their characteristics (RECIO et al., 2013). The goal is to map the crop in order to apply a suitable pesticide rate. The output of the classification comprises some crops as crisp regions and several crops as vague regions.

Figure 1.1a shows a remote sensed image and crops highlighted inside a white dashed rectangle. One crop is identified as being crisp and is shown in Figure 1.1b: C_6 . On the other hand, the remaining crops intrinsically have a vague shape, e.g. the crop C_1 has two parts that belong to it in black and one part that may belong to it in gray, and the crop C_5 has one part that belongs to it in black and four parts that may belong to it in gray.

The combination of remotely sensed imagery and variable rate technology enables the application of pesticides at specific locations (RECIO et al., 2013). Pesticides such as herbicides, insecticides, fungicides and defoliantes are applied aerially to crops, with some fraction intercepted by foliage and some fraction reaching the soil. Methods of application vary from ground equipment to aircraft, and the material can be applied as solids, dispersions, emulsions or solutions (LEONARD; KNISEL; STILL, 1987). Pesticides are not applied uniformly over the extent, due to variable spraying according to the weed area, real-time sensory and fuzzy control used to regulate dose of drugs (SHI et al., 2008). In addition, dissipation, plant transpiration and runoff influence the distribution of pesticide in a given extent (NEITSCH; ARNOLD; WILLIAMS, 2011). For instance, the crop C_1 and the area A_1 where a given pesticide was applied to, shown in Fig-

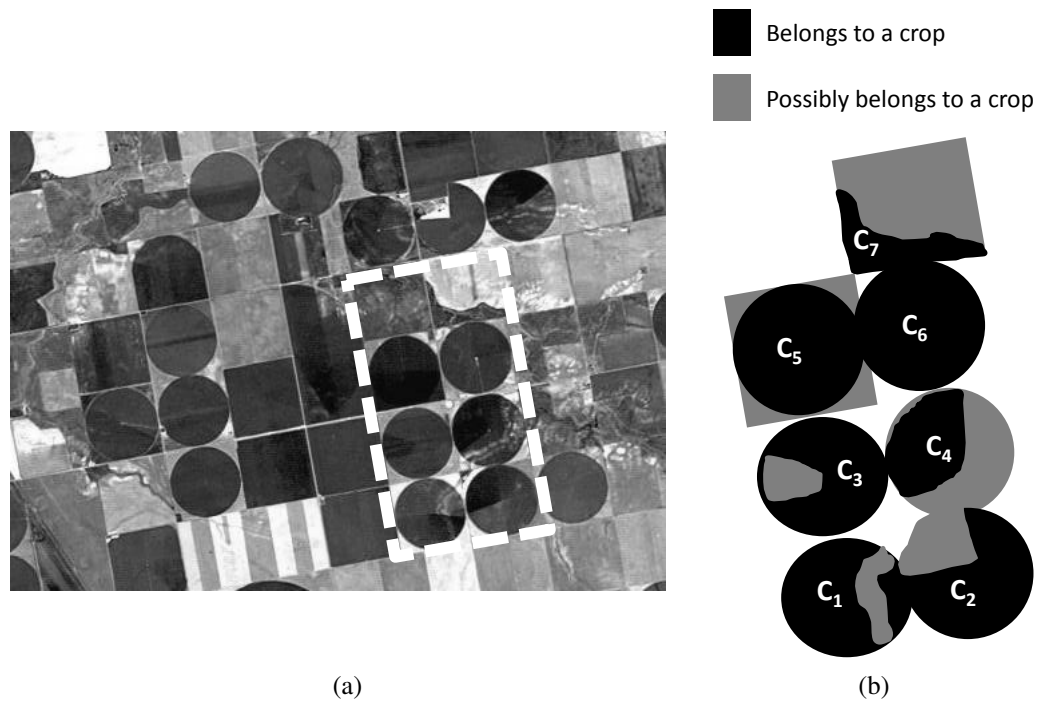


Figure 1.1: Crops in remote sensed image and after identification. (a) A remote sensed image (adapted from Crop Circles in Kansas, by NASA Image of the Day Gallery). (b) Identified crops.

ure 1.2a, indicate that the possibility of application is higher in the darker gradient than in the brighter gradient.

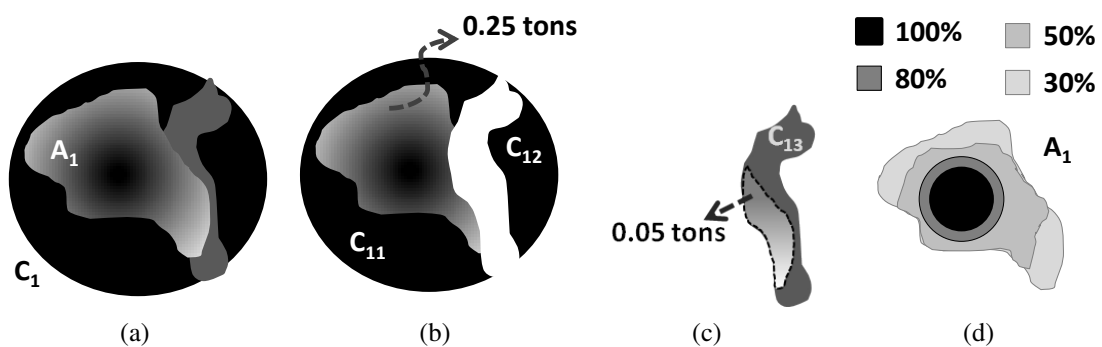


Figure 1.2: An applied area of pesticide A_1 over the crop C_1 . (a) A_1 over C_1 . (b) A_1 over C_{11} . (c) A_1 over C_{13} . (d) Subsets of A_1 .

In addition to estimate where pesticides were applied, another relevant measure in the study is the amount in tons of pesticides applied. For instance, considering that 0.3 t were applied over A_1 , some portions of pesticides were applied over parts of crops, as follows: 0.25 t were applied over the part C_{11} that certainly belongs to C_{11} , as shown in Figure 1.2b; no pesticide was applied over the part C_{12} that certainly belongs to C_1 ; and 0.05 t were applied over the part C_{13} that may belong to C_1 , in the extent circumscribed by a dashed line in Figure 1.2c. A discrete represen-

Table 1.1: Examples of queries for the analysis of the pest control activities.

Query	Description
PC1	Retrieve the amount of pesticides applied and areas where pesticides were applied to by pesticide by date by crop.
PC2	Retrieve the amount of pesticides applied and areas where pesticides were applied to by pesticide by date by parts that certainly (or possibly) belong to a crop.
PC3	Retrieve the amount of pesticides applied and areas where pesticides were applied to by pesticide type by month by agricultural land.
PC4	Retrieve the amount of pesticides applied and areas where pesticides were applied to by pesticide type by month by agricultural land, for herbicides applied in 2012 whose applied areas certainly intersect a spatial query window provided by the user.

tation of a continuous applied area is obtained as subsets having a possibility of application, according to Figure 1.2d, for A_1 and possibilities of 100%, 80%, 50% and 30%. Such representation use geometric shapes and the corresponding possibility (membership) values. Dates of pesticide applications are also recorded.

Agricultural lands from different owners are identified through the same procedure and have broad boundaries. Each agricultural land maintains several crops, but one crop is associated to at most one agricultural land. Aiming at correlating water quality and pesticide use, watersheds were included in the application and gathered from the database of the hydrological local department. Each watershed contains several agricultural lands.

Crops, areas of pesticide application, and agricultural lands are affected by spatial vagueness, while watersheds are crisp regions. The areas where pesticides were applied to and the amounts of applied pesticides in tons are reported by pesticide by crop by date to benefit the analysis of the agricultural and the environmental impacts. Table 1.1 lists a few queries used for the analysis of the pest control activities.

1.2 The HLB Case Study

Huanglongbing (HLB), also known as greening, is the world's most serious citrus disease (BOVÉ, 2014). A bacteria-carrying psyllid injects the bacterium into a healthy tree while feeding from it. Visual inspections of all trees of the plot are carried out monthly throughout multiple years to detect symptomatic and infected trees. The symptom severity is a value estimated after inspecting leaves and branches of a tree (PUSTIKA et al., 2008). Although infected trees have

often high symptom severity, some infected trees show no symptoms during a period.

Studying the spatial distribution of infection is valuable to motivate control practices, e.g. eradication of symptomatic trees and replacement by healthy trees. HLB spread is analyzed (GOTTWALD, 2010): (i) by tree, by plot, by farm, by city; and (ii) by group of certainly and possibly infected trees, by region with several groups. Given a certainly infected tree, adjacent trees within row and across row are possibly infected. The possibility of infection is higher as closer a tree is from an infected tree. A group may encompass up to 572 trees and the distance between groups was once estimated to be within 25 to 30 meters (GOTTWALD; GRAÇA; BASSANEZI, 2007). The edge effect determines that infected regions often overlap more than one plot, mainly when neighbor farms do not comply with a regional policy of HLB control. Furthermore, a recent study reported decreasing psyllid abundance with increased elevation, e.g. no psyllids were collected at an elevation above 600 meters (JENKINS; HALL; GOENAGA, 2015).

The Brazilian Agricultural Research Corporation (Embrapa) studies HLB. For instance, Jorge & Inamasu (2014) described an application whose main features are the following. Farms, their plots, and plot's trees are mapped. All of them have exact location and boundaries. A farm has an owner and comprises several plots. In a plot, trees of a given plant are grown. Monthly, inspectors examine all trees and annotate a status, i.e. "healthy", "infected", or "eradicated". The dataset used by Jorge & Inamasu (2014) has been kindly provided for use in this thesis. In order to comply with the characteristics of HLB infection, the following features extend the cited application.

The HLB infection is mapped monthly. An inspector examines a tree and estimates a possibility of infection in $[0,1]$ to the tree's location. The possibility of infection indicates whether the tree's location belongs to the HLB infection and quantifies the membership degree. For each location where the possibility of infection is 1, locations of adjacent trees within row and across row are assigned a possibility of infection in $]0,1[$ according to a distance function. An infected group is composed of adjacent locations that had a possibility of infection assigned.

After the complete examination of the farm, the extent occupied by the HLB infection is outlined as a region that covers one or more infected groups. Clustering techniques can be applied to outline infected regions, e.g. α -hull (EDELSBRUNNER; KIRKPATRICK; SEIDEL, 1983) and fuzzy c-means (BEZDEK; EHRLICH; FULL, 1984). An infected region is composed both by parts where the possibility of infection is 1 and by parts where the possibility of infection is in $]0,1[$. It may overlap more than one plot.

Figure 1.3a shows a remote-sensed image of citrus plots and labels the plots Plot1 and Plot2.

Figure 1.3b zooms into the rectangular area highlighted in Figure 1.3a and shows two certainly infected trees recognized in a given month: the red and the orange, each one in a plot. These trees belong to the red group and to the orange group as shown in Figure 1.3c. Each tree in a group has a possibility of infection. Both the orange group and the red group belong to the same infected region shown in Figure 1.3d, which overlaps both Plot1 and Plot2, even though a road separates these plots. Figure 1.3e shows a discrete representation of the continuous infected region exemplified in Figure 1.3d, according to the possibility of infection.

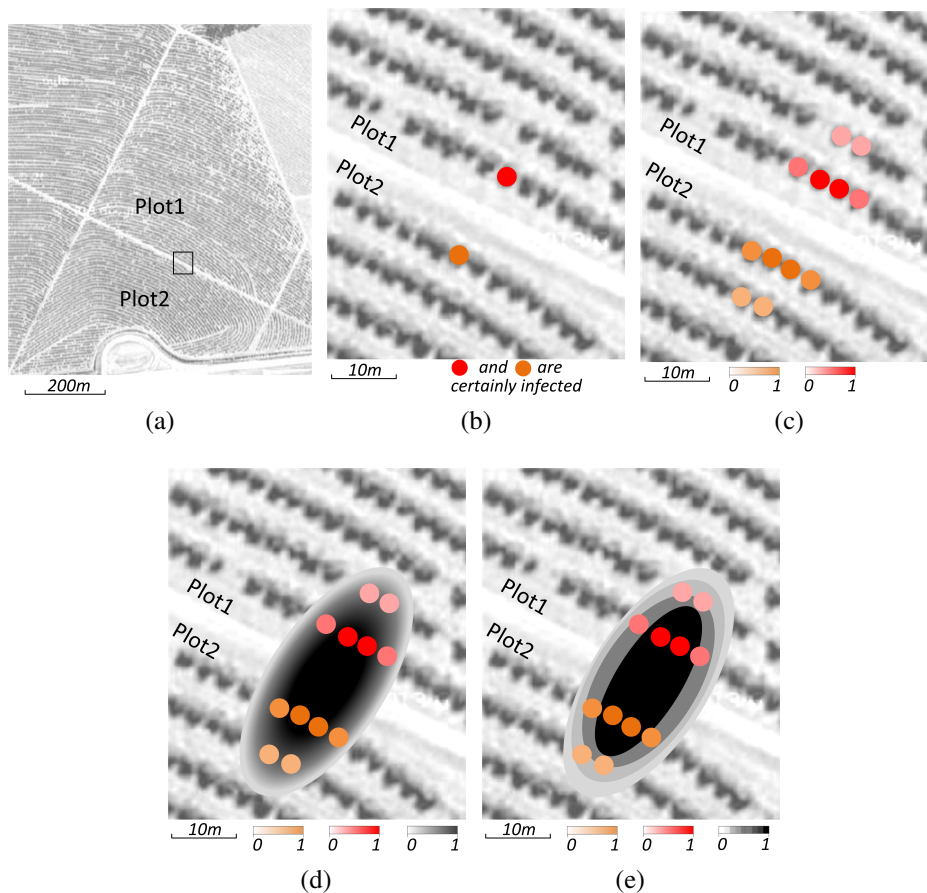


Figure 1.3: HLB infection. (a) Plots. (b) Two infected trees. (c) Two infected groups. (d) Two groups and a continuous representation for an infected region. (e) Two groups and a discrete representation for an infected region.

In addition to examine a tree and estimate a possibility of infection, the inspector also assigns the HLB symptom severity as a value within 0 and 9. Furthermore, the inspector indicates whether the examined tree must be eradicated or not. If a tree is eradicated, then its eradication date is assigned. Therefore, each tree must also have a date when it was planted, while the date of eradication due to HLB infection is initially unknown. Table 1.2 lists a few queries used for the analysis of HLB control.

Table 1.2: Examples of queries for the analysis of of HLB control.

Query	Description
HLB1	Retrieve an infection's location and possibility and the symptom severity by tree by infected group by month by inspector.
HLB2	Retrieve infections' location and maximum possibility by plot by quarter of 2014.
HLB3	Retrieve the average symptom severity by infected group by month in 2014.
HLB4	Retrieve the number of eradicated trees by plot by year by team.
HLB5	Retrieve the number of eradicated trees by year by team, such that infected regions intersect a spatial query window provided by the user.
HLB6	Retrieve the number of certainly infected trees located between 500m and 600m by plot by month in a given city between January and June, 2014.
HLB7	Retrieve infected regions identified in a given city between January and March, 2014, such that infection possibility was greater than 80%.

1.3 Thesis Organization

Chapter 2 addresses the main concepts that are necessary to comprehend this thesis, concerning SDW and spatial vagueness. Firstly, fundamentals of conceptual modeling, logical design and physical design for SDWs are surveyed. Secondly, spatial vagueness is characterized and the fundamentals of the main approaches for spatial vagueness are summarized. Exact models, fuzzy models and implementations for fuzzy models are revised according to their data types, operators, and topological relationships.

Chapter 3 surveys and discusses existing work in the literature that are related to this thesis. The main characteristics of existing conceptual models for SDW are reported in order to identify requirements of a conceptual model for vague SDWs. Similarly, existing work aimed at the logical design of SDWs are addressed to determine a baseline for the logical design of vague SDWs. Furthermore, indices for SDWs have their data structures and query processing algorithms studied to verify the adequacy for indexing vague spatial data. Finally, existing indices for vague regions are also surveyed regarding the data types they support, the query processing algorithms they offer, and the spatial predicates they resolve.

Chapter 4 addresses conceptual design of vague SDWs and describes two major contributions of this thesis: the VSCube conceptual model and the VSMultiDim conceptual model. The VSCube conceptual model is based on the multidimensional data cube and supports vague

spatial objects as geometric shapes and their corresponding membership values, both in dimensions or as measures in a fact. Furthermore, values of measures are allowed to be assigned to parts of a vague spatial object. Vague spatial aggregation functions (e.g. vague spatial union), vague spatial predicates (e.g. vague spatial range queries) and vague SOLAP operations (e.g. drill-down and roll-up) are described to enable spatial analysis combined with multidimensional exploitation of the data cube. The VSMultiDim conceptual model extends an existing conceptual model for SDWs by enabling vague spatial attributes in dimensions or as measures in a fact, and by specifying constraints involving vague spatial data. Differently from the VSCube model, the VSMultiDim model provides a graphical notation to allow the creation of diagrams that represent the multidimensional conceptual schema of the vague SDW. Both the VSCube and the VSMultiDim models overcome related work in expressiveness, as those models tackle both vague spatial data types and fuzzy spatial data types.

Chapter 5 focuses the logical design of vague SDWs underlying relational database management systems (DBMSs) with an extension for spatial data. Alternative logical designs for vague spatial attributes are duly described and compared. Mapping rules are provided to transform the conceptual schema of a vague SDW into a logical schema. These mapping rules address attributes, dimensions, hierarchies, measures, and fact. In addition, an specific design for the fact allows values of measures to be assigned to parts of a vague spatial object. Vague SOLAP is enabled by extending SQL and implementing operations for accessing vague spatial data, processing vague spatial predicates and computing aggregation of vague spatial data. Constraints are specified and implemented to maintain the integrity of the vague SDW. The guidelines for designing schemata and constraints for SDWs are also part of the major contributions of this thesis.

Chapter 6 focuses the physical design of vague SDWs and describes one of the major contributions of this thesis: the VSB-index. Firstly, an experimental evaluation of a database management system and of existing indices for SDWs identify bottlenecks and their limitations to provide a reasonable performance to process queries over vague SDWs. Secondly, the VSB-index is introduced and described. Thirdly, an experimental evaluation of the VSB-index is described and the benefits for the performance to process queries over vague SDWs are discussed.

The development of this doctoral research project resulted in the following publications that are related to this thesis:

1. Siqueira, Thiago Luís Lopes; Ciferri, Cristina Dutra de Aguiar; Times, Valéria Cesário; Ciferri, Ricardo Rodrigues. Modeling vague spatial data warehouses using the VSCube

- conceptual model. *Geoinformatica* (Dordrecht. Online), v. 18, p. 313-356, 2014.
2. Siqueira, Thiago Luís Lopes; Ciferri, Ricardo Rodrigues; Zimányi, Esteban. Extending the MultiDim conceptual model to enable the design of vague spatial data warehouses. In: Dutch Belgian Database Day, 2014, Anvers, Belgium. Online abstract available at: http://adrem.ua.ac.be/sites/adrem.ua.ac.be/files/abstract_14.pdf.
 3. Siqueira, Thiago Luís Lopes; Ciferri, Cristina Dutra de Aguiar; Times, Valéria Cesário; Ciferri, Ricardo Rodrigues. Towards Vague Geographic Data Warehouses. In: International Conference on Geographic Information Science, 2012, Columbus, OH, USA. Lecture Notes in Computer Science. New York: Springer-Verlag, 2012. v. 7478. p. 173-186.
 4. Siqueira, Thiago Luís Lopes; Mateus, Rodrigo Costa; Ciferri, Ricardo Rodrigues; Times, Valéria Cesário; Ciferri, Cristina Dutra de Aguiar. Querying Vague Spatial Information in Geographic Data Warehouses. In: The 14th AGILE International Conference on Geographic Information Science, 2011, Utrecht, Netherlands. Lecture Notes in Geoinformation and Cartography: Advancing Geoinformation Science for a Changing World, 2011. v. 1. p. 379-397.
 5. Siqueira, Thiago Luís Lopes; Oliveira, João Celso Santos; Times, Valéria Cesário; Ciferri, Cristina Dutra de Aguiar; Ciferri, Ricardo Rodrigues. Indexing Vague Regions in Spatial Data Warehouses. In: XIV Brazilian Symposium on Geoinformatics, 2013, Campos do Jordão, SP, Brazil. XIV Brazilian Symposium on Geoinformatics, Proceedings, 2013. p. 158-169.
 6. Siqueira, Thiago Luís Lopes; Oliveira, João Celso Santos; Times, Valéria Cesário; Ciferri, Cristina Dutra de Aguiar; Ciferri, Ricardo Rodrigues. Indexing and Querying Vague Spatial Data Warehouses. *Journal of Information and Data Management - JIDM*, v. 5, p. 161-170, 2014.

The article number 1 describes the VSCube conceptual model in the journal's special issue entitled "Spatial Data Warehouses and SOLAP" and provides most of the content for Chapter 4. The paper number 2 summarizes the VSMultiDim conceptual model, which is also addressed in Chapter 4. The paper number 3 tackles the logical design of vague SDWs reusing the relational model and provides substantial findings and content for Chapter 5. Three publications based on the elaboration of Chapter 6, as follows. The paper number 4 processes queries in a vague SDW using both a DBMS system and an existing index for SDW, and then compares and discusses

the results. The paper number 5 introduces the VSB-index for vague SDWs and evaluates its performance. It was awarded the 3rd best paper of the that conference. The article number 6 uses the VSB-index in a vague SDW built using a real dataset provided by Embrapa, evaluates the performance to process queries and discusses the main findings.

Chapter 2

THEORETICAL FOUNDATIONS

This chapter addresses theoretical foundations of spatial data warehouses in Section 2.1, of uncertain data management in Section 2.2, and of spatial vagueness in Section 2.3.

2.1 Spatial Data Warehouse Design

A spatial data warehouse (SDW) is a multidimensional, integrated, subject-oriented, historic and non-volatile database that stores conventional data like a data warehouse (DW), but additionally stores spatial data (STEFANOVIC; HAN; KOPERSKI, 2000; BÉDARD; MERRETT; HAN, 2001; FIDALGO et al., 2004; SILVA et al., 2010; VAISMAN; ZIMÁNYI, 2014b). While OLAP provide multidimensional queries that aggregate huge volumes of conventional data stored in a DW (HARINARAYAN; RAJARAMAN; ULLMAN, 1996; POURABBAS; RAFANELLI, 1999; KIMBALL; ROSS, 2002; CIFERRI et al., 2013), spatial analysis together with agile and flexible multidimensional analytical operations are provided by spatial OLAP (SOLAP) (BÉDARD; MERRETT; HAN, 2001; FIDALGO et al., 2004; BADARD; DUBÉ, 2009; BIANCHI; HATANO; SIQUEIRA, 2013; VAISMAN; ZIMÁNYI, 2014b). The database design encompasses the conceptual modeling, the logical design and a physical design (ELMASRI; NAVATHE, 2010). Since the SDW is a database, its conceptual modeling, logical design and physical design are addressed in Sections 2.1.1, 2.1.2 and 2.1.3, respectively.

2.1.1 Conceptual Modeling

Designers typically use conceptual models or languages for representing and conceptualizing abstractions, which consist of essential, relevant, or important parts of an application as interpreted by designers (THALHEIM, 2009). A conceptual model represents data in terms of

named sets of objects, named sets of values, named sets of relationships, and their corresponding constraints, as well as often use graphical symbols to represent data semantics (EMBLEY, 2009b). The conceptual schema of a database is a high-level description that is natural and direct for users of the database that does not take into account implementation details (BORGIDA; MYLOPOULOS, 2009).

Databases are usually designed at the conceptual level using some variation of the Entity-Relationship model or the Unified Modeling Language (ELMASRI; NAVATHE, 2010). They are outlined in Section 2.1.1.1. Furthermore, multidimensional models have been widely used to produce data schemas for multidimensional databases and to allow OLAP tools to present results of data aggregation in a multidimensional fashion (PEDERSEN, 2009). Multidimensional models are addressed in Section 2.1.1.2. Moreover, conceptual modeling of SDWs is not only related to elaboration of a conceptual data schema, but involves choosing between two alternative conceptualizations of spatial data: spatial object or continuous field, which are outlined in Section 2.1.1.3.

2.1.1.1 Entity-Relationship Model and Unified Modeling Language

The Entity-Relationship (E-R) model (CHEN, 1976; SONG; CHEN, 2009) represents the information structure of a problem domain in terms of entities and relationships. An entity is an object in the real world with an independent existence and is characterized by its properties called attributes. An entity type defines a collection of entities that have the same attributes. An identifier is a non-empty set of attributes whose values uniquely identify the entities. An identifier is also called key.

A relationship is a relation among entities. A relationship type relates entities of entity types. The cardinality is a constraint on a relationship type and stipulates the maximum number of entities that can be related with another entity via a relationship type. For instance, the cardinality of a relationship between two entity types is one-to-one (1:1), one-to-many (1:N) or many-to-many (M:N). The elaboration of a conceptual schema with the E-R model maps categories of individuals into entity types, models relations of entity types as relationship types and uses attributes to denote qualities and values of individuals that belong to entity types or relationship types (BORGIDA; MYLOPOULOS, 2009).

Since the E-R model facilitates the identification of data and constraints, it has been widely used in database design to produce the conceptual schema of a database as an E-R diagram. Different symbols are used to produce an E-R diagram according to the concept being represented, e.g. a rectangle for an entity type and a diamond for a relationship type (ELMASRI;

NAVATHE, 2010). These symbols improve the comprehension of application requirements and allow a better communication between designers and users.

The Unified Modeling Language¹ (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a system such as conceptual database schemata (GOGOLLA, 2009a). In the UML class diagram, a class is a descriptor for a set of objects that share the same structure and behavior. Object properties are described by attributes that assume data types. An identifier plays the same role as described for the E-R model. An association is a connection among a collection of classes and has a name. A multiplicity constrains the connections of classes linked by an association. The UML also defines the Object Constraint Language² (OCL) that allows the specification of integrity constraints (GOGOLLA, 2009b).

2.1.1.2 Multidimensional Modeling

Multidimensional models provide a data cube and categorize data as being either facts with associated measures, or as being dimensions that characterize facts (CODD; CODD; SALLEY, 1993; PEDERSEN, 2009; CIFERRI et al., 2013). A dimension provides perspectives to analyze data and encompasses attributes with different granularity that are hierarchically organized in levels of aggregation. Let L_{child} and L_{parent} be levels of aggregation. The operator \preceq imposes a partial order on L_{child} and L_{parent} , such that $L_{parent} \preceq L_{child}$ if and only if values of measures for L_{parent} can be computed using values of measures of L_{child} (HARINARAYAN; RAJARAMAN; ULLMAN, 1996). A level may consist of a single attribute (GOLFARELLI; MAIO; RIZZI, 1998), or comprise a few attributes and be analogous to an entity-type (MALINOWSKI; ZIMÁNYI, 2009). An instance of a level is called member. A fact relates levels and its attributes called measures are the subject of analysis. Measure values are summarized by traversing hierarchies and using aggregation functions. An instance of a fact is called fact member. Although the term “cube” implies three dimensions, a cube can have an arbitrary number of dimensions.

The views of a data cube can be organized by a directed acyclic graph often called lattice of cuboids, where each cuboid is a view of the data cube (HARINARAYAN; RAJARAMAN; ULLMAN, 1996). An edge links a view of finer granularity to a view of coarser granularity. The view with the finest granularity indicates values of measures detailed for the attributes with the finest granularity of each dimension, which are their (surrogate) keys. A view of coarser granularity has summarized values of measures, as it simply aggregates all values of one or more dimensions,

¹<http://www.omg.org/spec/UML/>

²<http://www.omg.org/spec/OCL/>

or it refers to attributes of coarser granularity according to hierarchies.

OLAP operations exploit dimensions and hierarchies of the data cube and enable multidimensional analysis (HARINARAYAN; RAJARAMAN; ULLMAN, 1996; CHAUDHURI; DAYAL, 1997; VAISMAN; ZIMÁNYI, 2014a). The *pivot* stands for switching the axis of the dimensions of the cube. A *slice* is the selection of fixed values in attributes of dimensions, while a *dice* is the selection of ranges of values from attributes of dimensions of the cube. The aggregation of measure values done by traversing a hierarchy from levels with finer granularity to levels with coarser granularity produces summarized results and is known as *roll-up*. Conversely, traversing a hierarchy from levels with coarser granularity to levels with finer granularity produces detailed results and is called *drill-down*.

2.1.1.3 Spatial Data

A spatial object has a descriptive component denoted by a set of conventional attributes and a spatial component that describes the location and the shape of the object in the space of interest (GOODCHILD, 1992; CÂMARA; FREITAS; CASANOVA, 1995). A spatial object has a spatial data type assigned to it, such as Point, Line and Region (Surface) (GOODCHILD, 1992; PARENT; SPACCAPIETRA; ZIMÁNYI, 2006). A point is 0-dimensional and often denotes a discrete location, such as an address. A line is 1-dimensional and usually represents linear features, such as rivers and roads. A region is 2-dimensional and regularly refer to extents, such as the extent of a county. More complex data types as Point Set, Line Set and Region Set (Surface Set) also exist and are more effective to represent real world phenomena (PARENT; SPACCAPIETRA; ZIMÁNYI, 2006). The implementation of spatial objects is commonly based on vector geometries, as explained in Section 2.1.2. For instance, a polygon represents the 2D region occupied by a city.

A continuous field represents a phenomenon that continuously change in space and/or time (GOODCHILD, 1992; CÂMARA; FREITAS; CASANOVA, 1995). Conceptually, a continuous field can be represented as a function that assigns to each point in space a value of a domain. For example, a continuous field for altitude varies in space, while a continuous field for temperature varies both in space and time. As for the altitude, the value for a given point in space is numeric. The functions are partial since they might be undefined at some points in space. The implementation of continuous fields is commonly based on tessellations, whose tiles are geometric shapes that do not overlap and neither have gaps. Digital elevation models provide tessellations, e.g. bitmap (raster-based) and triangulated irregular network (vector-based) (HOEL, 2008).

Topological relationships describe how spatial data are related in real world. Examples

of such relationships are *meets (touches)*, *contains*, *inside (within)*, *equals*, *overlaps*, *intersects*, *covers*, *covered by* and *disjoint* (EGENHOFER; FRANZOSA, 1991; CLEMENTINI; SHARMA; EGENHOFER, 1994; PARENT; SPACCAPIETRA; ZIMÁNYI, 2006). For instance, Belgium contains Brussels, while Brazil covers the state of São Paulo.

In the 9-intersection model (EGENHOFER; FRANZOSA, 1991), topological relationships between a pair of spatial objects A and B are interpretations of a 3×3 matrix. The matrix' cells indicate the disjointness F or the intersection T among interiors $^\circ$, boundaries ∂ and exteriors $'$ of the objects. The dimensionality of the intersection can also be specified, i.e. 0, 1 or 2. For instance, $A \text{ contains } B$ holds if there is intersection between their interiors and the exterior of A is disjoint from both the interior and boundary of B . Since both the dimensionality of the intersection and the values of the other cells are irrelevant (*), a simplified version of the matrix for A contains B is:

$$\begin{array}{c} B^\circ \quad \partial B \quad B' \\ A^\circ \begin{pmatrix} T & * & * \\ \partial A \begin{pmatrix} * & * & * \\ A' \begin{pmatrix} F & F & * \end{pmatrix} \end{pmatrix} \end{pmatrix} \end{array}$$

Topological relationships also allow querying spatial data. For instance, a spatial exact match query compares two spatial objects and yields the truth value whether the topological relationship *equals* is satisfied (GAEDE; GUNTHER, 1998). Conversely, a spatial range query compares a spatial object to a rectangular spatial query window and yields the truth value whether a given topological relationship is held (GAEDE; GUNTHER, 1998). In this sense, an intersection range query refers to the topological relationship *intersects* and a containment range query refers to the topological relationship *inside/within*.

2.1.1.4 Multidimensional and Spatial Data Cubes

Spatial data have been included in data cubes as spatial attributes in dimensions and as spatial measures in facts. Stefanovic, Han & Koperski (2000), Bimonte, Tchounikine & Miquel (2005) focused on spatial objects, while Vaisman & Zimányi (2009), Bimonte & Kang (2010) tackled continuous fields.

Stefanovic, Han & Koperski (2000) defined three types of spatial dimensions. A *non-spatial dimension* does not have any spatial levels or spatial attributes. In a *spatial to non-spatial dimension*, spatial levels are those with the lowest granularities while non-spatial levels have higher granularities. In a *spatial to spatial dimension*, all the levels are spatial levels. The aforementioned types of dimensions are also used to represent the SDW as a lattice of cuboids.

The aggregation of numeric measures is performed using well-known aggregation functions such as SUM and AVG. Conversely, the aggregation of the spatial measure is performed using geometric functions such as geometric union (region merge) (WANG et al., 2004), for example. The spatial measure can be a composite attribute rather than an atomic attribute (BIMONTE; TCHOUNIKINE; MIQUEL, 2005). Then, an *aggregation mode* is essential to ensure the use of the adequate aggregation function on each field of the composite attribute representing a spatial measure. The use of SUM for a numeric measure may imply the use of union for a spatial measure (SILVA et al., 2008).

Spatial OLAP (SOLAP) operations *slice*, *dice*, *pivot*, *roll-up*, and *drill-down* involve both conventional and spatial data. For instance, the *slice* involving a spatial attribute may refer to a spatial exact match query, while the *dice* might refer to a spatial range query. Furthermore, in *roll-up* and *drill-down*, numeric measures are summarized using aggregation functions such as SUM, AVG, MIN, MAX and COUNT, while spatial measures are aggregated by spatial aggregation functions, e.g. union, intersection and difference. Moreover, a *roll-up* in a spatial dimension to a non-spatial dimension traverses a hierarchy from a level with a geometric attribute to a level with a non-geometric attribute.

For example, let a cube describe the SDW of a retail application with the dimensions Customer, Part and Date, the fact Orders and the numeric measure Revenue aggregated using the function SUM. The dimension Customer holds the hierarchy NationGeo \preceq CityGeo \preceq CustomerKey, the dimension Part has the hierarchy Brand \preceq PartKey and the dimension Date has the hierarchy Month \preceq Date \preceq DateKey. CityGeo is a spatial attribute of type Region, while NationGeo is a spatial attribute of type Region Set since the territory of nations can have islands.

Considering the aforementioned cube and that q_1 and q_2 are spatial query windows provided by the user, Table 2.1 exemplifies multidimensional queries extended with spatial predicates. Query Q_1 has an intersection range query involving cities of customers and q_1 . Query Q_2 traverses the hierarchy from a finer level to a coarser level and has an intersection range query concerning nations of customers and q_2 , which is larger than q_1 and proportional to the extent of nations. Then, query Q_2 is a roll-up of query Q_1 . Conversely, query Q_1 is a drill-down of query Q_2 . Both queries Q_1 and Q_2 hold slice with a fixed value of brand and a dice involving a set of spatial objects.

2.1.2 Logical Design

The logical database design is a mapping from a conceptual database schema into a schema for the data model underlying a particular database management system (DBMS) (ELMASRI;

Table 2.1: Multidimensional queries extended with spatial predicates.

Query	Description
Q ₁	Retrieve the revenue earned by year with orders of parts whose brand is MFGR2239 for customers located in cities that intersect q_1
Q ₂	Retrieve the revenue earned by year with orders of parts whose brand is MFGR2239 for customers located in nations that intersect q_2

NAVATHE, 2010). The main goals of the logical database design are (BORGIDA; CASANOVA; LAENDER, 2009; ELMASRI; NAVATHE, 2010): (i) to preserve the ability to represent all valid states of the conceptual database schema; (ii) to address issues related to the ease and cost of querying the logical database schema; (iii) to estimate storage costs of the logical database schema; and (iv) to estimate costs related to the maintenance of constraints.

The ER model and its extensions are mapped to logical schemata for relational databases (TEOREY; YANG; FRY, 1986; BORGIDA; MYLOPOULOS, 2009; ELMASRI; NAVATHE, 2010). Relational OLAP (ROLAP) is a logical design that consists of storing the DW in relational databases, extending the SQL and providing specific access methods to efficiently implement the multidimensional data schema and operations. The SDW has been often implemented on a relational DBMS and accessed by SOLAP tools (CAZZIN et al., 2012; BIANCHI; HATANO; SIQUEIRA, 2013; BOULIL; BIMONTE; PINET, 2015). Relational databases are outlined in Section 2.1.2.1. The support provided by DBMS for spatial data is tackled in Section 2.1.2.2. The relational representation of spatial data warehouses is addressed in Section 2.1.2.3.

2.1.2.1 Relational Databases

The Relational Model describes data as named relations of labeled values (CODD, 1970; EMBLEY, 2009a). Each relation has a name and tuples as pairs (*label, value*). Commonly, relations are viewed as tables where labels are the columns and tuples are rows containing values for each column. The relational database of a given application is a collection of table schemata. A relation's schema has mainly a name, a set of attributes with distinct names to compose tuples, key constraints and referential constraints. The key constraint is a set of attributes whose values uniquely determine at most one tuple. When the key has the minimum number of attributes, it is called primary key. The referential constraint prevents referencing an inexistent tuple from another table and is called foreign key.

The Structured Query Language (SQL) is a standard commercial language adopted by DBMSs for specifying, modifying and querying a database, such that developers can declare tables and their constraints (ELMASRI; NAVATHE, 2010). Several DBMSs additionally support

the object-relational model and then allow preserving foundations of the relational model while data is organized using an object model. For instance, an attribute can be implemented as an user-defined type (UDT) with encapsulation and customized functions and operators. Besides, several DBMSs additionally offer non-relational resources. For instance, while the relational model allows only monovalued attributes, current DBMSs enable multivalued attributes to be implemented as columns of type array in tables (POSTGRESQL..., 2015).

2.1.2.2 Spatial Extensions of Database Management Systems

Several DBMSs have spatial extensions that provide spatial data types, spatial operators and spatial functions as well as enable indices for spatial objects. For example, PostgreSQL and Oracle 12c are object-relational DBMSs whose spatial extension are PostGIS (OBE; HSU, 2015) and Oracle Spatial and Graph (ORACLE..., 2014), respectively. Although PostGIS and Oracle Spatial and Graph are exemplified in this section, other DBMSs and their spatial extensions have similar characteristics (e.g. Microsoft SQL Server 2014³ and IBM DB2 Spatial Extender⁴). Both PostGIS and Oracle Spatial and Graph implement vector types defined by the Open Geospatial Consortium (OGC) and SQL/MM types and operators described by the International Organization for Standardization in the standard ISO/IEC 13249-3⁵. Besides, both PostGIS and Oracle Spatial and Graph provide implementations for raster. The aforementioned advanced features benefit the development of spatial applications in many domains.

The vector types implemented in PostGIS and Oracle Spatial and Graph are those defined by OGC. As a result, a column of a table can assume any of these types. Geometry is an abstract and root class of the hierarchy. Thus, geometries are casted to one of its subclasses. Every geometry is associated to a spatial reference system (SRS), which assigns coordinates in a mathematical space to a location in real-world space. There are several SRSs and each one is identified by a spatial reference system identifier (SRID). The spatial abstract data types mentioned in Section 2.1.1 can be directly mapped to vector types supported by the spatial extensions of DBMSs, as follows.

The type Point is mapped to the class Point whose instances are geometries with 0D. The type Line is mapped to the class Line whose instances are geometries with 1D. The type Region is mapped to the class Polygon whose instances are polygonal (and not polyhedral) geometries with 2D. Furthermore, the collection classes MultiPoint, MultiLineString and MultiPolygon refer to collections of points, linestrings and polygons, respectively. The class GeometryCollection

³<https://msdn.microsoft.com/en-us/library/bb933988.aspx>

⁴<http://www.ibm.com/software/products/en/db2spaext>

⁵http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=53698

represents instances with non-homogeneous collections, e.g. a collection composed of two lines and one polygon. An instance of the class TIN is a polyhedral surface composed of one or more triangles that do not overlap, such that each triangle is a polygon.

All subclasses of Geometry allow 0D, 1D and 2D geometries that exist in 2, 3 or 4-dimensional coordinate space, i.e. \mathbb{R}^2 , \mathbb{R}^3 or \mathbb{R}^4 . A geometry in \mathbb{R}^2 has vertices with coordinate values for x and y . A geometry in \mathbb{R}^3 has vertices with coordinate values x , y and z or for x , y and m . The z coordinate often denotes altitude or elevation. The m coordinate represents a measurement that can belong to a linear reference system and refer to events that occur along a network. Analogously, a geometry in \mathbb{R}^4 has points with coordinate values for x , y , z and m .

Both PostGIS and Oracle Spatial and Graph enable adding a geometry column into a table and the registration of geometry columns in OGC's metadata. The registration ensures automatic consistency checks (e.g. whenever a new geometry is inserted) and compatibility with other higher-level application programming interfaces for development of client applications (e.g. MapServer⁶). For example, in PostGIS, the function `AddGeometryColumn` adds a geometry column to a table and registers it in a metadata table (OBE; HSU, 2015).

PostGIS and Oracle Spatial and Graph not only provide geometry data types to represent spatial data types, but also implement OGC functions for the evaluation of topological predicates and for the aggregation of spatial data. In detail, these DBMS implement an extended version of the 9-intersection model (EGENHOFER; FRANZOSA, 1991) and manipulate 3×3 matrices expressed as 1×9 masks. For instance, the topological relationship *intersects* has a corresponding OGC function `Intersects` that evaluates the intersection between two spatial objects, which is implemented in PostGIS as the function `ST_Intersects` that returns whether two geometries provided as arguments intersect. Also, the union of two spatial objects is described by the OGC aggregate function `Union` (HERRING, 2011), which is implemented in PostGIS by the function `ST_Union` that receives a set of geometries as arguments and unions (merges) them into a single geometry with no intersecting regions (OBE; HSU, 2015).

2.1.2.3 Relational Representation of Spatial Data Warehouses

Three main logical models for DW exist: relational OLAP (ROLAP), multidimensional OLAP (MOLAP) and hybrid OLAP (HOLAP) (VAISMAN; ZIMÁNYI, 2014a). They differ on how to store the data cube. ROLAP stores data in relational databases, extends SQL and provides specific access methods to efficiently implement the multidimensional data model and the corresponding operations. Conversely, MOLAP stores data in specialized multidimensional

⁶<http://mapserver.org>

data structures such as arrays and implements the OLAP operations over those data structures. Finally, HOLAP combines both approaches. Since SDWs have been often deployed on a relational DBMS and accessed by SOLAP tools, this section focuses on the relational representation of SDWs.

Considering that existing DBMSs have extensions with support for spatial data types and implements functions to evaluate topological relationships, logical schemata of SDWs are designed reusing the relational model. The well-known star and snowflake schemata (KIMBALL; ROSS, 2002) are adapted to support the inclusion of spatial attributes as geometry columns in dimension tables and as measures in fact tables.

Stefanovic, Han & Koperski (2000) reused and adapted the star-schema and the snowflake schema proposed by Kimball & Ross (2002), but did not provide clear rules for the creation of the relational schema of the SDW. Three approaches were proposed for storing a spatial measure in a fact table: (i) pointers for spatial objects; (ii) rough approximations of spatial objects; and (iii) duly selected spatial objects. Stefanovic, Han & Koperski (2000) then developed a selective materialization of spatial objects based on the relative access frequency.

Spatial data introduces new storage costs, might impair the performance to process queries and often require the design of a hybrid schema (FIDALGO et al., 2004) because a geometry requires a varying storage space according to its shape (CIFERRI, 2002). In addition, the spatial data redundancy impairs the query processing performance in SDW and must be avoided (SIQUEIRA et al., 2008, 2009). The performance of some SOLAP queries can also be improved by a logical design that (MATEUS et al., 2010): (i) maintains a spatial attribute of type Point and other conventional attributes in the same table if, and only if these attributes have a 1:1 relationship; and (ii) maintains a surrogate key and a spatial attribute whose dimensionality is 1D or 2D in a table separate from the other conventional attributes.

For instance, consider an extended version of the retail SDW exemplified in Section 2.1.1, whose logical schema is shown in Figure 2.1 according to the relational model. LineOrder is a fact table, Date and Part are conventional dimension tables, Supplier and Customer are spatial dimension tables and City, Nation and Region are spatial level tables. The fact table LineOrder has a composite key to identify items of an order, columns with suffix “FK” with a foreign key referencing the dimension tables, and the numeric measures Revenue and Quantity. Spatial attributes have both a suffix “Geo” and an assigned geometry type. Conventional types are omitted for the sake of simplicity.

In the schema shown in Figure 2.1, dimension tables Supplier and Customer have each one a geometry column to store addresses, since the relationship between the spatial attribute of type

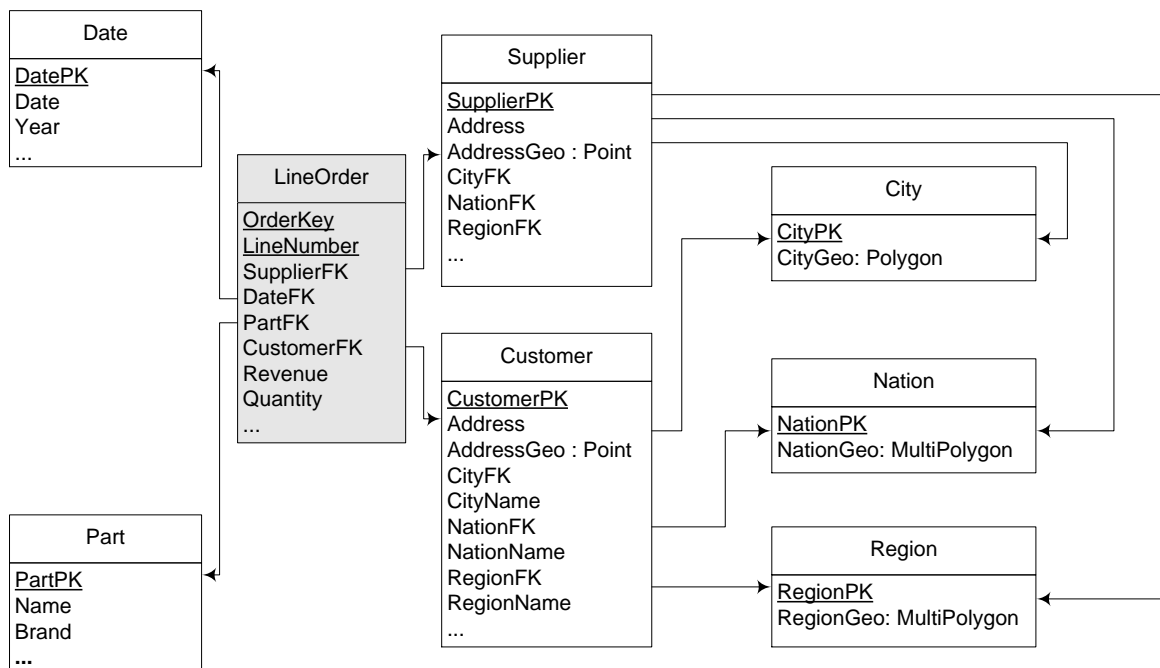


Figure 2.1: The relational schema of a SDW regarding a retail application (adapted from (SIQUEIRA et al., 2010)).

Point and the other attributes is 1:1. On the other hand, surrogate keys and geometry columns for cities, nations and regions are stored in separate tables, since spatial data redundancy must be avoided. Then, polygons of cities are not redundantly stored in the tables Supplier and Customer. Furthermore, City, Nation and Region are referenced by Supplier and Customer through foreign keys to prevent joining unused tables to process queries. Although the hierarchy $\text{RegionGeo} \preceq \text{NationGeo} \preceq \text{CityGeo}$ holds, City does not reference Nation and the Nation does not reference Region because such design would impose a snowflake schema (KIMBALL; ROSS, 2002) that introduces joins and impairs the performance to process queries.

The query shown in Listing 2.1 issued over the SDW schema shown in Figure 2.1 retrieves the total revenue earned by year by brand with customers located in cities that intersect the rectangular spatial query window whose vertices are (50.82, 3.99), (50.82, 4.93), (51.32, 4.93) and (51.32, 3.99). The PostGIS function `ST_GeomFromText` receives a string as input, formatted according to well-known text OGC standard, and converts into a geometry (OBE; HSU, 2015). The query requires five joins among tables, one conventional predicate to filter brands, one spatial predicate as the intersection range query involving cities, grouping and sorting. These operations may impair the performance and should be tackled in the physical design of the SDW.

Listing 2.1: A SQL query issued over the SDW described in Figure 2.1.

```

SELECT SUM(Revenue), Year, Brand
FROM LineOrder, Date, Part, Customer, City
WHERE DateFK = DatePK
      AND PartFK = PartPK
      AND CustomerFK = CustomerPK
      AND CityFK = CityPK
      AND Brand = 'MFGR2239'
      AND ST_Intersects (CityGeo, ST_GeomFromText('POLYGON((
50.82 3.99, 50.82 4.93, 51.32 4.93, 51.32 3.99, 50.82 3.99))'))
GROUP BY Year, Brand
ORDER BY Year, Brand;

```

2.1.3 Physical Design

The physical design of the SDW is crucial to determine reasonable performance and short query response time (VAISMAN; ZIMÁNYI, 2014a). Three techniques have been used to improve the performance to process queries in SDWs: indexing (PAPADIAS et al., 2001; MOHAN et al., 2008; SIQUEIRA et al., 2012b), view materialization (STEFANOVIC; HAN; KOPERSKI, 2000; RAO et al., 2003; GOMEZ et al., 2009) and partitioning (BALTZER; RAU-CHAPLIN; ZEH, 2013; MATEUS et al., 2015).

Since this thesis focuses on indexing, the following sections summarize indices that improve the performance to process queries that involve joins, conventional predicates, aggregation and spatial predicates, as these operations are usually present in SOLAP queries. Section 2.1.3.1 addresses the bitmap index and the bitmap join index. The former is useful for processing conventional predicates and conventional data aggregation, while the latter avoids costly joins among tables. Finally, Section 2.1.3.1 tackles spatial indices that aid the resolution of spatial predicates and the aggregation of spatial data.

2.1.3.1 Bitmap Index and Bitmap Join Index

The first implementation of the bitmap index was developed for the non-relational database system called Model 204 (O'NEIL, 1989). Operations such as COUNT, AND, and OR were efficiently processed by Model 204. The benefits of the bitmap index have motivated its extension as a join index, i.e. the bitmap join index (O'NEIL; GRAEFE, 1995). The bitmap index and the bitmap join index are described in the following, but considering the relational database.

Let T be a table, A be a column in T and $|A|$ be the cardinality of A , i.e. the number of distinct values that A can assume. The bitmap index built on A indicates, for each value a that A can assume, the rows of T where $A = a$, as follows. A bit-vector is a vector of bits whose entries can assume the value 1 to represent *true* (or *on*) or the value 0 to denote *false* (or *off*).

The bitmap index built on A comprises one bit-vector for each distinct value a that A can assume. Then, the cardinality of the indexed column determines the quantity of bit-vectors. The length of every bit-vector is equal to the number of rows stored by T . If $A = a$ in the i -th row of the T , then the i -th bit of the bit-vector created for the value a is set to 1. Otherwise, the bit is set to 0.

Let T_L and T_F be tables and L be a column in T_L . A bitmap join index built on L indicates the set of rows in T_F that can be joined with a certain value of T_L , as follows. The bitmap join index built on L comprises one bit-vector for each distinct value l that L can assume. The length of every bit-vector is equal to the number of rows of the join between T_L and T_F . If $L = l$ in the i -th row of the join between T_L and T_F , then the i -th bit of the bit-vector built for value l is set to 1. Otherwise, the bit is set to 0. Note that the join between T_L and T_F requires a single join if T_F references T_L . Otherwise, several joins might be required. Joins among huge DW tables are necessary only once to build the bitmap join index. After the index is built, the queries can be processed by accessing the index and avoiding joins.

Figure 2.2 shows the table that corresponds to the selection of the columns Revenue, Year, Brand and CityFK after joining the tables LineOrder, Date, Part, and Customer from the SDW schema shown in Figure 2.1. For the sake of simplicity only the first five rows are displayed. Figure 2.2 also illustrates bitmap join indices built on the columns Revenue, Year, and Brand of the aforementioned table. Each bit-vector indicates the rows of the fact table where a given value occurs. For instance, Year = 2013 occurs in the first and second rows, as the bit-vector for Year = 2013 has the bit 1 in both the first and second entries.

Furthermore, conventional predicates involving logical operations are solved using bit-wise operations. For instance, the conjunction Year = 2013 AND Brand = 'MFGR#2239' executes the bit-wise AND using the corresponding bit-vectors, i.e. 11000 AND 10010 whose result is 10000. Therefore, the first row of the fact table satisfies Year = 2013 AND Brand = 'MFGR#2239'. The same bit-vector 10000 can be used to group results by year by brand and apply the function SUM to aggregate values of Revenue, i.e. only the first row of the table is processed by SUM. The aforementioned bitmap join indices and the operations described using bit-vectors aid to process the query shown in Listing 2.1, except the spatial predicate.

The bitmap index and the bitmap join index provide efficient processing of bit-wise logical operations even if the number of involved bit-vectors is high (STOCKINGER; WU, 2006; WU et al., 2009). Therefore, they have been used to index columns of DWs (VAISMAN, 1998; LANE; POTINENI, 2014) in addition to B-tree and hashed indices provided by the DBMS. A column with high cardinality requires an index with many bit-vectors which are sparse, i.e. that have few bits with value 1. In the past, the use of the bitmap index was avoided if the cardinal-

Bitmap join indices

$\Pi_{Revenue, Year, Brand, CityFK}(\text{Lineorder} \bowtie \text{Date} \bowtie \text{Part} \bowtie \text{Customer})$				Revenue					Year			Brand	
Revenue	Year	Brand	CityFK	3,589	5,410	6,901	7,931	10,012	2012	2013	2014	MFGR# 2221	MFGR# 2239
7,931	2013	MFGR#2221	12	0	0	0	1	0	0	1	0	1	0
6,901	2013	MFGR#2339	11	0	0	1	0	0	0	1	0	0	1
10,012	2012	MFGR#2339	13	0	0	0	0	1	1	0	0	0	1
3,589	2014	MFGR#2221	13	1	0	0	0	0	0	0	1	1	0
5,410	2014	MFGR#2339	11	0	1	0	0	0	0	0	1	0	1
...

Figure 2.2: A table and its columns indexed by bitmap join indices.

ity of the column was high (O'NEIL, 1989). In current relational DBMSs, bitmap join indices should be built on columns with low cardinality (LANE; POTINENI, 2014). Furthermore, three techniques have been continuously improved to overcome limitations imposed by the high cardinality: compression, binning and encoding (WU; OTOO; SHOSHANI, 2006; WU et al., 2009). As a result, recent implementations of the bitmap index provide efficient query processing even for attributes with huge cardinality.

The bitmap join index is reused by the SB-index that is an efficient index for SDW, as detailed in Section 3.3. Besides, the bitmap join index is also reused by the VSB-index, which is one of the most relevant contributions of this thesis, as described in Section 6.3.

2.1.3.2 Spatial Indices

An approximation is a coarser representation of a spatial object and is, therefore, less complex than the original spatial object. A conservative approximation is a superset of the extent of the spatial object, while a progressive approximation is a subset of the extent of the spatial object (BRINKHOFF; KRIEGEL; SCHNEIDER, 1993). For instance, the minimum bounding rectangle (MBR) and the convex-hull are conservative approximations, while the maximum enclosed rectangle (MER) is a progressive approximation. The MBR is the smaller iso-oriented rectangle that circumscribes a given geometry. The convex-hull is the smallest convex set of points that contains a geometry. The MER is a rectangle that intersects the longest enclosed horizontal connection starting in a vertex of the polygon. In addition, the vertices of the MER are also vertices of the polygon. Figure 2.3a depicts a region and its MBR, Figure 2.3b depicts a region and its MER and Figure 2.3c depicts a region and its MBR and MER. The extent that belongs to a conservative approximation but does not belong to the spatial object is called deadspace. Note that the spatial query window w intersects the deadspace of the MBR but does not intersect the spatial object in Figure 2.3d.

The multi-step resolution of the spatial predicate (BRINKHOFF; KRIEGEL; SCHNEIDER, 1993) is illustrated in Figure 2.4. In the filter step, the spatial query window is tested against the approximations built for the spatial objects of the dataset. A spatial object is either an *answer* if the test concludes that it satisfies the spatial predicate, or a *candidate* if the test assumes it might satisfy the spatial predicate. Otherwise, the spatial object is ignored. The conclusion of the filter step depends on the approximations being used and on the spatial predicate being tested.

In the refinement step, the original spatial objects of the candidates are accessed and tested against the spatial query window. Candidates that are not answers are ignored as they are merely false hits, while spatial objects that satisfy the spatial predicate are answers. The refinement step is in general more costly than the filter step, since the former requires the access to the original spatial objects stored in secondary memory, while the latter accesses the approximations. To reduce the cost of the refinement step, the filter step can use progressive approximations in addition to conservative approximations to identify answers of the spatial predicate (GAEDE; GUNTHER, 1998).

For instance, Figure 2.3d shows a region, its MBR, its MER and also two spatial query windows w and v that aim to evaluate the spatial predicate IRQ. Concerning the spatial query windows v , the filter step verifies that the intersection between v and the MBR is true while the intersection between v and the MER is true. As a result, the filter step considers the region as being an answer of the IRQ, since v intersects a subset of the region, i.e. its MER. Conversely, regarding the spatial query windows w , the filter step verifies that the intersection between w and the MBR is true while the intersection between w and the MER is false. Then, the filter step considers the region as being a candidate, since w intersects a superset of the region, i.e. its MBR. In the refinement step, the polygon of the region is fetched and compared to w . The intersection between w and the region is false. Therefore, the region does not belong to the set of answers of the IRQ.

Spatial indices are efficient data structures to improve the resolution of spatial predicates. They store approximations and implement the multi-step resolution described in Figure 2.4. In the following, spatial indices are summarized because they are implemented in existing spatial extensions for DBMSs. While Oracle Spatial and Graph provides an implementation of the R-tree, PostGIS offers an implementation of the Generalized Search Tree (GiST).

The R-tree (GUTTMAN, 1984) is a spatial index that supports spatial range queries and is commonly implemented by DBMSs to index spatial objects data in the secondary memory. Its hierarchical data structure is based on the B-tree (BAYER; MCCREIGHT, 1972) and has non-leaf

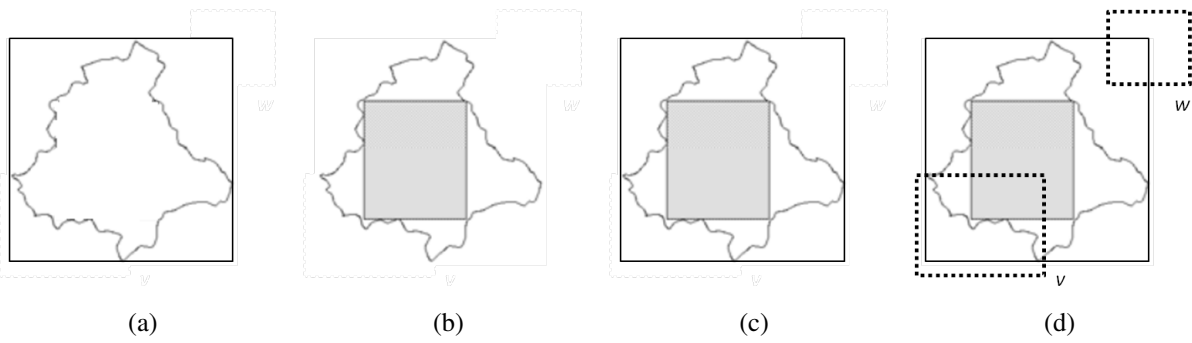


Figure 2.3: A region, its approximations and spatial range queries. (a) A region and its MBR. (b) A region and its MER. (c) A region and its MBR and MER. (d) Spatial query windows w and v .

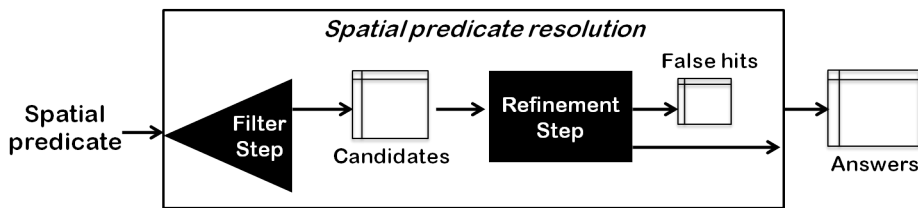


Figure 2.4: The multi-step resolution of spatial predicates (adapted from Brinkhoff, Kriegel & Schneider (1993)).

nodes and leaf nodes. A leaf node has entries holding the key value and the MBR of a spatial object, also called input MBR. A non-leaf node has entries holding a pointer to a child node and a MBR that encompasses all the MBRs of the corresponding child node. Every node can store M entries and must store at least m entries, such that often $m \leq M \div 2$. Only the root node can have less than m entries. The capacity of nodes is determined by the disk page size.

The insertion algorithm allocates input MBRs in entries of leaf nodes. Initially, the root node is also the single leaf node and entries are inserted until the node becomes full. The fulfilled root node causes the creation of two leaf nodes and the distribution of the entries between them. Two distinct subsets of entries are created, such that each subset has its MBRs encompassed by one computed MBR whose coverage of the extent is minimal. Each subset is then allocated in one leaf node. The two MBRs with minimal coverage are allocated in distinct entries of the new root node. Each entry of the root node maintains a pointer to one leaf node. As a result, the computed MBRs refer to two distinct clusters of input MBRs.

After the division of the root node, each new input MBR is inserted as follows. The R-tree is traversed comparing the input MBR to MBRs held by non-leaf nodes. The subtree chosen to store the input MBR is the one pointed to by the entry whose MBR demands the minimal increase on area. When more than one subtree is able to store the input MBR, the preference is for that subtree pointed to by the entry whose MBR has the smallest area. If the leaf node is already fulfilled, then the aforementioned procedure of distribution is applied. Finally, the

modifications due to distribution are back-propagated to higher levels of the R-tree.

The hierarchical structure allows pruning the traversal to process the filter step. A top-down traversal of the tree starts at the root node and tests the intersection between the spatial query window and MBRs in entries of non-leaf nodes. If the intersection is true, the traversal continues through child nodes. Leaf nodes are also visited and their MBRs that satisfy the spatial predicate become candidates and have the corresponding key values collected. These candidates are processed in the refinement step that accesses the original geometries. It is noteworthy that the creation of MBRs with the minimal coverage for the entries of non-leaf nodes reduce the possibility of intersection among MBRs of distinct non-leaf nodes and a spatial query window. When a spatial query window intersects MBRs of distinct non-leaf nodes, there is a ramification of the tree traversal during the search, which might impair the performance.

Consider the SDW whose schema is shown in Figure 2.1. Figure 2.5a illustrates five cities of customers as polygons, whose MBRs are r_{11} , r_{12} , r_{13} , r_{21} and r_{22} . These MBRs were the input to build a R-tree whose nodes can maintain at most 3 entries, i.e. $M = 3$. The insertion algorithm computed the MBRs R_1 and R_2 and distributed of the input MBRs between R_1 and R_2 . The MBR R_1 encompasses r_{11} , r_{12} and r_{13} , while the MBR R_2 encompasses r_{21} and r_{22} . The data structure of the obtained R-tree is depicted in Figure 2.5b.

Like the query described in Listing 2.1, an intersection range query is issued using the spatial query window w depicted in Figure 2.5a. To resolve such spatial predicate, the entries of the R-tree that intersect w and are highlighted in gray in Figure 2.5b are accessed, since w intersects R_1 , r_{11} and r_{13} . As a result, the cities represented by the MBRs r_{11} and r_{13} are considered candidates. A subsequent refinement step verifies that the city represented by the MBR r_{11} is an answer, but the city denoted by the MBR r_{13} is just a false hit. The resolution described for the intersection range query complies with the spatial predicate of the query shown in Listing 2.1.

The R*-tree (BECKMANN et al., 1990) improves the R-tree insertion algorithm according to the criteria of coverage, overlap, margin and storage, rather than only applying the criterion of reducing the coverages. While the overlap criterion aims at minimizing the intersection area of the MBRs, the margin criterion is used to minimize the perimeter of the MBRs, and the storage criterion is used to maximize the occupation rate of the structure nodes. Analyzing these criteria during the insertion guarantees to the R*-tree a better space partitioning and, consequently, a better search performance than the R-tree.

The GiST (AOKI, 1998) is aimed at supporting an extensible set of queries and data types that can unify and generalize the behavior of different search trees. Due to this flexibility, some

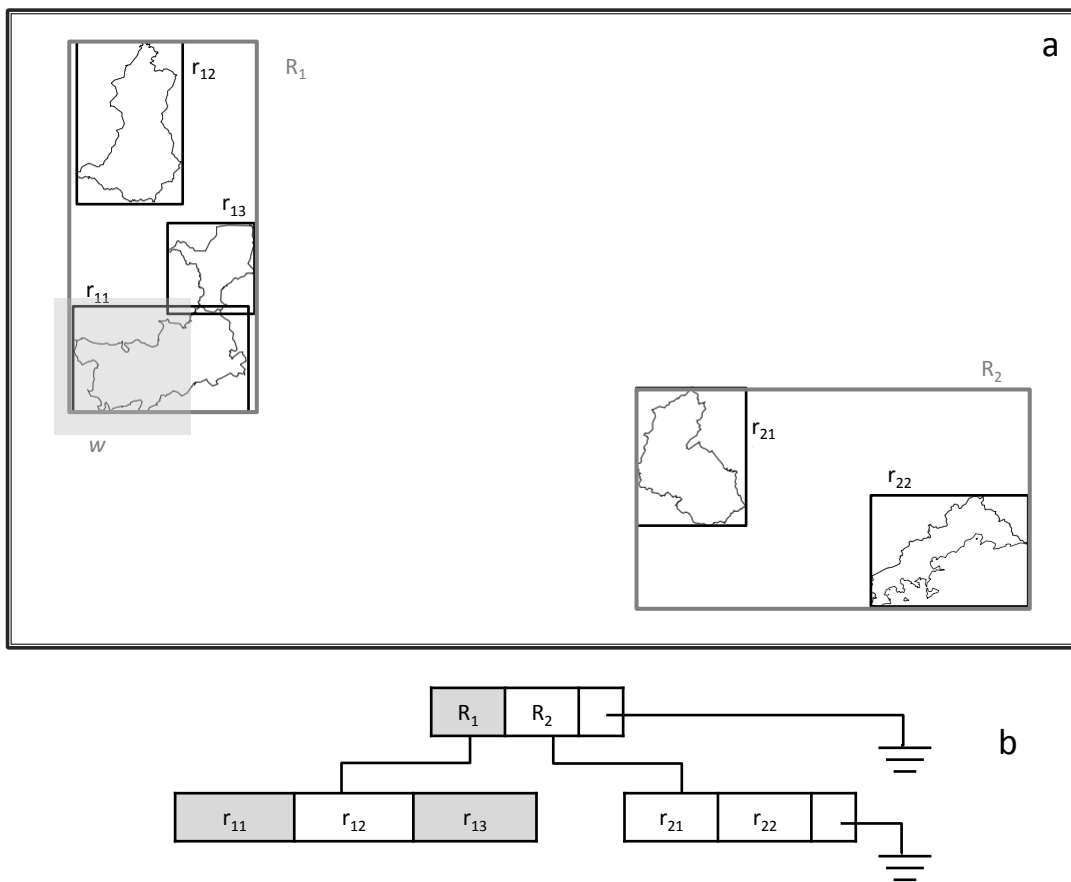


Figure 2.5: An R-tree built using cities of customers from the SDW described in Figure 2.1. (a) Cities, their MBRs and the R-tree's space partitioning method. (b) The R-tree data structure.

DBMSs have implemented the GiST to efficiently index and retrieve conventional and complex data for different queries. If the GiST is built on a spatial attribute, it inherits the characteristics of the R-tree.

The performance to resolve a spatial predicate using a spatial index can be impaired by several factors, as follows (CIFERRI, 2002). The factors related to spatial data include but are not limited to data type, complexity (number of vertices), distribution of objects in space and volume of data. The object's extent and shape and the degree of overlapping among objects may also influence, except if the type is point. The factors associated to the workload include but are not limited to the spatial predicate, the spatial query window's size, shape and spatial distribution and selectivity. Given an attribute with cardinality c and a spatial query window w that retrieves n distinct values in the domain of the attribute, the selectivity is $n \div c$.

Spatial indices such as R-tree and GiST are implemented by the DBMS to improve the performance to resolve spatial predicates, which are often present in SOLAP queries issued over SDWs. For instance, PostGIS implements the GiST (OBE; HSU, 2015), while Oracle Spatial and

Graph provides the R-tree (ORACLE..., 2014). The R-tree is also extended by existing indices for SDW such as the aR-tree tackled in Section 3.3 as well as by existing indices for vague regions such as the Vague R-tree and the FMBR R-tree addressed in Section 3.4. Finally, the multi-step resolution of the spatial predicate using conservative and progressive approximations is also adopted by the VSB-index, which is one of the most relevant contributions of this thesis, as described in Section 6.3.

2.2 Uncertain Data Management

Information from the real world is often imperfect in several ways. A database is an abstraction of a piece of the real world and is intrinsically imperfect, since its relevant entities and its level of detail are determined according to the needs of the database applications (ZIMÁNYI; PIROTTE, 1997). On the one hand, the information contained in a database is *complete* and *certain* if it accurately and adequately represents the corresponding application domain in the real world. On the other hand, a piece of information in a database is *imprecise* if it approximates the corresponding piece of information in the real world. Also, a piece of information in a database is *uncertain* if the corresponding piece of information in the real world is imperfectly known. Uncertainty and imprecision in databases require representing and querying information that is uncertain, vague, fuzzy, probabilistic, unknown, partially known, indefinite, disjunctive, possible, maybe, incomplete, approximate, erroneous, or imprecise (DYRESON, 1997).

Uncertainty in spatial data is the gap between a phenomenon in the real world and its spatial representation in a database (GOODCHILD, 2008). Such gap appears, for example, if broad boundaries of a phenomenon are modeled as sharp boundaries (BEJAOU, 2009). The terms “imperfection” and “uncertainty” are often used interchangeably, as *spatial uncertainty* encompasses different types of *spatial data imperfections* (BÉDARD, 1987). Imperfections are inherent to spatial data and directly influence the reliability of spatial analysis output (GOODCHILD, 2008). There is not a commonly accepted terminology to define concepts related to imperfections of spatial data. Consequently, the terms and categorizations associated to imperfections assume different interpretations depending on the people using the terms and their scientific community (DEVILLERS et al., 2010). Even though, in different categorizations of spatial data imperfections, spatial uncertainty is considered the root and is a generic imperfection that can be specialized into different forms (WORBOYS, 1998; FISHER, 1999; JADIDI et al., 2014).

This thesis addresses uncertain data. Hence, two prominent representations for uncertain data are summarized in the following. Section 2.2.1 tackles probabilistic data, while Sec-

tion 2.2.2 stands for fuzzy data. Probability and fuzziness on spatial data and in DWs are also outlined.

2.2.1 Probabilistic Data

Probabilistic data refer to uncertain data modeled using the mathematical fundamentals of Probability Theory (whose core concepts have been recently addressed by Klenke (2014)). Let X indicate a finite universe and p be a probability distribution. The value given of $p(x)$ specifies the probability that an element $x \in X$ occurs such that $p(x) \in [0,1]$. In addition, $\sum_{x \in X} p(x) = 1$. A discrete probability distribution can also be denoted as being the set $\{x_1/p(x_1), x_2/p(x_2), \dots, x_n/p(x_n)\}$, where $x_i \in X$ for $i = 1, \dots, n$ such that $\sum_{x \in X} p(x) \leq 1$.

An example with conventional data is “John teaches Physics with probability 0.8” (ZIMÁNYI; PIROTTE, 1997). The statement is represented, in the relation *Teaches* (Student, Course, Probability), by the tuple (‘John’, ‘Physics’, 0.8). It is also possible to claim that “John teaches no course with probability 0.2”. Nevertheless, such statement would not require a tuple in the relation *teaches*.

As for spatial data, an *error* is the difference between the available value and another one considered as true (GOODCHILD, 1995). For example, a measurement device that is misused or inadequately calibrated provides erroneous measurements that are inserted into the spatial database and afterwards are considered as true values (BEJAOU, 2009). The inability of measuring an object precisely is a measurement uncertainty (PAULY; SCHNEIDER, 2010). The lack of knowledge about the position and shape of a spatial object that has a definable boundary is a positional uncertainty (PAULY; SCHNEIDER, 2010). The uncertainty of a crisp spatial object is caused by errors and is intrinsically probabilistic (FISHER, 1999).

There are three categories of probability models that refer to different granularities of uncertainty, as follows (TAO; XIAO; CHENG, 2007). A table-based solution estimates how much percentage of tuples are present in a table (WIDOM, 2005). A tuple-based solution assigns a probability to each individual tuple to indicate the likelihood that the tuple exists in the table (DALVI; SUCIU, 2004; SUCIU, 2009). Finally, an attribute-based solution is required when attribute of a tuple is not known precisely and introduces a probability distribution for describing a set of possible values, together with their probabilities of occurrence (WOLFSON et al., 1999; CHENG; KALASHNIKOV; PRABHAKAR, 2003).

Section 2.2.1.1 addresses probabilistic models for DWs and Section 2.2.1.2 tackles probabilistic models for spatial data.

2.2.1.1 Probabilistic Data Warehouses

This section describes the combination of tuple-based and attribute-based solutions that enabled imprecision and uncertainty in multidimensional modeling (BURDICK et al., 2005, 2007).

To illustrate how imprecision is supported, consider the fact table *LineOrder* shown in Figure 2.1. Rather than referencing a key value from *Customer*, an imprecise fact would reference a key value from *City*. As a result, the fact would be associated to a given city, but still unsure about the customer (and its address). Nevertheless, a fact member cannot reference a member of a level that is a non-leaf level of a hierarchy.

Therefore, the column *Allocation* is necessary and is added to the fact table to denote the probability of a fact member being associated to a given customer, based on the referenced city. Such probability is called weight. If the referenced city has x customers, then x weights are created and their sum is equal to 1. Then, the imprecise fact comprises x rows in the fact table.

Uncertainty in measure values was provided by applying a probability distribution function to indicate the degree of belief that a true value is being represented. For example, consider the existence of the uncertain measure *Customer Satisfied?* in the fact table *LineOrder* shown in Figure 2.1. Its uncertain domain is {Yes, No} and a discrete probability distribution function indicates, with a pair of probabilities, whether the customer was satisfied with a given part referenced by the fact table, e.g. {Yes/0.8, No/0.2}.

The function results from a classifier algorithm that analyses values of other attributes of the schema and outputs the pair of probabilities. The classifier is applicable to domains with more than two elements, but is beyond the scope of this section. The aggregation of an uncertain measure performs a weighted linear combination of probability distribution functions to reach a consensus from a set of opinions. For example, the query “how likely were customers satisfied with a given part brand in 2015?” requires aggregating the uncertain measure *Customer Satisfied?*. The aggregation functions SUM, AVG, and COUNT are extended for uncertain and imprecise data and their completeness and faithfulness characteristics ensure the summarizability.

2.2.1.2 Probabilistic Spatial Data

A spatial probability distribution can model the random position of a spatial object. Then, attribute-based probability models have been often used to model uncertainty in spatio-temporal databases (TAO; XIAO; CHENG, 2007). The probabilistic models mainly deal with the expectation of a future event, based on something currently known (PAULY; SCHNEIDER, 2010).

Suppose a location-based service used to monitor moving objects such as vehicles. Each vehicle o informs its current location whenever it has moved away from its previously updated position x by a certain distance ε . At any time, the precise location of o is unknown, but must be inside a circle whose center is x and radius is ε and constrained by the underlying road network (WOLFSON et al., 1999). Figure 2.6 illustrates the position of the vehicle o , ε , the circle, and the road network. It also depicts a grid that approximates the distribution of o , such that o can be strictly in the gray cells that intersect both the circle and the road network. Imposing an equal chance for all the gray cells would be a simple solution. A more realistic solution would be to assign chances taking into account the distance between the cell and x and the speed limits of roads, for example. A probability distribution can also assign a weight to each pixel of a raster that displays the extent of an object (PEBESMA; KARSSENBERG; JONG, 2006).

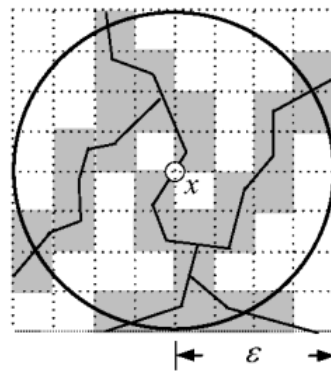


Figure 2.6: Probabilistic modeling of uncertain objects: the location of a vehicle (extracted from Tao, Xiao & Cheng (2007)).

Spatial range queries are processed using probabilistic constrained rectangles (PCRs) that approximate object's probabilities acting as MBRs (TAO; XIAO; CHENG, 2007). First, PCRs prune and validate the majority of non-qualifying and qualifying probabilistic data. Second, a refinement thus processes a small number of objects by invoking more costly routines, which comprise loading and calculating an object's probability.

Nearest neighbor queries often do not have an obvious answer, since no object is the nearest neighbor with absolutely certainty. The superseding nearest neighbor core (SNN-core) overcomes such limitations (YUEN et al., 2010). It is the minimum set of nearest neighbor candidates, such that each candidate overcomes (supersedes) all the nearest neighbor candidates outside the SNN-core. Once the SNN-core is usually a singleton, it benefits the nearest neighbor search as it can minimize the number of retrieved objects.

2.2.2 Fuzzy Data

Fuzzy Logic and Fuzzy Set Theory provide means of addressing the uncertainty regarding the absence of sharply defined criteria of membership (ZADEH, 1965). A fuzzy set \tilde{A} defined in \mathbb{R}^n has a function $\mu_{\tilde{A}} : \mathbb{R}^n \rightarrow [0, 1]$ called membership function, which quantifies the membership degree of an element a to the fuzzy set \tilde{A} . Membership values are in the interval $[0, 1]$. Higher membership values represent increasing membership degrees, while lower membership values denote decreasing membership degrees. The membership value 1 ensures a certain membership to the fuzzy set, while the membership value 0 indicates a false membership to the fuzzy set. According to the application, a fuzzy set can be interpreted as degree of similarity (to a prototype in a class), degree of plausibility, or degree of preference (DUBOIS; PRADE, 1997). It is noteworthy that fuzzy sets do not have a statistical nature (ZADEH, 1965) and are not meant to disguise probabilities (DUBOIS; PRADE, 2015).

In database management, two main approaches are adopted (KACPRZYK; ZADROZNY; TRÉ, 2015). One approach enables issuing fuzzy queries against a traditional database such that matching a tuple against a query provides a gradual rather than binary result set. The other approach considers a fuzzy database whose data are affected by fuzziness. Table-based solutions assume that relations in a database are fuzzy. In attribute-based solutions, an imperfectly known value of an attribute at some tuple is represented by a possibility distribution. Fuzzy databases also require modifications on query languages in order to retrieve query answers by similarity instead of equality, for example.

Section 2.2.2.1 addresses fuzzy data warehouses. Spatial fuzziness is closely related to the difficulty of delineating boundaries for regions (HAZARIKA; COHN, 2001) and to provide gradual rather than abrupt membership to fuzzy spatial objects. Conversely, spatial vagueness is more general than spatial fuzziness and can also refer to the broadness of shapes such as point (set), line (set), and region (set). This thesis focuses on spatial vagueness, which is duly addressed in Section 2.3. Besides, related work regarding spatial vagueness in spatial data warehouses are surveyed in Section 3.2.

2.2.2.1 Fuzzy Data Warehouses

Multidimensional modeling and data warehouses have been extended with fuzzy data in several manners. This section outlines a few of them.

A membership degree can be assigned to elements of a data cube considering membership values interpreted as degrees of truth rather than possibility, as follows (LAURENT, 2003). An

attribute value is not a single, but a pair of values (v, c) , where v is either a precise value or a linguistic label obtained from a fuzzy set and d is a confidence degree in $[0,1]$. Then, an attribute of a dimension is (v_D, c_D) and a measure (attribute of a fact) is (v_m, c_m) . A fact member f comprises pairs of values (v_D, c_D) and (v_m, c_m) and is assigned a degree μ_f in $[0,1]$ that denotes the extent to which it belongs to the cube.

For instance, let *Food*, *Season*, and *City* be dimensions and *Sales* be a fact with the measure *Quantity*. An example of fact member is $((\text{'Pamonha'}, 1.0), (\text{'Corn Season'}, 0.7), (\text{'Piracicaba'}, 0.8), (\text{'High'}, 90,000), \mu_f=0.9)$. The OLAP operator dice, for example, selects fact members that satisfy conditions involving (LAURENT, 2003): (i) the attribute value(s) v_d (or v_m), e.g. $\text{Season}=\text{'Corn Season'}$; (ii) the degree μ_f , e.g. $\mu_f=0.9$; and (iii) the degree of confidence c_D (or c_m), e.g. $c_{\text{Season}}=0.7$.

A fuzzy hierarchical relation addresses the imprecision in the relationship between members of different levels (DELGADO et al., 2004, 2007). The value in $[0,1]$ representing the relation between members of a level and members of parent levels can be replaced by linguistic labels. For example, consider the hierarchy $\text{LegalAge} \preceq \text{Age}$ and the fuzzy hierarchy $\text{AgeGroup} \preceq \text{Age}$. The members of *AgeGroup* are 'Young' and 'Adult', and 'Old', the members of *LegalAge* are 'Yes' and 'No', and the members of *Age* are values between 0 and 120. Each age is associated to a legal age with possibility that is either 1 or 0. Conversely, ages are classified within the fuzzy concept of an age group. The age 25 belongs to 'Young' with a membership degree of 0.7 and to 'Adult' with a degree of 0.3. Therefore, a roll-up to the level *AgeGroup* would distribute age 25 as 17.5 (obtained from 0.7×25) for 'Young' and 7.5 (obtained from 0.3×25) for 'Adult'.

As for the logical design, the methodology of Kimball & Ross (2002) is also extended, as follows. A fuzzy dimension has at least one category attribute as a fuzzy concept (SAPIR; SHMILOVICI; ROKACH, 2008). An additional table has one column for each linguistic label defined by the membership function. It also has a column referencing the level on which the fuzzy concept was defined. The tuples store membership values regarding the linguistic labels. Consider the dimension *Customer* containing the hierarchy $\text{AgeGroup} \preceq \text{Age}$ as described in the previous example. The table *Customer* (CustomerPK, Age, ...) has a surrogate key of a customer and its age, respectively, and a tuple is ('Customer1', 25, ...). The table *CustomerAge* (CustomerFK, Young, Adult, Old) has a foreign key and one column for each linguistic label, and a tuple is ('Customer1', 0.7, 0.3, 0.0).

An alternative is to separate fuzzy membership tables from fuzzy classification tables (FASEL, 2014). Their meanings are illustrated as follows considering the previous example. The fuzzy membership table *CustomerAgeFMT* (CustomerAgeFmtPK, CustomerFK, LabelFK, Mem-

bership) holds a surrogate key, a reference to the table *Customer*, a reference to the table *CustomerAgeFCT*, and a membership value, respectively. Two tuples are ('Fmt25', 'Customer1', 'FctYoung', 0.7) and ('Fmt25', 'Customer1', 'FctAdult', 0.3). The fuzzy classification table *CustomerAgeFCT* (CustomerAgeFctPK, Label) holds a surrogate key and a linguistic label, respectively. Two tuples are ('FctYoung', 'Young') and ('FctAdult', 'Adult'). To sum up, a fuzzy data warehouse has at least one or more fuzzy membership tables and zero or more fuzzy classification tables.

2.3 Spatial Vagueness

Figure 2.7 illustrates an UML class diagram representing a taxonomy that contextualizes spatial uncertainty in spatial data modeling and the models used to handle it. The uncertainty affecting an object's definition is related to the object's semantics and shape (FISHER, 1999). The spatial representation of a given phenomenon is either a well-defined object or a poorly-defined object. A well-defined (crisp) object is considered definable in both its conventional attributes and spatial representation. As a result, it has Boolean occurrence, such that any location is either part of the object, or it is not. If the object is well-defined, then uncertainty is addressed by probabilistic models. Conversely, a poorly-defined object is affected by imperfections that affect its definition, such as spatial vagueness and spatial ambiguity.

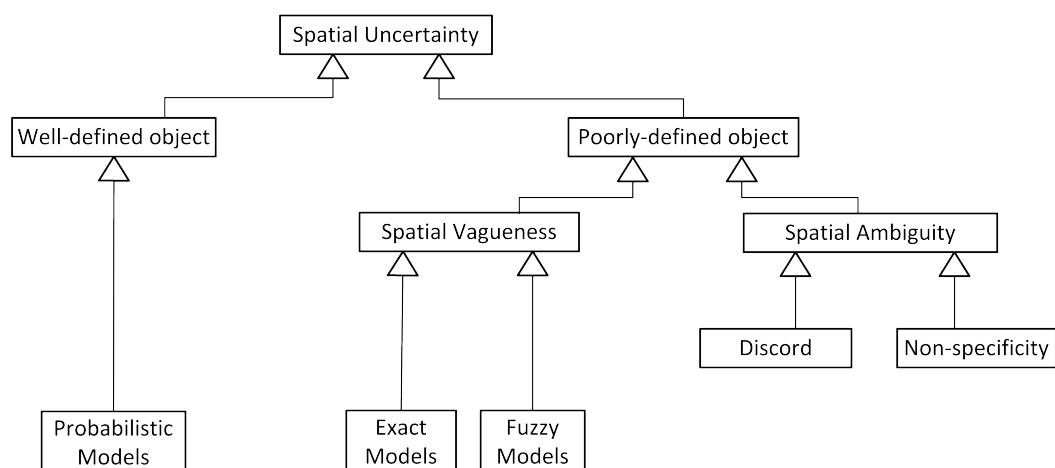


Figure 2.7: A conceptual model of uncertainty in spatial data (adapted from Fisher (1999) and Jadidi et al. (2014)).

Spatial vagueness influences the broadness of a spatial object's shape, i.e. its boundary and interior (HAZARIKA; COHN, 2001). If it is not possible to clearly define the shape of an object to be mapped or analyzed, the object is considered *vague*. There is not a combination of object's properties (attributes) that allows the definition of the precise shape of the object (FISHER, 1999).

A vague spatial object comprises parts that are *certainly* a member and parts that are *possibly* a member (ERWIG; SCHNEIDER, 1997). The membership degree to a given vague concept cannot be computed using a binary logic, i.e. true or false, because spatial vagueness impairs the unequivocal assignment of members to an object (FISHER, 1999). Temporality is another facet that may affect the shape of a spatial object (HAZARIKA; COHN, 2001).

Spatial vagueness has been mainly addressed by distinct models that assign membership degrees to vague concepts: exact models and fuzzy models. The difference between the terms vague and fuzzy refers exclusively to the approach used to define the model as an exact model based on crisp spatial data types or as an approach based on mathematical theories such as the fuzzy set theory. Regardless the term vague or fuzzy, these spatial data models comprise vague or fuzzy points, vague or fuzzy lines, vague or fuzzy regions (JADIDI et al., 2014). In this thesis, vague spatial data can be modeled either according to a vague spatial data type from an exact model or to a fuzzy spatial data type from a fuzzy model.

Exact models extend the research legacy regarding crisp spatial data by employing a 3-valued logic based on $\{true, maybe, false\}$ to define vague spatial data types, operators and topological relationships. These models reuse and adapt either the point-set topology (EGENHOFER; FRANZOSA, 1991) or the region connection calculus (RANDELL; CUI; COHN, 1992). A vague spatial object designed according an exact model has: (i) one geometric component that *certainly* belong to it, whose membership to the object is *true*; (ii) one geometric component that *possibly* belong to it, whose membership to the object is *maybe*; and (iii) an exterior whose membership to the object is *false*. All components are processed by operators, such as union, which merges a pair of vague spatial objects. Analogously, a vague spatial topological relationship yields either *true*, *maybe*, or *false*.

Conversely, fuzzy models are based on Fuzzy Set Theory (ZADEH, 1965). Fuzzy models define fuzzy spatial topologies and fuzzy spatial data types. A fuzzy spatial object is a vague spatial object with the additional feature of using a membership function that quantifies the membership degrees of its components. Membership values are in the interval $[0, 1]$. An example of membership function is a distance function. Furthermore, fuzzy models utilizes the fuzzy logic to determine the truth degree of a topological relationship. They also extend fuzzy set operations, such as union, to merge two fuzzy spatial objects.

In general, crisp spatial data and exact models neglect the loss of information as they adapt reality to a coarse representation with low resolution (HWANG; THILL, 2005). These representations are not able to adequately address phenomena that have characteristics of spatial object and continuous fields simultaneously (BURROUGH, 1996). Fuzzy models overcome these gaps

by adopting a multivalued logic with a continuum of truth values in the interval $[0,1]$, instead of a 3-valued logic (ZADEH, 1965). Since it is not feasible to store infinite elements of a fuzzy set representing a fuzzy spatial object, implementations for fuzzy models mainly hold geometric features and their corresponding membership values in $]0,1[$ (VERSTRAETE; HALLEZ; TRÉ, 2006; DILO, 2006; PETRY; LADNER; SOMODEVILLA, 2007; KANJILAL; LIU; SCHNEIDER, 2010; SCHNEIDER, 2014).

On the other hand, the utilization of fuzzy models might be unfeasible if elaborating a membership function to assign values in $]0,1[$ is impossible, or if a membership function does not exist (LEUNG, 1987; BURROUGH; FRANK, 1996). Then, adopting an exact model may become necessary. Alternatively, clustering and classification methods can be used to search for patterns within a dataset and assign membership values, e.g. fuzzy c-means algorithm and fuzzy neural networks (FISHER, 1999; JADIDI et al., 2014).

In this thesis, the design of vague SDWs tackles data types, operators and topological relationships from exact models, fuzzy models, and implementations for fuzzy models. Therefore, Section 2.3.1 addresses exact models, Section 2.3.2 addresses fuzzy models, Section 2.3.3 addresses implementations for fuzzy models, and Section 2.3.4 summarizes relevant characteristics of the studied models and implementations. The description of the taxonomy illustrated in Figure 2.7 is resumed and concluded in the following paragraph.

Different perceptions on the classification of a phenomenon cause spatial ambiguity (FISHER, 1999), as distinct results are obtained using divergent classification methods for the same set of elements (BEJAOU, 2009). It is noteworthy that ambiguity results from the classification process itself and not from the characteristics of the classes involved (BEJAOU, 2009). Discord is a subtype of spatial ambiguity and occurs when completely different naming conventions are used by designers (OORT, 2006). Conflicting territorial claims, such as the existence or non-existence of a nation of Kurds, exemplify a ambiguity through discord in the creation of maps (FISHER, 1999). Non-specificity is also a subtype of spatial ambiguity that may affect spatial relationships, as exemplified by Fisher (1999). For the statement “A is north of B”, both of the following definitions are valid, but hold ambiguity through non-specificity: (i) A lies on exactly the same line of longitude and towards the north pole from B; and (ii) A lies somewhere to the north of a line running east to west through B.

2.3.1 Exact Models for Spatial Vagueness

Exact models for spatial vagueness or simply exact models define vague spatial data types, vague operators and vague topological relationships. Their comprehension is essential to enable

Table 2.2: Nomenclatures adopted by exact models for components of a vague spatial object.

Component	Egg-Yolk	RBB	QMM	VASA
Membership is true	Yolk	Inner boundary	Minimal extent	Kernel
Membership is maybe	White	Broad boundary	Maximal extent	Conjecture

the design of vague SDWs and provide both multidimensional and spatial analyses. The next sections survey the following exact models for spatial vagueness, or simply exact models: the Egg-Yolk model proposed by Cohn & Gotts (1996), the model for regions with broad boundaries (RBB) introduced by Clementini & Felice (1996), the qualitative min-max (QMM) model described by Bejaoui et al. (2009, 2010), and the vague spatial algebra (VASA) proposed by Pauly & Schneider (2010). Section 2.3.1.1 addresses vague spatial data types, Section 2.3.1.2 tackles vague spatial operators, and Section 2.3.1.3 stands for vague topological relationships.

2.3.1.1 Vague Spatial Data Types

Exact models describe a vague spatial object using a pair of geometric components, as illustrated in Figure 2.8. One component *certainly* belongs to the object, its membership is *true*, and its color is black in Figure 2.8. The other component *possibly* belongs to the object, its membership is *maybe*, and its color is gray in Figure 2.8. Exact models adopt distinct nomenclatures for these components, as listed in Table 2.2.

In VASA, a vague point P has the point set P_k as *kernel* and the point set P_c as *conjecture*. The sets are disjoint. A vague point P , according to VASA, is interpreted as a set of known locations where some locations *certainly* belong to P , while other locations *possibly* belong to P . Figure 2.8a-d exemplify vague points designed according to VASA.

In the QMM model, a vague point P is a broad point visualized as a region, as exemplified in Figure 2.8e. It is interpreted as an undetermined location within a determined area. P has an empty *minimal extent* P_{min} and a non-empty *maximal extent* P_{max} composed of the area occupied by P .

The QMM model defines 9 different subtypes of vague line. The endpoints and the interior points of a crisp line are strictly crisp points as exemplified in Figure 2.8f. Conversely, the endpoints and/or the interior points of a vague line can be crisp points or vague points. Thus, vague lines of the QMM model assume broad boundary and/or broad interior, as exemplified in Figures 2.8g-l. Additionally, a completely vague line is a broad line as exemplified in Figure 2.8m. The *maximal extent* L_{max} of a vague line L encompasses the set of vague end and interior points from L . The *minimal extent* L_{min} of a vague line L encompasses one or more lines from L that

do not cross themselves and that do not form a loop.

In VASA, a vague line L has the line set L_k as *kernel* and the line set L_c as *conjecture*. Distinct lines can meet but cannot overlap. Figure 2.8f,n-p exemplify vague lines designed according to VASA.

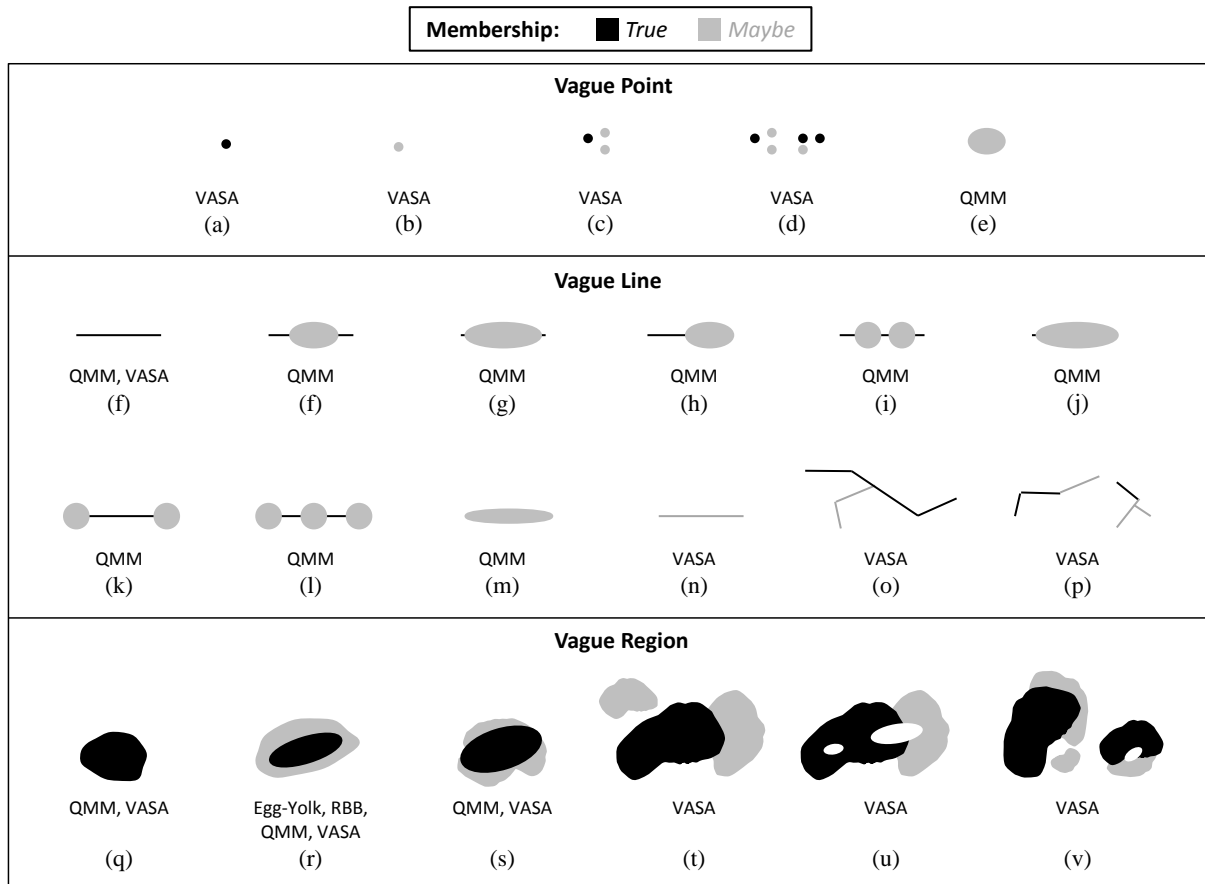


Figure 2.8: Examples of vague spatial objects designed according to existing exact models for spatial vagueness.

In the Egg-Yolk model, a vague region R is composed of a pair of concentric regions, such that the inner is the *yolk* R_{yolk} , the outer is the *white* R_{white} , and they together compose the *egg* R . There is an explicit constraint that the boundary of the white does not touch the boundary of the yolk. Conceptually, the indeterminate boundary of the vague region R is an outline within the white. Figure 2.8r exemplifies a vague region designed according to the Egg-Yolk model.

In the RBB model, a region with broad boundaries R is a pair of sets in \mathbb{R}^2 : R_{in} and R_{out} such that $R_{in} \subseteq R_{out}$. Then, ∂R_{in} is the internal boundary of R , ∂R_{out} is the external boundary of R , $\Delta R = R_{in} - R_{out}^\circ$ is the broad boundary of R in \mathbb{R}^2 and $R^\circ = R_{out} - \Delta R$ is the interior of R . Figure 2.8r exemplifies a vague region designed according to the RBB model.

In the QMM model, a vague region R has the region R_{max} as *maximal extent* and the region

R_{min} as *minimal extent*. R_{max} is comparable to the egg from the Egg-Yolk model and to the external boundary of the RBB model. Conversely, R_{min} is comparable to the yolk from Egg-Yolk model and to the internal boundary of the RBB model. Figure 2.8q-s exemplify vague regions designed according to the QMM model. Differently from the Egg-Yolk and the RBB models, the QMM model constrains a vague region R such that:

$$Equals(R_{min}, R_{max}) \vee Contains(R_{min}, R_{max}) \vee Covers(R_{min}, R_{max}).$$

In VASA, a vague region R has the region set R_k as *kernel* and the region set R_c as *conjecture*. Points within the kernel R_k *certainly* belong to R , while points within the conjecture R_c *possibly* belong to R . Figure 2.8q-v exemplify vague regions designed according to VASA. Differently from the Egg-Yolk, RBB and the QMM models, VASA constrains a vague region R such that the interior of the kernel R_k is disjoint from the interior of the conjecture R_c . In other words:

$$Disjoint(R_k, R_c) \vee Touches(R_k, R_c).$$

2.3.1.2 Vague Spatial Geometric Set Operators

Concerning the exact models that have been surveyed in this chapter, only VASA comprises operators that processes vague spatial objects and return a vague spatial object, as *union*, *intersection* and *difference*, since its vague spatial data types are closed under these operations.

The union of the pair of vague spatial objects A and B produces a vague spatial object with the same data type of both A and B . The kernel of the resulting vague spatial object is the union of the kernels of A and B . Also, the conjecture of the resulting vague spatial object comprises the union of the conjectures of A and B , except from the extent that already belong to the kernel. Considering that \oplus is the geometric union and \ominus is the geometric difference, then:

$$A \text{ union } B = (A_k \oplus B_k, (A_c \oplus B_c) \ominus (A_k \oplus B_k))$$

The intersection of the pair of vague spatial objects A and B produces a vague spatial object with the same data type of both A and B . The kernel of the resulting vague spatial object encompasses the intersection between the kernels of A and B . The conjecture of the resulting vague spatial object comprises the intersections between the kernel of A and the conjecture of B , between the conjecture of A and the kernel of B , and between the conjectures of A and B . Considering that \otimes is the geometric intersection, then:

$$A \text{ intersection } B = (A_k \otimes B_k, (A_c \otimes B_c) \oplus (A_k \otimes B_c) \oplus (A_c \otimes B_k))$$

The difference of the pair of vague spatial objects A and B produces a vague spatial object with the same data type of both A and B . The kernel of the resulting vague spatial object encompasses parts of the kernel of A that do not intersect B . The conjecture of the resulting vague spatial object comprises the intersection of the conjectures of A and B , the intersection of the kernel of A and the conjecture of B , and parts of the conjecture of B that do not intersect A . Considering that \oplus is the geometric union, \otimes is the geometric intersection, and \ominus is the geometric complement (the set of points that do not belong to a given geometry), then:

$$A \text{ difference } B = (A_k \otimes (\ominus(B_k \oplus B_c)), (A_c \otimes B_c) \oplus (A_k \otimes B_c) \oplus (A_c \otimes (\ominus(B_k \oplus B_c))))$$

2.3.1.3 Vague Topological Relationships

Let A and B be two vague regions modeled according to the Egg-Yolk model. Then, the topological relationship between A and B is an interpretation of four region connection calculi, such that each calculus C is in $\{Distinct\ Regions, Partially\ Overlapping, Proper\ Part, Equal, Proper\ Part\ Inverse\}$ (RANDELL; CUI; COHN, 1992):

$$C(A_{yolk}, B_{yolk})$$

$$C(A_{yolk}, B_{white})$$

$$C(A_{white}, B_{yolk})$$

$$C(A_{white}, B_{white})$$

In Figure 2.9, the analysis of the four calculi concludes that A nearly overlaps B .

Let A and B be two vague regions modeled according to the RBB model. Then, then the topological relationship between A and B is an interpretation of a 3×3 matrix of intersections, such that $^\circ$, Δ , and $^-$ denote interior, broad boundary, and exterior, respectively:

$$\begin{pmatrix} A^\circ \cap B^\circ & A^\circ \cap \Delta B & A^\circ \cap B^- \\ \Delta A \cap B^\circ & \Delta A \cap \Delta B & \Delta A \cap B^- \\ A^- \cap B^\circ & A^- \cap \Delta B & A^- \cap B^- \end{pmatrix}$$

In Figure 2.9, the analysis of the 3×3 matrix concludes that the relationship number 12 specified by Clementini & Felice (1996) is held by A and B .

Let A and B be two vague spatial objects modeled according to the QMM model. Then, the topological relationship between A and B is an interpretation of a 2×2 matrix where R_1 , R_2 , R_3 ,

$R_4 \in \{ \text{meets, contains, inside, equals, overlaps, covers, covered by, disjoint} \}$:

$$\begin{pmatrix} R_1(A_{min}, B_{min}) & R_2(A_{min}, B_{max}) \\ R_3(A_{max}, B_{min}) & R_2(A_{max}, B_{max}) \end{pmatrix}$$

In Figure 2.9, the analysis of the 2×2 matrix concludes that A strongly overlaps B , according to the relationship number 190 specified by Bejaoui et al. (2010).

Let A and B be two vague spatial objects modeled according to VASA and \oplus be the geometric union. Then, the topological relationship between A and B is an interpretation of $p, q, r, s, v, w \in \{ \text{meets, contains, inside, equals, overlaps, covers, covered by, disjoint} \}$:

$$\begin{aligned} & p(A_k, B_k) \wedge \\ & q(A_k \oplus A_c, B_k) \wedge \\ & r(A_k, B_k \oplus B_c) \wedge \\ & s(A_k \oplus A_c, B_k \oplus B_c) \wedge \\ & v(A_k, A_k \oplus A_c) \wedge \\ & w(B_k, B_k \oplus B_c) \end{aligned}$$

In Figure 2.9, the analysis of the six conjunctions concludes that A maybe meets B .

2.3.2 Fuzzy Models for Spatial Vagueness

Exact models for spatial vagueness or simply exact models define vague spatial data types, vague operators and vague topological relationships. Their comprehension is essential to enable the design of vague SDWs and provide both multidimensional and spatial analyses. The next sections survey the following exact models for spatial vagueness, or simply exact models: the Egg-Yolk model proposed by Cohn & Gotts (1996), the model for regions with broad boundaries (RBB) introduced by Clementini & Felice (1996), the qualitative min-max (QMM) model described by Bejaoui et al. (2009, 2010), and the vague spatial algebra (VASA) proposed by Pauly & Schneider (2010). Section 2.3.1.1 addresses vague spatial data types, Section 2.3.1.2 tackles vague spatial operators, and Section 2.3.1.3 stands for vague topological relationships.

2.3.2.1 Fuzzy Spatial Data Types

Figures 2.10a-1 exemplify vague spatial objects represented according to different fuzzy spatial data types. The shaded representation using gradients indicate a higher membership values for black and darker tones of gray, a lower membership value for brighter tones of gray and membership value equal to 0 for white.

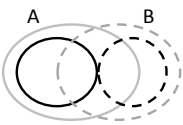
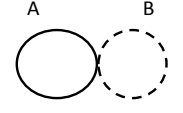
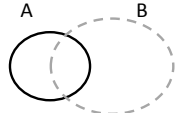
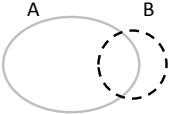
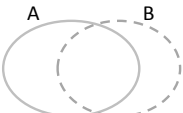
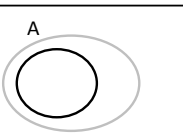
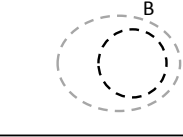
	Egg-Yolk	RBB	QMM	VASA
	$C(A_{yolk}, B_{yolk})$ Distinct Regions	$A^\circ \cap B^\circ$ is false False	$R_1(A_{min}, B_{min})$ Meets	$p(A_k, B_k)$ Meets
	$C(A_{yolk}, B_{white})$ Partially Overlapping	$A^\circ \cap \Delta B$ True	$R_2(A_{min}, B_{max})$ Overlaps	$q(A_k \oplus A_{c'}, B_k)$ Overlaps
	$C(A_{white}, B_{yolk})$ Partially Overlapping	$\Delta A \cap B^\circ$ True	$R_3(A_{max}, B_{min})$ Overlaps	$r(A_k, B_k \oplus B_{c'})$ Overlaps
	$C(A_{white}, B_{white})$ Partially Overlapping	$\Delta A \cap \Delta B$ True	$R_4(A_{max}, B_{max})$ Overlaps	$s(A_k \oplus A_{c'}, B_k \oplus B_{c'})$ Overlaps
	-	-	-	$v(A_k, A_k \oplus A_{c'})$ Inside
	-	-	-	$w(B_k, B_k \oplus B_{c'})$ Inside
How are A and B related?	<i>A nearly overlaps B</i> (9)	$\begin{pmatrix} 0 & T & T \\ T & T & T \\ T & T & T \end{pmatrix}$ (12)	$\left\{ \begin{array}{l} \text{Meets Overlaps} \\ \text{Overlaps Overlaps} \end{array} \right\}$ <i>A strongly overlaps B</i> (190)	$\text{Meets} \wedge \text{Overlaps} \wedge \text{Overlaps} \wedge \text{Overlaps} \wedge \text{Overlaps} \wedge \text{Inside} \wedge \text{Inside}$ <i>A maybe overlaps B</i>

Figure 2.9: Vague regions A and B and the topological relationships between them according to existing exact models.

Dilo (2006), Dilo, By & Stein (2007) and Schneider (2008) defined a simple fuzzy point as an exact location (a, b) in \mathbb{R}^2 with a membership value in $]0, 1]$ concerning the observed phenomenon. Therefore, a simple fuzzy point is the unitary fuzzy set $\mu_{\tilde{p}(a,b)}$ such that:

$$\mu_{\tilde{p}(a,b)}(x, y) = \begin{cases} m & \text{if } (x, y) = (a, b), \text{ where } 0 < m \leq 1 \\ 0 & \text{otherwise.} \end{cases}$$

Figure 2.10a shows a simple fuzzy point with membership value 1, while Figure 2.10b depicts a simple fuzzy point with membership value in $]0, 1[$.

Dilo (2006) and Dilo, By & Stein (2007) also defined the fuzzy point \tilde{P} as a finite set of one

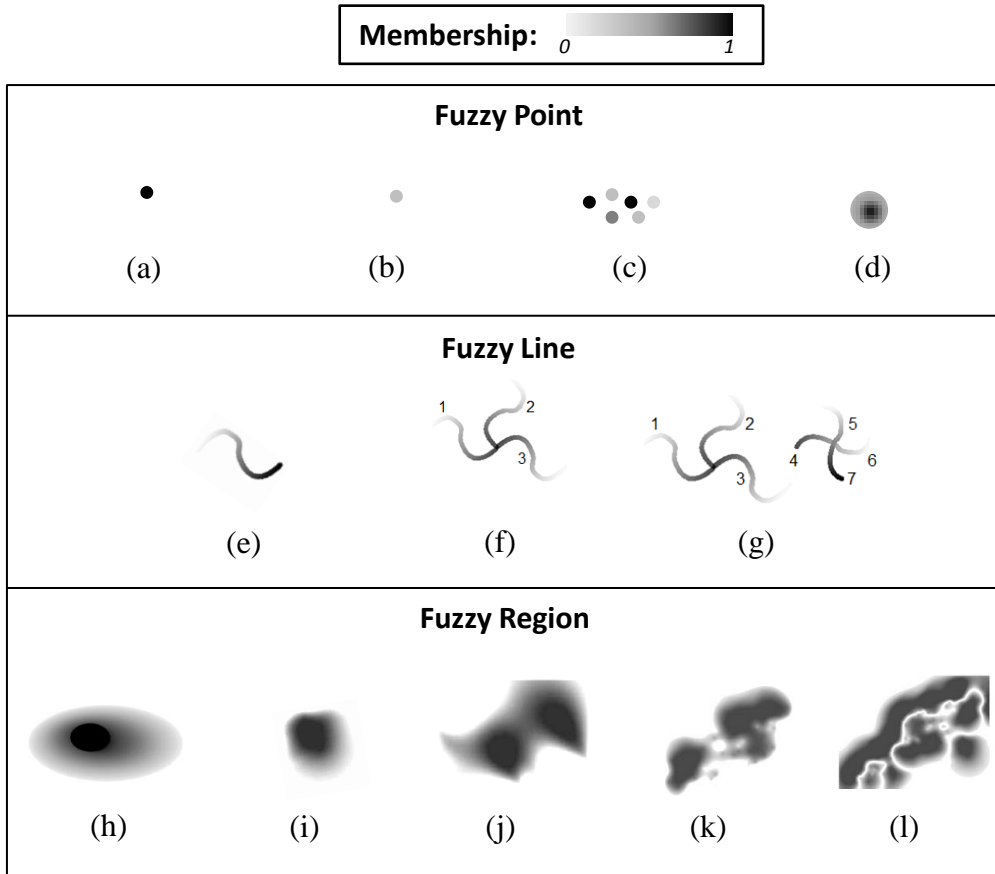


Figure 2.10: Example of vague spatial objects designed according to existing fuzzy models for spatial vagueness.

or more disjoint simple fuzzy points \tilde{p} as:

$$\mu_{\tilde{p}} = \bigwedge_{i=1}^n \mu_{\tilde{p}_i(a,b)}.$$

Figure 2.10c exemplifies a fuzzy point composed of six simple fuzzy points.

Besides, Schneider (2008) defined a broad fuzzy point composed of all pairs of coordinates (x,y) within an arbitrary distance from the coordinates (a,b) . The membership values are assigned according to a distance function, which is commonly a circle equation. As a result, there is a buffer zone circumscribing (a,b) composed of simple fuzzy points. As the distance from (a,b) increases the membership value decreases, and vice-versa. Such definition of fuzzy point is:

$$\mu_{\tilde{p}(a,b)}(x,y) = \begin{cases} 1 - \frac{\sqrt{(x-a)^2 + (y-b)^2}}{r} & \text{if } (x-a)^2 + (y-b)^2 \leq r^2 \\ 0 & \text{otherwise.} \end{cases}$$

Figure 2.10d exemplifies a broad fuzzy point.

Dilo (2006), Dilo, By & Stein (2007) and Schneider (2008) agreed that a simple fuzzy line is a (curvi)linear feature with exact location but with a vague outline composed of simple fuzzy points \tilde{p} . The membership values of these points vary smoothly between the endpoints $f_{\tilde{l}}(0)$ and $f_{\tilde{l}}(1)$, whose membership values are the lowest and the greatest, respectively. Although stepwise variations in the membership values may occur, discontinuities and self-intersection of the interior are disallowed. A simple fuzzy line \tilde{l} comprising these characteristics is exemplified in Figure 2.10e.

Schneider (2008) also defined a fuzzy block \tilde{b} as a finite set of simple fuzzy lines that have disjoint interiors and that share one endpoint whose membership value is the same. Figure 2.10f exemplifies a fuzzy block composed of three numbered simple fuzzy lines that share a common endpoint. The membership value in such endpoint is the same for the three simple fuzzy lines.

Schneider (2008) extended the definition of fuzzy block to define a fuzzy line \tilde{L} as being a collection of disjoint fuzzy blocks. The fuzzy line defined by Dilo (2006) and Dilo, By & Stein (2007) also complied with the definitions and constraints of the fuzzy line proposed by Schneider (2008). Figure 2.10g exemplifies a fuzzy line composed of two fuzzy blocks, such that one block has three simple fuzzy lines and the other block has four simple fuzzy lines.

Verstraete, Hallez & Tré (2006) defined a simple fuzzy region as a set of simple fuzzy points. With U being the universe of locations p , the membership $\mu_{\tilde{A}}(p)$ quantified the degree of membership of p to the simple fuzzy region \tilde{r} , i.e.:

$$\tilde{r} = \{(p, \mu_{\tilde{r}}(p))\}$$

where $\mu_{\tilde{r}} : U \rightarrow [0, 1]$ and $p \mapsto \mu_{\tilde{r}}(p)$.

Given a fuzzy set \tilde{A} , an α -cut \tilde{A}_α is a set of elements of \tilde{A} such that the membership value is greater than or equal to α :

$$\tilde{A}_\alpha = \{x \in \mathbb{R}^n \mid \tilde{A}(x) \geq \alpha\}$$

Figure 2.11 depicts the simple fuzzy region \tilde{r} and highlights the interval $[a, b]$ that encompasses the α -cut for $\alpha = 0.5$. Figure 2.10h repeats \tilde{r} , while Figure 2.10i depicts another simple fuzzy region.

Although Dilo (2006), Dilo, By & Stein (2007), Schneider (2008), and Tang, Kainz & Wang (2010) adopted distinct topologies, they agreed that a simple fuzzy region is composed of a set of simple fuzzy points whose membership values vary gradually and smoothly among neighbor points. In general, they defined a fuzzy region \tilde{R} as a finite set of simple fuzzy regions and disallowed the following irregularities: abrupt variation of the membership value, isolated

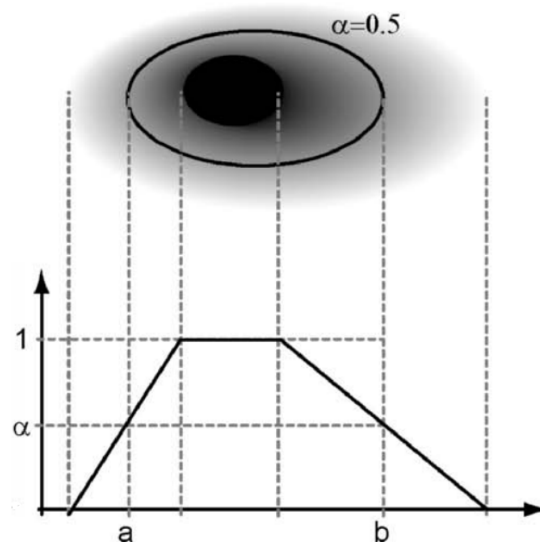


Figure 2.11: A simple fuzzy region and its membership function (VERSTRAETE; HALLEZ; TRÉ, 2006).

points, and isolated lines. Figures 2.10j-l depict fuzzy regions encompassing different numbers of simple fuzzy regions.

Dilo (2006), Dilo, By & Stein (2007), Schneider (2008), and Hazarika & Hazarika (2012) allowed disjoint holes in a fuzzy region if, and only if they are not located over the following transitions of membership values: from 1 to 0 and from 0 to more than 0. Tang, Kainz & Wang (2010) also disallowed a fuzzy region to have holes where the membership value is equal to 1.

2.3.2.2 Fuzzy Spatial Operators

Fuzzy models' operators are based upon the following operators for union, intersection and difference of fuzzy sets (ZADEH, 1965).

The union of two fuzzy sets \tilde{A} and \tilde{B} is $\tilde{A} \sqcup \tilde{B}$, whose membership function yields the maximum membership values of the functions $\mu_{\tilde{A}}(x)$ and $\mu_{\tilde{B}}(x)$:

$$\mu_{\tilde{A} \sqcup \tilde{B}}(x) = \text{Max}[\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)], x \in \mathbb{R}^n$$

The union of fuzzy sets was adapted by Dilo (2006), Dilo, By & Stein (2007), and Schneider (2008) to yield the maximum membership value of a pair of operands that belong to the same fuzzy spatial data type, i.e. two fuzzy points, or two fuzzy lines, or two fuzzy regions. In general, each fuzzy model defined its own operators according to the adopted topology and to the available data types. Intersection, complement and difference, which are described in the following, were extended analogously.

The intersection of two fuzzy sets \tilde{A} and \tilde{B} is $\tilde{A} \cap \tilde{B}$, whose membership function yields the minimum membership values of the functions $\mu_{\tilde{A}}(x)$ and $\mu_{\tilde{B}}(x)$:

$$\mu_{\tilde{A} \cap \tilde{B}}(x) = \text{Min}[\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)], x \in \mathbb{R}^n$$

The complement of a fuzzy set \tilde{A} is \tilde{A}' whose membership function is:

$$\mu_{\tilde{A}'}(x) = 1^{\mathbb{R}^n} - \mu_{\tilde{A}}$$

The difference of two fuzzy sets \tilde{A} and \tilde{B} is $\tilde{A} - \tilde{B}$, whose membership function yields the minimum membership values of the functions $\mu_{\tilde{A}}(x)$ and $\mu_{\tilde{B}'}(x)$ (complement of \tilde{B}):

$$\mu_{\tilde{A} - \tilde{B}}(x) = \text{Min}[\mu_{\tilde{A}}(x), \mu_{\tilde{B}'}(x)], x \in \mathbb{R}^n.$$

2.3.2.3 Fuzzy Spatial Topological Relationships

Some authors defined fuzzy topological relationships based on the crisp topological relationships commented in Section 2.1.1.3. Zhan (1998) determined the fuzzy topological relationship R between two simple fuzzy regions \tilde{A} and \tilde{B} by using a finite number of α -cuts without holes $\alpha_1=1 > \alpha_2 > \dots > \alpha_n = 0$, which are generated using a pair of input α -cuts provided as reference. $R(\tilde{A}, \tilde{B})$ assumes a value in $[0, 1]$, such that:

$$R(\tilde{A}, \tilde{B}) = \sum_{i=1}^{n-1} \sum_{j=1}^{n-1} (\alpha_i - \alpha_{i+1}) \times (\alpha_j - \alpha_{j+1}) \times R(\tilde{A}_{\alpha_i}, \tilde{B}_{\alpha_j})$$

where $R(\tilde{A}_{\alpha_i}, \tilde{B}_{\alpha_j})$ is the truth value 1 or 0 regarding *meets (touches)*, *contains*, *inside (within)*, *equals*, *overlaps*, *intersects*, *covers*, *covered by* or *disjoint*.

Figure 2.12a exemplifies a pair of input α -cuts in gray provided as reference for determining the topological relationship between the simple fuzzy regions \tilde{A} and \tilde{B} . More α -cuts are generated, for each vague region, between the black α -cut and the gray α -cut. They have α in $]0, 1[$, since the black α -cuts have $\alpha = 1.0$ and the gray α -cuts have the minimum α values. The evaluation of $\text{disjoint}(\tilde{A}, \tilde{B})$ yields a value between 0 and 1, since the reference α -cuts clearly have $\text{disjoint}(\tilde{A}_{\alpha_0}, \tilde{B}_{\alpha_0}) = 0$ because they are not disjoint.

Schneider (2008) extended the method of Zhan (1998) to provide support for fuzzy points, fuzzy lines and fuzzy regions, as well as to allow linguistic labels provided by either a trapezoidal membership function or a triangular membership function. For instance, if $\text{disjoint}(\tilde{A}, \tilde{B}) = 0.30$ in Figure 2.12a, then \tilde{A} e \tilde{B} are *slightly* disjoint according to the membership functions shown in Figure 2.12b.

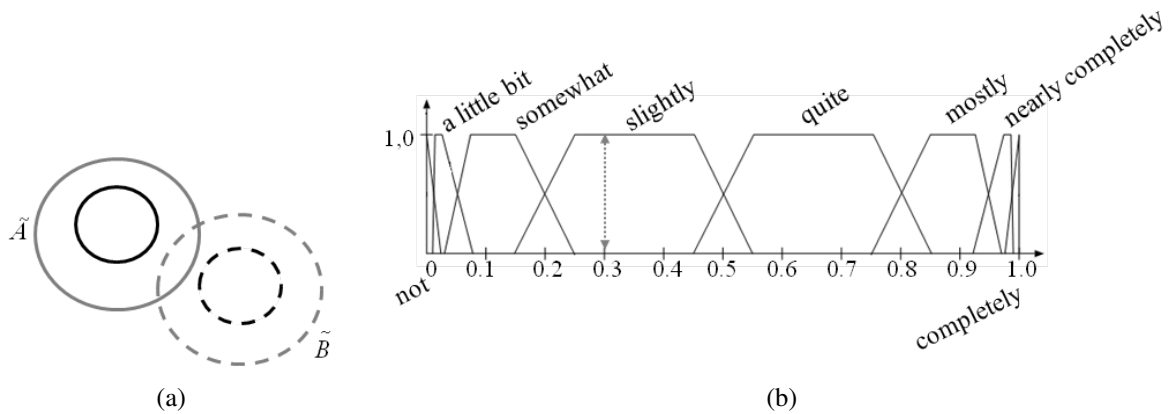


Figure 2.12: Evaluating the fuzzy topological relationship between two fuzzy regions. (a) Pairs of reference α -cuts in black, and $\tilde{A}_{\alpha_0}, \tilde{B}_{\alpha_0}$ in gray. (b) Membership functions and the linguistic labels they assume.

Other authors adapted the 9-intersection matrix commented in Section 2.1.1.3, as follows. Shi & Liu (2007) and Liu & Shi (2009) reused the 9-intersection matrix and, for each cell, used integrals to quantify the intersection between the components of the inputs \tilde{A} and \tilde{B} . Indeed, these integrals correspond to the area shared by the both input membership functions. Also, these integrals fit the data type of the inputs, i.e. surface integral for a simple fuzzy region and line integral for a simple fuzzy line. The fuzzy model described by Tang, Kainz & Wang (2010) adopt a topology that defines fuzzy regions with five components. Therefore, a 5×5 matrix is used to identify intersections between a pair of components from the input fuzzy regions \tilde{A} and \tilde{B} . The existence of intersection is indicated by 1, while the absence of intersection is indicated by 0. Hazarika & Hazarika (2012) defined a matrix to relate a fuzzy region \tilde{A} with n holes and \tilde{B} , which is either a fuzzy point, a fuzzy line or a fuzzy region. The number of cells of the matrix varies according to the number of holes n in \tilde{A} .

2.3.3 Implementations for Fuzzy Models

Fuzzy models have been implemented in several different manners. The comprehension of these implementations is essential to indicate requirements that the design of vague SDWs should satisfy. The following sections survey existing implementations for fuzzy models that both support one or more fuzzy spatial data types and describe their operators. Section 2.3.3.1 addresses spatial plateau objects in the context of the plateau algebra (KANJILAL; LIU; SCHNEIDER, 2010; SCHNEIDER, 2014). Section 2.3.3.2 tackles fuzzy lines implemented as lines with gradual transitions (DILO, 2006). Section 2.3.3.3 stands for fuzzy regions implemented as bitmaps (VERSTRAETE et al., 2005; VERSTRAETE; TRÉ; HALLEZ, 2006). Section 2.3.3.4 focuses

on fuzzy regions implemented as triangulated irregular networks (DILO et al., 2006; VERSTRAETE et al., 2007). Finally, Section 2.3.3.5 addresses fuzzy regions implemented as minimum bounding rectangles (SOMODEVILLA; PETRY, 2003).

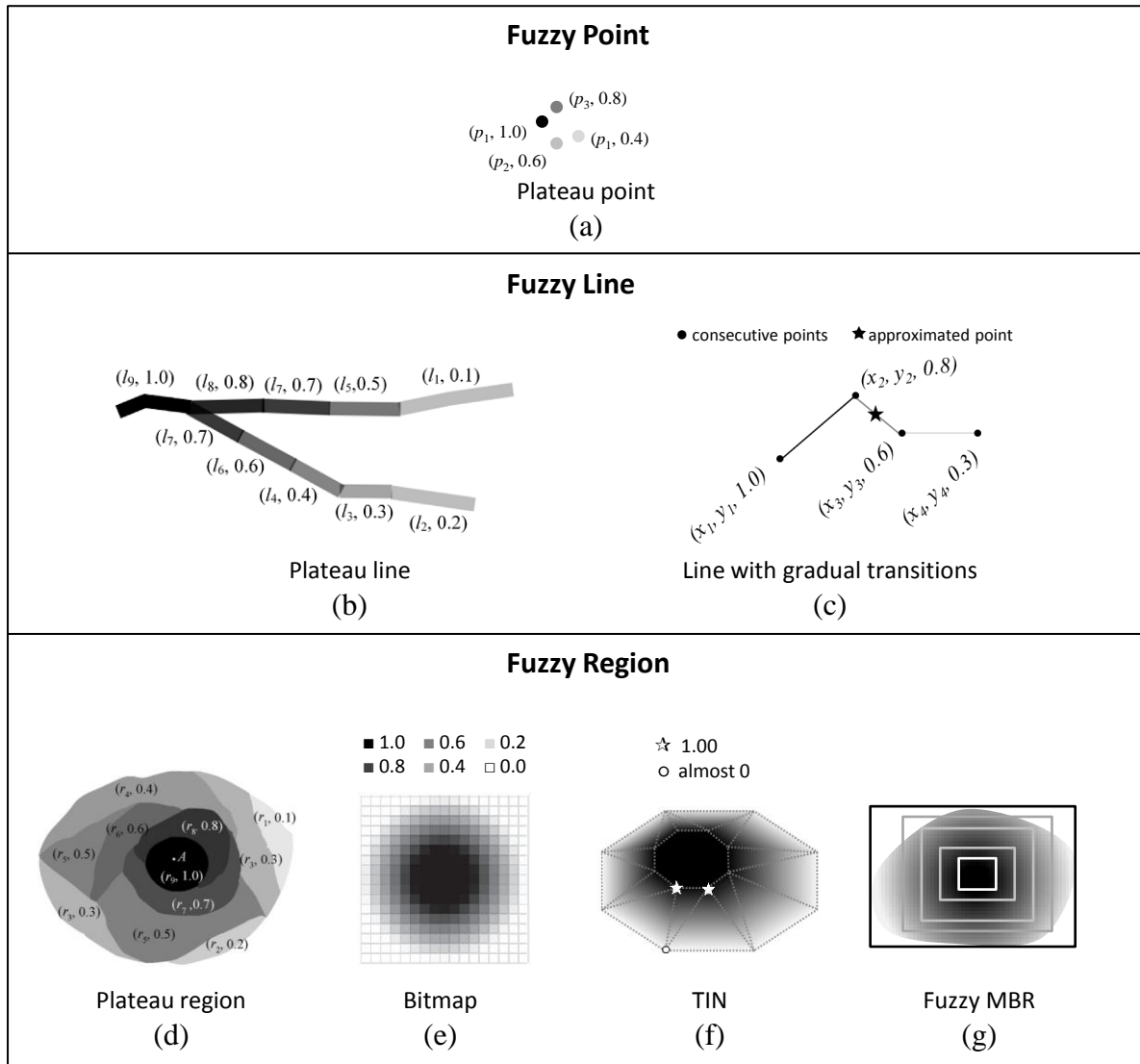


Figure 2.13: Examples of implemented fuzzy point, fuzzy lines and fuzzy regions.

2.3.3.1 Spatial Plateau Objects

The plateau algebra provides the following implementations of fuzzy spatial data types: plateau point, plateau line and plateau region. A plateau spatial object po represents the fuzzy spatial object \tilde{o}_i with n subsets as n pairs containing a geometry $o_{i,j}$ and its corresponding membership value m_{i,n_j} , for $1 \leq i \leq j \leq n$:

$$po = \langle (o_{i,1}, m_{i,1}) \dots (o_{i,n_i}, m_{i,n_i}), o_i \rangle$$

The pairs are ordered in ascending order of membership values and each geometry $o_{i,j}$ approximates a subset from \tilde{o}_i . Every geometry $o_{i,j}$ has the same crisp spatial data type closed under geometric set operations, such as those from OGC and implemented by PostGIS and Oracle Spatial and Graph. Also, $o_i = \bigoplus_{j=1}^{n_i} o_{i,j}$, where \bigoplus denotes geometric union.

A plateau point is a finite set of disjoint points (or point sets) with distinct membership values. Figure 2.13a exemplifies the plateau point $pp = \langle (p_1, 0.4), (p_2, 0.6), (p_3, 0.8), (p_4, 1.0), p_1 \rangle$. A plateau line is a finite set of lines (or line sets) with distinct membership values. The component lines (or line sets) are disjoint or meet or have a finite number of intersection points. Figure 2.13b exemplifies the plateau line $pl = \langle (l_1, 0.1), (l_2, 0.2), (l_3, 0.3), (l_4, 0.4), (l_5, 0.5), (l_6, 0.6), (l_7, 0.7), (l_8, 0.8), (l_9, 1.0), l \rangle$. A plateau region is a finite set of crisp adjacent or disjoint regions (or region sets) with distinct membership values. Figure 2.13d depicts the plateau region $pr = \langle (r_1, 0.1), (r_2, 0.2), (r_3, 0.3), (r_4, 0.4), (r_5, 0.5), (r_6, 0.6), (r_7, 0.7), (r_8, 0.8), (r_9, 1.0), r \rangle$. The union of two plateau spatial objects requires performing geometric union and yielding the maximum membership value where the plateau spatial objects intersect.

2.3.3.2 Lines with Gradual Transitions

Fuzzy lines are also represented by lines with gradual transitions, by applying a function over membership values of two consecutive vertices of a linear feature. The line with gradual transition \tilde{l} is a sequence of triples, each triple providing the (x, y) coordinates to locate a point in the line with the corresponding membership value mv :

$$\tilde{l} = (x_1, y_1, mv_1), \dots, (x_n, y_n, mv_n).$$

Consecutive points are interpolated to approximate the line tracing and its membership values. Figure 2.13c depicts a line with gradual transition holding four consecutive points and their membership values, the approximate tracing of the line and one highlighted approximated point whose membership value is omitted, but given by a linear interpolation with the vertices (x_2, y_2, mv_2) and (x_3, y_3, mv_3) .

2.3.3.3 Bitmaps

The bitmap is a raster approach that implements fuzzy regions. It distributes a finite number of rectangular or hexagonal cells c_i over a grid G and associates a membership value m_i in $]0,1]$ to each cell:

$$fuzzyBitmap = \{c_i, m_i \mid c_i \in G\}.$$

Each cell covers a limited extent and holds one membership value. Figure 2.13e depicts a bitmap where membership values are shown for each cell. Both the union and the intersection of two input bitmaps requires an overlay and the construction of a new grid that combines the grids of the input bitmaps.

2.3.3.4 Triangulated Irregular Networks

A triangulated irregular network (TIN) models a fuzzy region by creating non-overlapping triangles whose vertices have membership values. Let P be the set of input data points on which the TIN is constructed, E be a set of edges, and T be the set of triangles that compose the TIN, then a fuzzy region represented by a TIN is:

$$fuzzyTIN = [(P, E, T), f].$$

Figure 2.13f depicts a TIN whose triangles with membership value 1 were omitted. The vertices and the corresponding membership values are shown for one of the triangles. Let each triangle be defined by the coordinates of the three vertices (x, y) and their membership values z , i.e. (x_1, y_1, z_1) , (x_2, y_2, z_2) and (x_3, y_3, z_3) . The membership value of any point inside a triangle is obtained by a linear interpolation considering the membership values of the vertices, i.e.:

$$-\frac{A}{C}x - \frac{B}{C}y - \frac{D}{C} \quad (2.1)$$

such that:

$$A = y_1(z_2 - z_3) + y_2(z_3 - z_1) + y_3(z_1 - z_2)$$

$$B = z_1(x_2 - x_3) + z_2(x_3 - x_1) + z_3(x_1 - x_2)$$

$$C = x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)$$

$$D = -Ax_1 - By_1 - Cz_3$$

The union of two TINs requires rebuilding the frontier of the TINs using Delaunay Triangulation, removing the overlapping edges and triangles in holes and add more vertices to the resulting TIN in order to assign them more realistic membership values.

2.3.3.5 Fuzzy Minimum Bounding Rectangles

The fuzzy minimum bounding rectangle, or simply fuzzy MBR, approximates a fuzzy region by creating: (i) a rectangle with that is circumscribed by the fuzzy region; (ii) the MBR of the fuzzy region, which has them minimum membership value; and (iii) a finite set of α MBR-

cuts that are rectangles with increasing membership values between the circumscribed rectangle and the MBR. The circumscribed rectangle has membership value 1.0, the MBR has the minimum membership value and the α MBR-cuts have increasing membership values as closer they are from the circumscribed rectangle and decreasing membership values as closer they are to the MBR. Figure 2.13g depicts a fuzzy region and its fuzzy MBR, where the circumscribed rectangle is white, α MBR-cuts are gray and the MBR is black. The union (overlay) of two input fuzzy MBRs requires the construction of a grid.

2.3.4 Summary

Spatial vagueness is an imperfection of spatial data that has been tackled by the work surveyed in Sections 2.3.1 to 2.3.3. Those work are listed and compared in Table 2.3. They are classified as exact models, fuzzy models, and implementations for fuzzy models. Data types supported are characterized according to their complexity, i.e. simple point “.”, simple line “~”, simple region “ \square ”, complex point “.:”, complex line “ \approx ”, or complex region “ $\square\square$ ”. Note that complex data types allow collections, holes, and islands, for example. The existence of operators for union “ \cup ”, intersection “ \cap ”, and difference “ $-$ ” of a pair of spatial objects is indicated. The approach extended to describe topological relationships is also specified, i.e. the region connection calculus “RCC” from Randell, Cui & Cohn (1992) or the 9-intersection model “9IM” from Egenhofer & Franzosa (1991), Egenhofer & Herring (1991).

Egg-Yolk (COHN; GOTTS, 1996), RBB (CLEMENTINI; FELICE, 1996), QMM (BEJAOUI et al., 2009, 2010), and VASA (PAULY; SCHNEIDER, 2010) are exact models that adopt a 3-valued logic to assign a membership to each component of an object and to describe vague spatial data types, operators, and topological relationships. The definitions of vague spatial data types revealed several incompatibilities among the models. For instance, QMM allows a vague line to be composed of lines and regions, while VASA allows a vague line strictly to be composed of a pair of lines or a pair of line sets. All the aforementioned exact models explore the Cartesian product of relationships among crisp and vague components of two input vague spatial objects, but only the Egg-Yolk model reuses the region connection calculus. In addition, VASA is the only one that considers the union of all components of the input vague spatial objects in the comparisons. To sum up, VASA is the more complete exact model studied, since it supports complex vague spatial data types for points, lines and regions, which are closed under operators for union “ \cup ”, “ \cap ”, and difference “ $-$ ” of vague spatial objects.

Dilo (2006), Dilo, By & Stein (2007), Verstraete, Hallez & Tré (2006), Schneider (2008), Tang, Kainz & Wang (2010), and Hazarika & Hazarika (2012) proposed fuzzy models that

adopt fuzzy logic to assign a membership to each component of an object and to describe fuzzy spatial data types, operators, and topological relationships. Differences among fuzzy models mainly arise due to the adoption of distinct topologies, which define the parts of vague spatial objects. Some fuzzy models do not provide data types closed under operations such as union, intersection and difference. The evaluation of topological predicates extends the 9-intersection model and its 3×3 matrix, either by using subsets provided by α -cuts, or fulfilling a matrix after the identification of intersections.

Table 2.3: Comparing exact models, fuzzy models and implementations for fuzzy models.

Model/Implementation	Class	Data Types	Operators	Relationships
Egg-Yolk	Exact Model	\diamond	-	RCC
RBB	Exact Model	\diamond	-	9IM
QMM	Exact Model	\cdot, \sim, \diamond	-	9IM
VASA	Exact Model	$\cdot, \approx, \diamond, \diamond$	$\cup, \cap, -$	9IM
Dilo (2006), Dilo, By & Stein (2007)	Fuzzy Model	$\cdot, \approx, \diamond, \diamond$	$\cup, \cap, -$	9IM
Verstraete, Hallez & Tré (2006)	Fuzzy Model	\diamond	\cup	9IM
Schneider (2008)	Fuzzy Model	$\cdot, \approx, \diamond, \diamond$	-	9IM
Tang, Kainz & Wang (2010)	Fuzzy Model	\diamond	\cup, \cap	9IM
Hazarika & Hazarika (2012)	Fuzzy Model	\diamond	-	9IM
Plateau Algebra	Fuzzy Impl.	$\cdot, \approx, \diamond, \diamond$	$\cup, \cap, -$	9IM
Lines with gradual transitions	Fuzzy Impl.	\sim	$\cup, \cap, -$	-
Bitmaps	Fuzzy Impl.	\diamond	\cup, \cap	-
TIN	Fuzzy Impl.	\diamond, \diamond	$\cup, \cap, -$	-
Fuzzy MBR	Fuzzy Impl.	\diamond	\cup	9IM

The main implementations for fuzzy models found in the literature were the plateau algebra (KANJILAL; LIU; SCHNEIDER, 2010; SCHNEIDER, 2014), lines with gradual transitions (DILO et al., 2006), bitmaps (VERSTRAETE et al., 2005; VERSTRAETE; TRÉ; HALLEZ, 2006), TIN (VERSTRAETE et al., 2005; DILO et al., 2006), and the fuzzy MBR (SOMODEVILLA; PETRY, 2003). They approximate a vague spatial object as pairs composed of *a portion of space* and an associated *membership value* that is either *precomputed* or *calculated*. Plateau algebra is the only implementation that supports complex data types for point, line and region. A line with gradual transitions provides a continuous outline, while a plateau line is discrete. Analogously, a TIN provides a continuous surface, while a plateau region, a bitmap and a fuzzy MBR are discrete. Furthermore, lines with gradual transitions and TINs maintain precomputed membership values for vertices and provide functions to calculate the membership values of the other points on the fly. Conversely, spatial plateau objects, bitmaps and fuzzy MBRs maintain precomputed membership values for their geometries, cells and rectangles, respectively. Operators such as union were described for most of the surveyed implementations for fuzzy models. However,

topological relationships between a pair of fuzzy spatial objects were not tackled.

Chapter 3

RELATED WORK

This chapter surveys existing work in the literature that are related to this thesis and involve either SDW design or spatial vagueness or both. Section 3.1 addresses both conceptual modeling and logical design of SDWs. Section 3.2 surveys spatial vagueness in SDWs. Section 3.3 tackles the physical design of SDWs and focuses on indices. Section 3.4 addresses indices for vague regions. Finally, Section 3.5 summarizes the main findings of this chapter and compares related work to the major contributions of this thesis.

3.1 Conceptual Modeling and Logical Design of Spatial Data Warehouses

The comprehension of existing conceptual models for SDWs is a prerequisite for the elaboration of a conceptual model for SDWs characterized by spatial vagueness. Section 3.1.1 addresses the MultiDim conceptual model introduced by Malinowski & Zimányi (2009) and improved by Vaisman & Zimányi (2014b). Section 3.1.2 tackles multidimensional modeling of SDWs using UML profiles from Pinet & Schneider (2010) and Boulil, Bimonte & Pinet (2015). The aforementioned conceptual models also enable the logical design of SDWs by providing mapping rules to transform a conceptual schema into a logical schema. Conversely, Section 3.1.3 tackles the elaboration of logical schemata disassociated from a conceptual schema by using the Spatial Data Warehouse Metamodel (CUZZOCREA; FIDALGO, 2012). Finally, Section 3.1.4 discusses and compares the surveyed related work.

3.1.1 The MultiDim Conceptual Model

MultiDim is a conceptual multidimensional model that enables the graphical representation of the concepts of a data cube, i.e. dimensions, hierarchies, fact and measures. In addition, pictograms represent spatial data types used to model attributes of dimensions and spatial measures, as well as topological relationships that establish topological constraints. The graphical notation of the MultiDim model is partially depicted in Figure 3.1 and, apart from pictograms, it resembles the symbols of the E-R model.

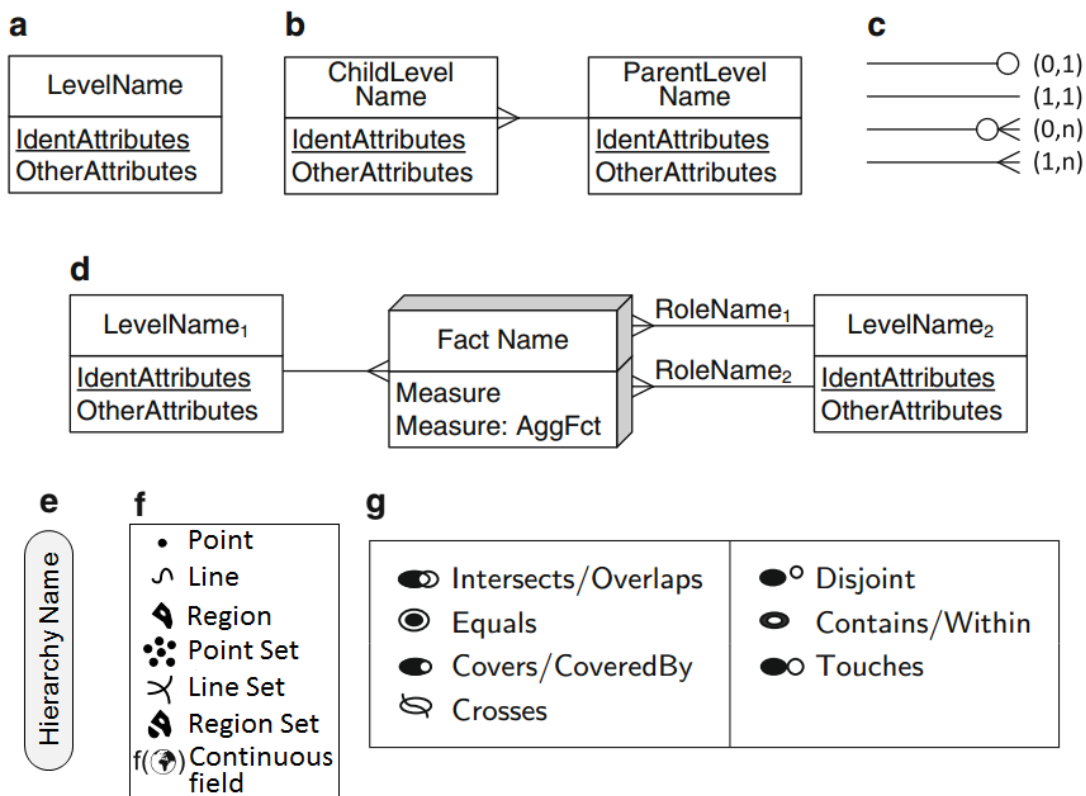


Figure 3.1: The graphical notation of the MultiDim conceptual model: (a) Level. (b) Hierarchy. (c) Cardinalities. (d) Fact with measures and related levels. (e) Hierarchy name. (f) Spatial data types. (g) Topological relationships. Adapted from Vaisman & Zimányi (2014a).

A schema encompasses a set of dimensions and a set of facts. A dimension comprises either one level or one or more hierarchies, while a hierarchy is a set of related levels. A level is equivalent to an entity type from the E-R model. Instances of a level are called members. As shown in Figure 3.1a, a level has a name, one or more attributes and at least one identifier. An identifier can be composed of one or several attributes. Every conventional attribute has a data type which is omitted for the sake of simplicity. Figure 3.2 exemplifies a schema with four dimensions and the fact .

The MultiDim model supports spatial data types and topological relationships defined by

the MADS model (PARENT; SPACCAPIETRA; ZIMÁNYI, 2006), as shown in Figure 3.1f-g. A spatial level is a level that refers to spatial data according to the application requirements. Every spatial level has one spatial data type assigned and the corresponding pictogram placed on the right of the level's name. The same notation is valid for spatial attributes. The MultiDim model also supports continuous fields. Figure 3.2 exemplifies the spatial levels Highway and Section with type Line Set, Segment with type Line, and State with type Region Set. County is a spatial level with type Region and also holds the spatial attribute Capital of type Point.

Given a pair of related levels in a hierarchy, the level with finest granularity is called *child* and the level with coarsest granularity is called *parent*, as shown in Figure 3.1b. The parent-child relationships assume one of the cardinalities depicted in Figure 3.1c. Every hierarchy has a name specified by a rounded rectangle as shown in Figure 3.1e. Figure 3.2 exemplifies the hierarchy HighwayStructure that relates the levels Segment, Section and Highway and the hierarchy Location involving the levels County and State.

Given a pair of spatial levels of a hierarchy, the topological relationship satisfied by their spatial members must be specified as a topological constraint. Then, one pictogram that denotes such topological relationship is placed close to the link representing the association between the levels. The pictogram is one of those shown in Figure 3.1g. Figure 3.2 exemplifies that states contain counties (as well as counties are located within states).

A fact relates several levels and its instances are called fact members. As illustrated in Figure 3.1d, a level can participate more than one time in a fact and play different roles. The relationship between a level and a fact also assume the cardinalities depicted in Figure 3.1c. A fact may contain measures and the aggregation function applied on a measure can be specified on the right of the measure's name, as shown in Figure 3.1d. If the aggregation function is omitted, SUM is assumed by default.

A spatial measure has a spatial data type assigned by placing one of the pictograms shown in Figure 3.1f on the right of the measure's name. The aggregation function of a spatial measure can also be specified, on the right of the pictogram. Otherwise, Union is assumed by default and denotes the geometric union. Figure 3.2 exemplifies the spatial measure CommonArea of type Line whose associated aggregation function has been omitted and is Union.

A *spatial fact* is a fact that relates several levels such that at least two of them are spatial levels. A spatial fact may introduce a topological constraint among members of the spatial levels related by the fact. Such topological constraint is specified by a pictogram placed inside the fact. The pictogram is one of those shown in Figure 3.1f. Figure 3.2 exemplifies that a segment intersects a county and that a county overlap a segment.

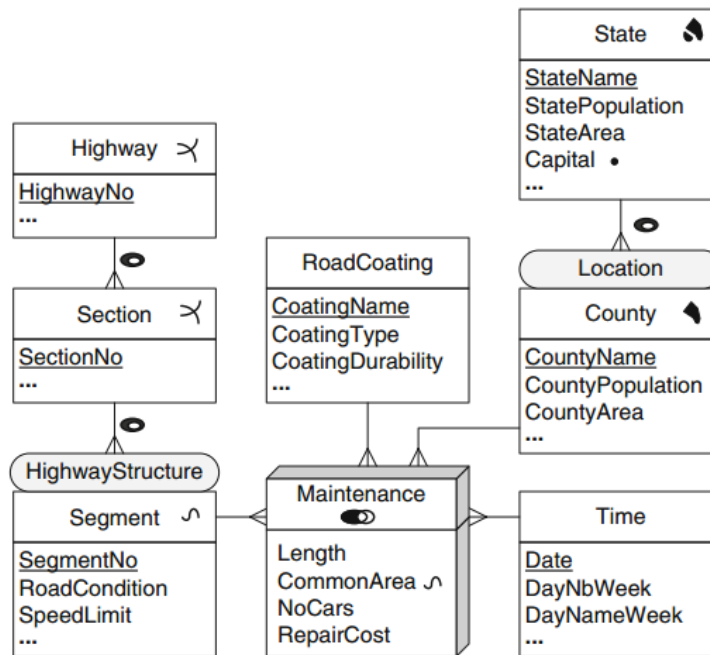


Figure 3.2: A SDW regarding the maintenance of highways modeled according to the MultiDim model (extracted from Vaisman & Zimányi (2014a)).

Mapping rules were provided to transform a conceptual schema of SDW into a relational logical schema of SDW. A spatial attribute is represented by a geometry column. A spatial level table contains all the attributes of the corresponding level and, specially, the spatial attribute as a geometry column. A spatial measure is represented by a geometry column in the fact table. Hierarchies are designed according to the cardinality of the relationships, i.e. 1:1, 1:N or M:N. Topological constraints imposed by hierarchies and by the spatial fact were implemented as triggers in the DBMS, according to the application addressed by the SDW.

3.1.2 UML Profiles

Pinet & Schneider (2010) proposed an UML profile for building data cubes of SDWs using the notation of an UML class diagram, whose metaclasses were customized mainly with the stereotypes *class of facts*, *class of members*, *measure*, *id* and *aggregating association*. A class of fact denotes a fact, while a measure is an attribute of a class of fact. A class of members is equivalent to a level. The stereotype *id* annotates the identifier of the class. An aggregating association plays the role of a hierarchy relationship between classes of members. The class of members can also assume the role of a class of facts, depending on the analysis required. Therefore, given a SDW schema with several cubes, the dimension of one cube can be the fact of another cube. OCL was used to specify constraints such as the allowed topological relationships among members (from a class of members) and measure values (from a class of

facts).

Boulil, Bimonte & Pinet (2015) elaborated an even more detailed UML profile for conceptual modeling of spatio-temporal DW and definition of constraints using OCL. The main details regarding SDW are summarized in the following and the support for temporality is omitted for the sake of simplicity. A hypercube is a package and is composed of at least one fact and one dimension. A fact is a class and has zero or more measures. A measure is a property and is described by its type. Aggregation levels, hierarchies and dimensions may be spatial or descriptive (thematic). A dimension is a package and is composed of at least one hierarchy. A hierarchy is a package and encompasses aggregation levels and aggregation relationships. An aggregation level is class composed of at least one dimensional attribute. Stereotypes customize UML metaclasses, as follows.

A dimensional attribute is either an identifier with the stereotype «IDAttribute», or descriptive with the stereotype «DescriptiveAttribute», or geometry with the stereotype «LevelGeometry». An identifier allows grouping when performing roll-up operations. A descriptive attribute allows selection when performing slice and dice operations and assumes an UML data type, e.g. Integer, float, or String. A geometry attribute represent geometric shapes of spatial dimension members and assume one spatial data type from the MADS model (PARENT; SPACCAPIETRA; ZIMÁNYI, 2006).

A dimensioning relationship with stereotype «DimRelationship» is an association between a fact and one aggregation level. Two aggregation levels are related through an aggregation relationship with stereotype «AggRelationship», such that one of the levels has a coarser granularity. Although Boulil, Bimonte & Pinet (2015) allowed constraints to be defined using OCL, they did not specify which constraints involved spatial data.

The SDW conceptual schema shown in Figure 3.3 was modeled using the UML profile from Boulil, Bimonte & Pinet (2015). It addresses the maintenance of highways and is equivalent to the one previously modeled according to the MultiDim model and shown in Figure 3.2. The only exception is the spatial attribute Capital in the spatial aggregation level State, which has not been associated to a stereotype because «LevelGeometry» does not reflect what a capital is for a state. In other words, Capital cannot be a «LevelGeometry» because the point of a capital does not adequately represent a state. In addition, the UML profile constrains that there is only one geometry attribute per spatial aggregation level, but «LevelGeometry» has already been assigned to StateGeo that denotes the territory of the state.

Boulil, Bimonte & Pinet (2015) implemented a tool that generates and executes SQL commands in the DBMS Oracle that create a star-schema or a snowflake schema to represent the

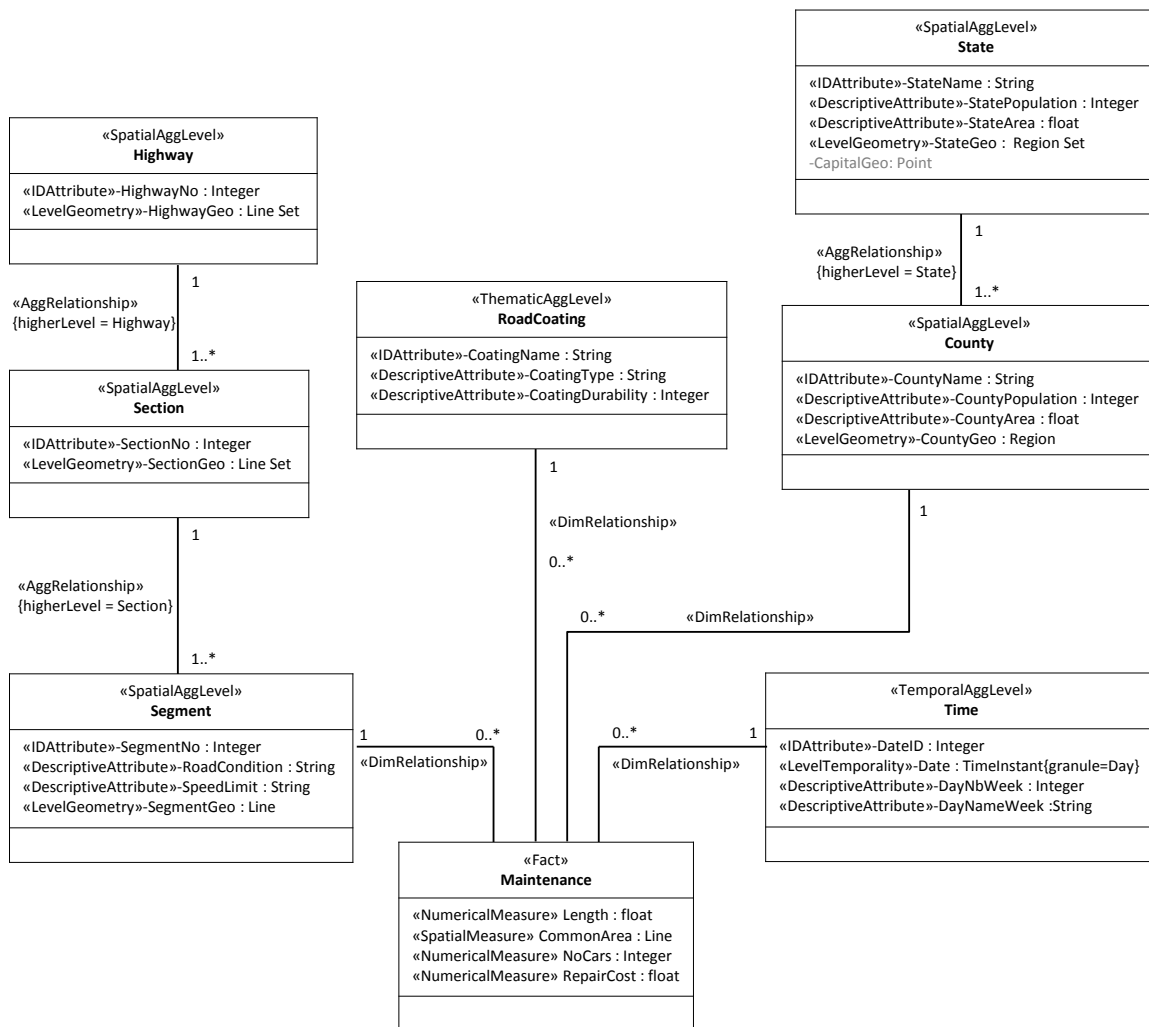


Figure 3.3: A SDW regarding the maintenance of highways modeled according to the UML profile proposed by Boulil, Bimonte & Pinet (2015).

conceptual schema described by an UML profile. Basically, they employ object-relational mapping to transform a transform a conceptual schema into a logical relational schema. In addition, the tool creates triggers that implement the constraints that were defined using OCL (BOULIL et al., 2010).

3.1.3 The Spatial Data Warehouse Metamodel

Cuzzocrea & Fidalgo (2012) defined the spatial data warehouse metamodel (SDWM) that allows the design of valid relational schemata for SDWs through an UML class diagram. They disassociated the relational logical design of SDW from the multidimensional conceptual design of SDW to enable support for techniques that are intrinsic of relational DWs, such as degenerated dimensions (KIMBALL; ROSS, 2002). Tables are either a dimension table, a bridge table or a fact table. A spatial attribute is implemented as a geometry column in a dimension table.

The metamodel allows spatial attributes to be normalized or shared among dimension tables. A spatial measure is implemented as a geometry column in a fact table. A bridge table is used for relationships with cardinality M:N. Tables and attributes are stereotyped with pictograms that differentiate fact tables from dimension tables, integer attributes from date attributes, and spatial attributes of type Point from spatial attributes of type Polygon, for example.

A CASE tool called SDWCASE was implemented to evaluate the correctness of SDWM. It provides a graphical user interface that allows the designer to edit the SDW schema. Furthermore, SDWCASE can validate the SDW schema and generate source code as input for PostgreSQL/PostGIS.

3.1.4 Discussion

In Sections 3.1.1 to 3.1.3, different approaches for conceptual modeling and logical design of SDW have been surveyed. Multidimensional modeling produces data cubes with spatial and non-spatial dimensions and measures, but do not offer graphical representations of the concepts, which could aid designers to build conceptual schemas of SDW. In this sense, graphical representations of the concepts of a data cube are enabled by the MultiDim model elaborated by Malinowski & Zimányi (2009) and Vaisman & Zimányi (2014b) and by UML profiles created by Pinet & Schneider (2010) and Boulil, Bimonte & Pinet (2015). Furthermore, Cuzzocrea & Fidalgo (2012) defined the SDWM metamodel for creating logical relational schemata for SDW using UML class diagrams and provided stereotypes and pictograms to differentiate tables and attributes.

A conceptual SDW schema created using the MultiDim model resembles an E-R diagram, while the conceptual SDW schema outlined using an UML profile is an UML class diagram. Also, a logical schema according to SDWM is also an UML class diagram. The MultiDim model represents a spatial level using a single pictogram indicating a spatial data type, while UML profiles utilize a stereotype above the class' name and adds an attribute with type geometry in the class. The MultiDim model allows more than one spatial attribute per spatial level and specifies the spatial data type with a pictogram, while the UML profiles do not allow more than one spatial attribute per spatial level. The SDWM metamodel also provides pictograms according to the spatial data type.

The following topological constraints were found in the surveyed work: (i) among members from distinct spatial levels related through a hierarchy; (ii) among members from distinct spatial levels related through a fact; and (iii) between members from a spatial level and measure values from a fact. The MultiDim model allows the graphical representation of the first and second

using pictograms. OCL is used by Pinet & Schneider (2010) to specify the third in their UML profile. Boulil, Bimonte & Pinet (2015) claimed that OCL can be used to write constraints, but did not specify which topological constraints involve spatial data. The SDWM metamodel did not address constraints.

A conceptual schema modeled according to the MultiDim model is transformed into a relational logical model by applying mapping rules that are analogous to ER to relational mapping. Conversely, UML profiles apply object-relational mapping to transform the UML class diagram into a relational logical schema. The SDWM metamodel does not provide mapping rules because it addresses the relational logical design.

3.2 Spatial Vagueness in Spatial Data Warehouses

The related work surveyed in the following sections tackle the design of SDW that admit the existence of vague spatial data. They propose different methods to handle spatial vagueness in SDWs. To the best of the author's knowledge, only the Fuzzy Spatial Data Warehouse (Fuzzy SDW) by Perez, Somodevilla & Pineda (2007, 2010) preceded the start of the doctoral research project that resulted in this thesis. On the other hand, during the development of such doctoral research project, the topic was also considered in the elaboration of the frameworks described by Jadidi et al. (2013, 2014) and the RADSOLAP method proposed by Edoh-Alove, Bimonte & Bédard (2014), Edoh-Alove et al. (2014). Section 3.2.1 addresses the Fuzzy SDW, Section 3.2.2 tackles the frameworks, Section 3.2.3 stands for the RADSOLAP method, and Section 3.2.4 discuss and compare those work.

3.2.1 The Fuzzy Spatial Data Warehouse

The Fuzzy SDW focused on the logical design of relational SDWs, considered fuzzy regions implemented as fuzzy MBRs, and was experimented in volcanic crisis management. Its design method: (i) identifies crisp spatial objects of distinct granularities that are related; (ii) creates a fuzzy region for related crisp spatial objects; (iii) implements each fuzzy region as a fuzzy MBR; (iv) creates a table and, for each fuzzy MBR, stores the surrogate key and the membership value of every α MBR-cut; (v) adds a column to maintain linguistic labels that are assigned to denote fuzziness; and (vi) drops geometry columns.

In order to exemplify the design method, consider sample points within studied areas, which are located close to a volcano. Figure 3.4a shows sample points and a studied area, while Fig-

ure 3.4b depicts the relational schema denoting the hierarchy among sample points and studied areas. The potential extent of lava flow is a fuzzy region that covers sample points and is delimited by the studied area. Different from crisp spatial data types, a fuzzy region can both outline the extent of the lava flow and estimate the possibility degree of a given location being affected. Then, the lava flow is represented by the fuzzy MBR shown in Figure 3.4c.

Thereafter, a table *LavaFlow* is created as shown in Figure 3.4d. For each α MBR-cut of the fuzzy MBR, a surrogate key value is generated and stored together with rectangle's membership value, in the columns *LavaFlowPK* and *PossibilityDegree*, respectively. The column *PossibilityLabel* is added to *LavaFlow*. For each row in *LavaFlow*, a label such as *low*, *moderate* or *high* is assigned to *PossibilityLabel* based on the value in *PossibilityDegree*. Then, both *SamplePoint* and *StudiedArea* have their geometry columns dropped. The definitive schema is that depicted in Figure 3.4d.

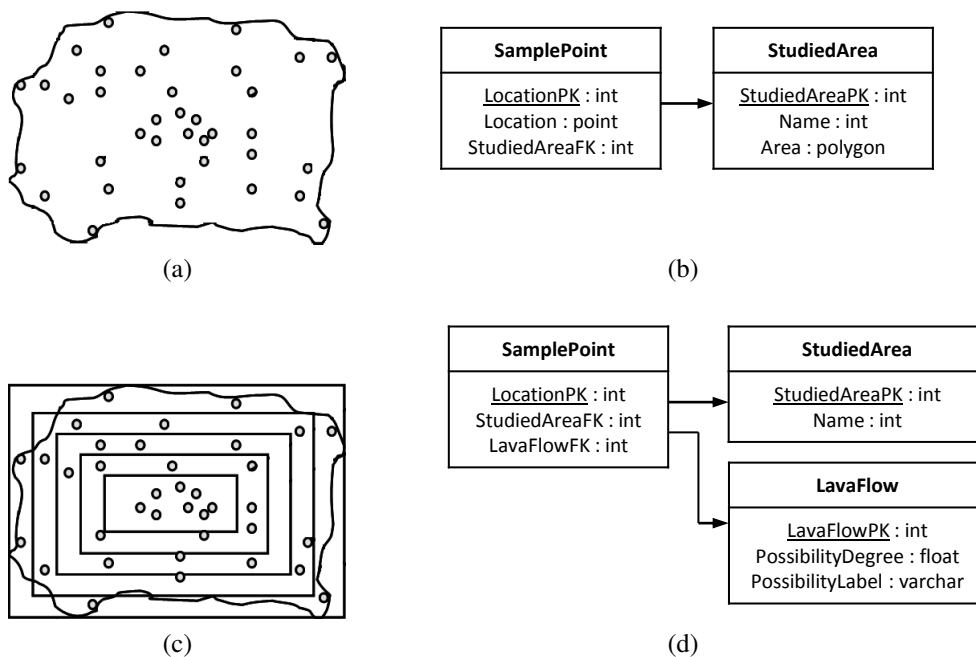


Figure 3.4: The logical design according to the Fuzzy SDW (adapted from Somodevilla & Petry (2003), Perez, Somodevilla & Pineda (2007)). (a) Sample points and a studied area. (b) Original schema. (c) The fuzzy MBR. (d) Modified schema.

Furthermore, in the Fuzzy SDW, geometry columns are dropped and replaced by results of spatial relationships that are held by members of distinct spatial dimensions. In addition, linguistic labels are assigned to denote fuzziness, instead of fuzzy regions. The goal is to describe the semantics of implicit spatial relationships that are held by crisp spatial objects. For example, distance was considered as a metric spatial relationship, distances among cities and evacuation routes were calculated, and results were stored into a separate table. Then, linguistic labels such

as *very far*, *far*, *near*, and *very near* were assigned according to the calculated distance. These labels described the distance between cities and routes. Finally, polygons of cities and lines of evacuation routes were eliminated.

3.2.2 Conceptual Frameworks for Risk Assessment

An analytical conceptual framework was proposed for coastal erosion risk assessment (CERA). Such framework added steps to conceptual modeling of SDW, such as the identification of vulnerability indexes for elements exposed to erosion. Using the framework, a conceptual schema of SDW for CERA was designed as an UML class diagram, which is partially illustrated in Figure 3.5.

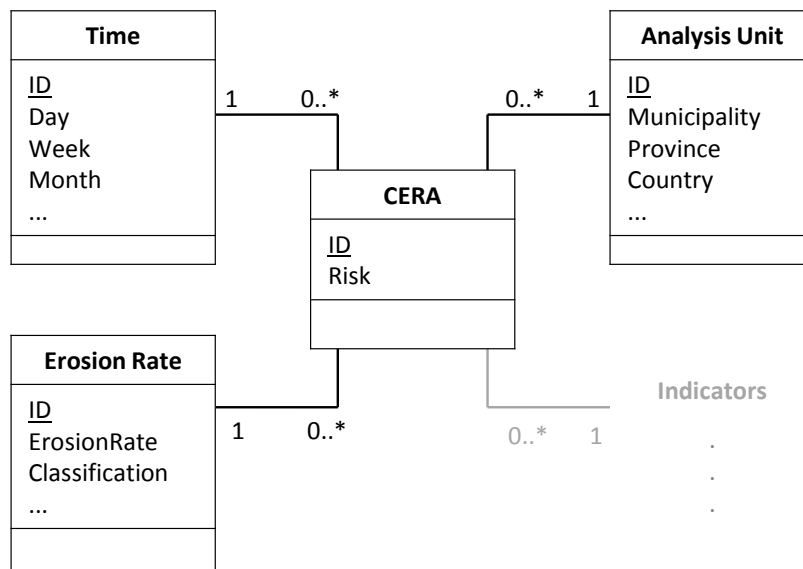


Figure 3.5: A conceptual schema of SDW for CERA (adapted from Jadidi et al. (2013)).

The dimension Time is temporal. The spatial dimension Analysis Unit denotes administrative units and encompasses the hierarchy Country \preceq Province \preceq Municipality. Spatial attributes can be implemented using tessellations. The dimension Erosion Rate is a vulnerability index intrinsic of CERA. It classifies a erosion as *very low* (1), *low* (2), *average* (3), *high* (4), or *very high* (5). A vulnerability index is based on expert-knowledge and on interests of stakeholders and decision makers through multiple indicators. The annotation “Indicators” refer to omitted dimensions denoting other categories of vulnerability indexes. For instance, a socio-economical vulnerability index is land occupation, which could be represented by a dimension to classify a risk level as *low* for rural zones and *high* for urban zones. The fact CERA holds the numeric measure Risk. Its values in [1,5] are assigned by an equation whose input are the referenced values of Erosion Rate and other dimensions denoting indicators.

Thereafter, a conceptual framework for fuzzy representation of risk zones was proposed and considered the conceptual schema of SDW shown in Figure 3.5. The method: (i) generates regular tessellations to represent Erosion Rate and the dimensions denoting indicators; (ii) determines the membership functions and the respective membership value for each cell of a grid within each indicator to obtain fuzzy regions; and (iii) calculate the overall risk for a given region, by aggregating multiple layers of grids.

Regular tessellations are produced using a GIS such that the size of the cell depends on required scales and available information. For each class of an indicator, one layer of the grid is created and contains fuzzy regions. For instance, five layers are created for the dimension Erosion Rate and two would be created for the dimension of land occupation. As a result, the layer denoting very low erosion rate contains fuzzy regions that indicate where such vulnerability exists.

Membership functions are derived from vulnerability index classification and membership values are assigned to each cell of the grid, considering multiple layers. The membership value is assigned to the center of each cell as shown in Figure 3.6a and scattered as a Gaussian function toward the outside as shown in Figure 3.6b. For instance, considering the grid for erosion rate and the layer for very low erosion rate, the value 0.5 meters per year yields a membership value of 0.2, which is assigned to the center of the corresponding cell. After assigning membership values to each cell, the grid contains fuzzy regions as shown in Figure 3.6b.

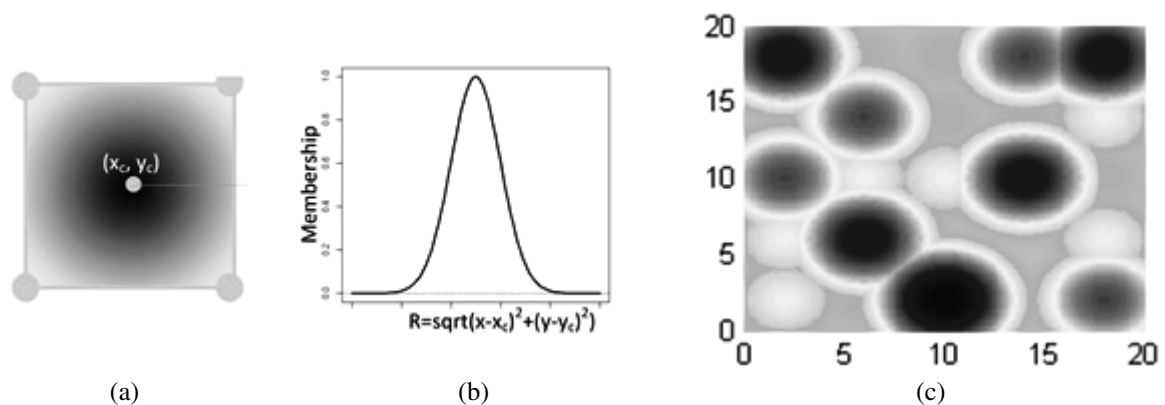


Figure 3.6: Creating fuzzy regions for an arbitrary indicator (adapted from Jadidi et al. (2014)). (a) A magnified cell and its center. (b) The Gaussian function. (c) A grid and its vague regions defined over multiple cells.

The calculation of the overall risk for a given region requires the aggregation of multiple layers of information. Rules for aggregation are defined according to the risk equation and the priority of stakeholders and authorities of the region under study. An example of a fuzzy rule is “if hazard to hydrological network is very high and hazard to protection structure is very high

and distance to vulnerable object is very high and erosion rate is very high, then use fuzzy union to aggregate fuzzy regions for obtaining the overall risk". Aggregation can be performed using fuzzy union, fuzzy intersection, mean, or weighted mean.

The SDW conceptual schema shown in Figure 3.5 was elaborated according to the analytical conceptual framework. However, such schema does not consider vague spatial data in dimensions' attributes and neither as measures in the fact. In addition, the framework did not provide the transformation of the conceptual SDW schema into a logical schema.

The conceptual framework for fuzzy representation of risk zones assumed the existence of the SDW conceptual schema shown in Figure 3.5 in order to generate tessellations, design fuzzy regions, and calculate overall risks. Nevertheless, the validation of such framework did not utilize an implemented SDW, since the input to generate tessellations and to design fuzzy regions were not the SDW's dimensions representing indicators. In addition, the output comprising calculated overall risks was not stored into a SDW. In fact, the implementation of the proposed method in a multidimensional context was considered a future work.

3.2.3 The RADSOLAP Method

The RADSOLAP method supplies several prototypes of the SDW conceptual schema to users. Each prototype is tailored according to users' tolerance levels to vague spatial data. If the tolerance is low, new prototypes have vague spatial data gradually eliminated from the SDW schema. Otherwise, vague spatial data are maintained. The conceptual schema of SDW is an UML class diagram, while the logical schemata is obtained through object-relational mapping. The RADTool implements the RADSOLAP method and allow designers to automatically and incrementally design conceptual schemata, which are deployed on the DBMS, loaded with sample data and accessed by a SOLAP tool in order to test. The RADSOLAP method uses simple geometry types to represent and implement vague spatial data types, i.e. Point, Line, and Polygon.

A case study regarding fertilizer spreading considered each spreading area as being composed of a part that certainly belong to it and a part that possibly belong to it, like a broad boundary. For instance consider the vague region S1 shown in Figure 3.7a. A relational schema of SDW has been provided for such application and a sample of its fact table is shown in Figure 3.7b. Both rows refer to the flow of fertilizers spread over the vague region S1. For each row, one column of the measure Flow provides a value for the part that certainly belongs to S1, while the other column of the measure provides a value for the part that possibly belongs to S1.

There is a risk of misinterpretation of the values of the measures that denote flow, because

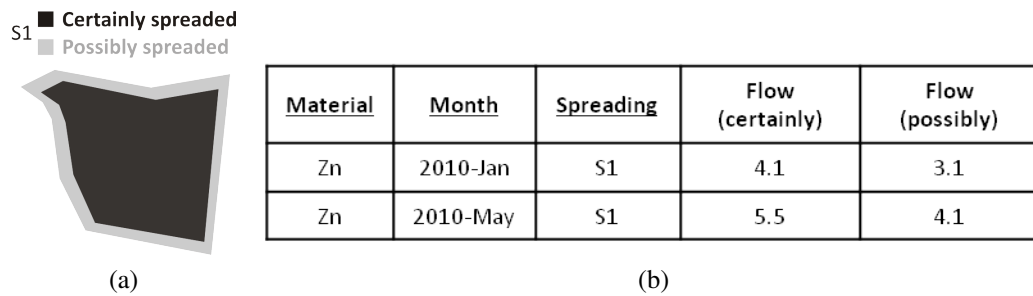


Figure 3.7: Flow of fertilizer by material by month by spreading region. (a) A spreading region. (b) The SDW's fact table.

for each row of the fact table there are two possible values for the same measure. One value is for the spreading over the part that certainly belong to the spread area. The other value is for the part that possibly belongs to the spread area. Such risk of misinterpretation was defined as a risk related to vague spatial data implemented as complex geometries in dimensions of the SDW.

Furthermore, there is a hierarchy $\text{Watershed} \preceq \text{Spreading Area}$. The cardinality is 1:N and, in most of the cases, a vague region of a spreading area is fully contained by a crisp region of a watershed. Otherwise, the split method proposed by Malinowski & Zimányi (2009) aggregates measure values. However, for a vague region, a pair of measure values are defined by the pair of columns in the fact table. Then, the use of only one of these values for aggregation can under evaluate or over evaluate the total regarding the watershed related to a spreading area. The risk of aggregation concerns measures associated to vague regions.

Decision-makers informally identify and assess risks of misinterpretation and risks of aggregation for each prototype of the SDW schema. They report a tolerance level. Actions may modify the SDW schema (e.g. deletion of level), the aggregation function utilized, or the visualization technique employed (e.g. coloring cells of the pivot table). The tolerance level zero indicates an unacceptable risk and causes the elimination of vague spatial data from the SDW schema, the removal of any attributes related to vague spatial data, and the replacement of complex geometries by simple geometries. In the aforementioned example, the level containing the vague spatial attribute of spreading areas is eliminated, the fact table references the level containing crisp regions of watersheds, and the measure in the fact table is transformed into a single column rather of a pair of columns.

Nevertheless, initiatives to be taken whether tolerance level is 1 or 2 were not clarified. If tolerance level is 3, then a totally acceptable risk is assumed and vague spatial data can remain in the SDW schema. However, in contrast with the RADSOLAP method that prioritizes simple geometries, spreading areas cannot be designed using simple polygons, as shown in Figure 3.7a.

3.2.4 Discussion

A few existing work in the literature address the design of SDW and admit the existence of vague spatial data. The Fuzzy SDW described by Perez, Somodevilla & Pineda (2007, 2010) identified fuzzy regions by relating crisp spatial objects of distinct granularities from a relational SDW. Jadidi et al. (2013) proposed a framework and created a conceptual schema of SDW for coastal erosion risk assessment. The schema was reused by (JADIDI et al., 2014) to propose a framework that guides the creation of fuzzy regions for dimensions and the overlay of fuzzy regions to obtain measure values. However, the Fuzzy SDW and the cited frameworks did not define attributes in the SDW schema for the fuzzy regions and did not store fuzzy regions in the SDW. As a result, they did not enable querying fuzzy regions maintained by the SDW.

Conversely, Edoh-Alove, Bimonte & Bédard (2014) and Edoh-Alove et al. (2014) admitted that vague spatial attributes may exist in dimensions of the SDW. Their RADSOLAP method provided users with several prototypes of the SDW conceptual schema according to users' tolerance levels to vague spatial data. If the tolerance is low, new prototypes have vague spatial data gradually eliminated from the SDW schema. The RADSOLAP method uses simple geometry types to represent and implement vague spatial data types, i.e. Point, Line, and Polygon.

Regarding the SDW used as case study for RADSOLAP, it is noteworthy that: (i) the level table containing a vague spatial attribute implemented as one column of type MultiPolygon, was designed according to the logical design proposed by Siqueira et al. (2012a), which is presently described in Section 5.2.1; and (ii) the hierarchy involving a finer level with a vague spatial attribute and a coarser level with a crisp spatial attribute were also tackled by Siqueira et al. (2012a, 2014) and is presently discussed in Sections 4.3, 4.9.3 and 5.5.

Also regarding the SDW used as case study for RADSOLAP, the numeric measure designed as a pair of columns is substantially different from the vague spatial fact described by Siqueira et al. (2014) and presently detailed in Sections 4.4.4 and 5.7. The vague spatial fact does not constrain the assignment of one measure value to each of the two elements of a vague spatial object. Rather, since it considers vague spatial objects may be composed of several elements instead of only two, it allows assigning partial measure values to each element of a vague spatial object. Moreover, the vague spatial fact does not implement measures as two columns such that each column refer to one of the two parts of a vague spatial object Rather than adopting one column for each element of a vague spatial object, only one column is used.

3.3 Indices for Spatial Data Warehouses

The comprehension of existing indices for SDWs precedes the elaboration of an index for vague SDWs. The following sections describe two distinct indices developed for SDW: the aggregate R-tree (aR-tree) (PAPADIAS et al., 2001) that is addressed in Section 3.3.1 and the Spatial Bitmap Index (SB-index) (SIQUEIRA, 2009; SIQUEIRA et al., 2009, 2012b) that is tackled in Section 3.3.2. Finally, Section 3.3 discusses and compares the surveyed related work.

3.3.1 aR-tree

The aR-tree reuses the R-tree's data structure and space partitioning method. An entry of a leaf node maintains: (i) a key value of a spatial object stored in the SDW; (ii) the MBR of that spatial object; (iii) a pointer to a multidimensional array that details the values of a measure associated to that spatial object according to values of the attributes from other dimensions of the SDW; and (iv) the value of the measure obtained by applying the aggregation function over the values of the multidimensional array. An entry of a non-leaf node holds a pointer to the child node, the MBR encompassing all MBRs of the child node, a pointer to a multidimensional array containing values obtained from applying the aggregation function over the multidimensional arrays pointed to by the entries of the child node, the value of the measure obtained by applying the aggregation function over the values held by the child nodes.

To process an intersection range query, the tree is traversed top-down from the root to the leaves. For all entries visited, one of the following conditions applies: (i) the MBR of the entry is disjoint from the spatial query window and, therefore, the node pointed to by the entry must not be visited; (ii) the MBR of the entry is within the spatial query window and, consequently, the entry has all the relevant values of the measure; (iii) the MBR of the entry intersects the spatial query window and, thus, the child node is recursively followed. Note that for the containment range query only the conditions (i) or (ii) may apply. It is also noteworthy that when condition (ii) applies, it prunes the traversal of the tree, avoid the search until the leaf nodes and prevent the costly refinement step. The same algorithm is used to process intersection range queries and containment range queries.

Consider the SDW schema shown in Figure 2.1, the cities of customers shown in Figure 3.8a, and the SDW data shown in Figure 3.8b such that CityFK=11 refers to the city whose MBR is r_{11} in Figure 3.8a and so forth. An aR-tree is built for the SDW data shown in Figure 3.8b such that each entry points to a bi-dimensional array like the one shown in Figure 3.8c and assigns values of the measure Revenue to each pair of Brand and Year. The bi-dimensional

array has $|\text{Brand}| \times |\text{Year}|$ entries, i.e. the multiplication of the cardinalities. The aR-tree applies the space partitioning method of the R-tree over the MBRs as shown in Figure 3.8a and builds the data structure shown in Figure 3.8d.

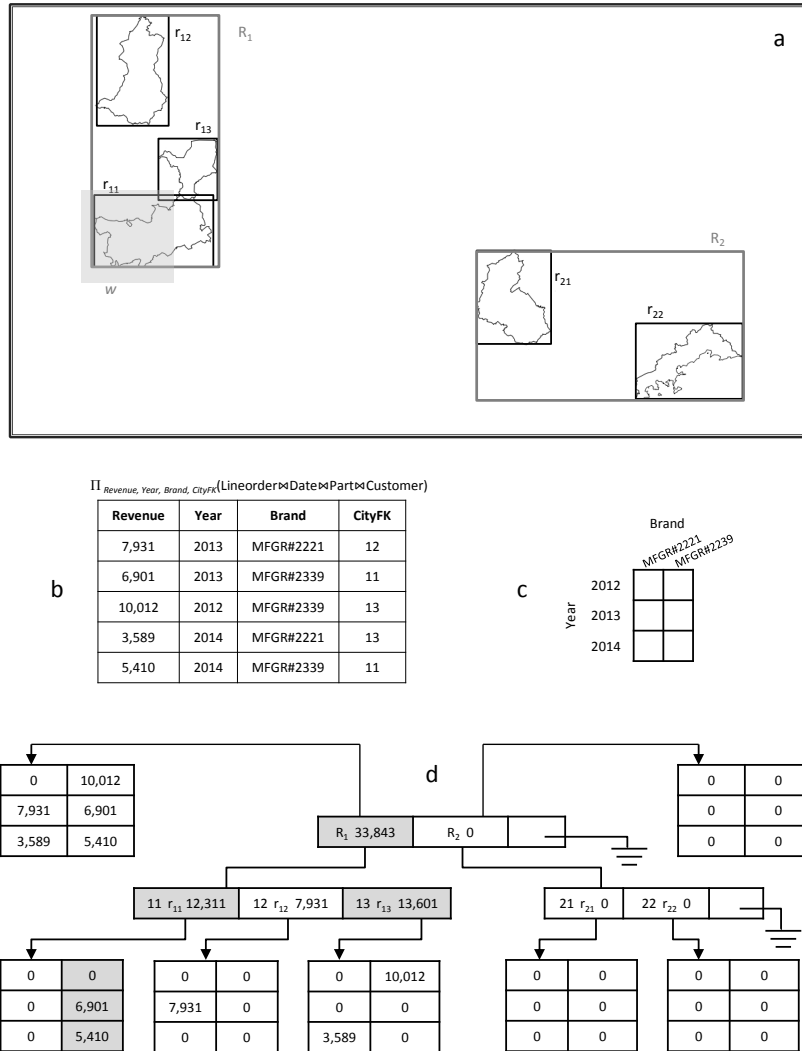


Figure 3.8: An example of aR-tree. (a) Cities, MBRs and a spatial query window. (b) SDW data. (c) Structure of the bi-dimensional array. (d) Data structure of the aR-tree.

The aR-tree depicted in Figure 3.8d is used to process the query described in Listing 2.1, considering the spatial query window w shown in Figure 3.8a, as follows. The root node is accessed and the first entry tested is the one holding the MBR R_1 , which intersects w . Thus, the child node is accessed and the MBR r_{11} of the first entry is tested against w . Since the intersection is true, the city with key value 11 becomes a candidate and is processed in the refinement step. As the corresponding city intersects w , as shown in Figure 3.8a, it is an answer of the spatial predicate. Then, the multidimensional array pointed to by the entry is accessed. Only the values where Brand='MFGR#2239' are added to the result set, i.e. those from the right column. Afterwards, the search continues in the tree and the entry holding the MBR R_{12} is

tested against w , but w does not intersect R_{12} . Further, the entry holding the MBR R_{13} is tested against w . Since they intersect, the city with key value 13 becomes a candidate and is processed in the refinement step, which yields that the city and w are disjoint as shown in Figure 3.8a. Since all the entries of the node were tested, the search continues in the root node, and the entry holding the MBR R_2 is tested against w . As they are disjoint, the traversal stops. Finally, the result set is sorted and then returned.

3.3.2 SB-index

The SB-index is an array of pairs (Key, MBR) sorted in ascending order of key values, whose i -th entry points to the i -th bit-vector of a bitmap join index. The SB-index is built by extracting the primary key values and the geometries of a table in the SDW and after by creating a bitmap join index for the key values. Thus, each bit-vector specifies the tuples of the fact table referencing the spatial object identified by a key value.

The algorithm for query processing of the SB-index is summarized as follows, considering the intersection range query (IRQ) and the containment range query (CRQ) as spatial predicates. The filter step comprises a sequential scan that iterates reading each disk page of the index file and copying to the main memory. For each entry, the spatial predicate is tested against the MBR. If the test yields true and the spatial predicate is an IRQ, the corresponding key value is added to the collection of candidates. If the test yields true, but the spatial predicate is a CRQ, the corresponding key value is added to the collection of answers. If candidates were collected, there is a refinement step that tests the original geometry of each candidate against the spatial predicate to identify answers. After collecting answers, a key matching uses the answers to replace the spatial predicate by a conventional predicate. After the replacement, the rewritten query is processed accessing bitmap join indices.

The spatial filter performs a fixed number of disk accesses since it is a sequential scan that consecutively reads all the pages of the index file. The refinement step requires accessing the DBMS to fetch complex geometries and, therefore, is more costly than the filter step. The access to bitmap join indices avoids joining huge tables and accesses strictly the indices to perform conventional predicates, aggregation and sorting.

A SB-index is built for table City of the SDW schema shown in Figure 2.1, which comprises the cities shown in Figure 3.9a. The built data structure is depicted in Figure 3.9d, considering that CityFK=11 refers to the city whose MBR is r_{11} in Figure 3.9a and so forth. Each entry of the SB-index sequential file holds a key value and a city's MBR as well as points to a bit-vector whose bits indicate tuples of the fact table referencing the indexed city. Besides, bitmap

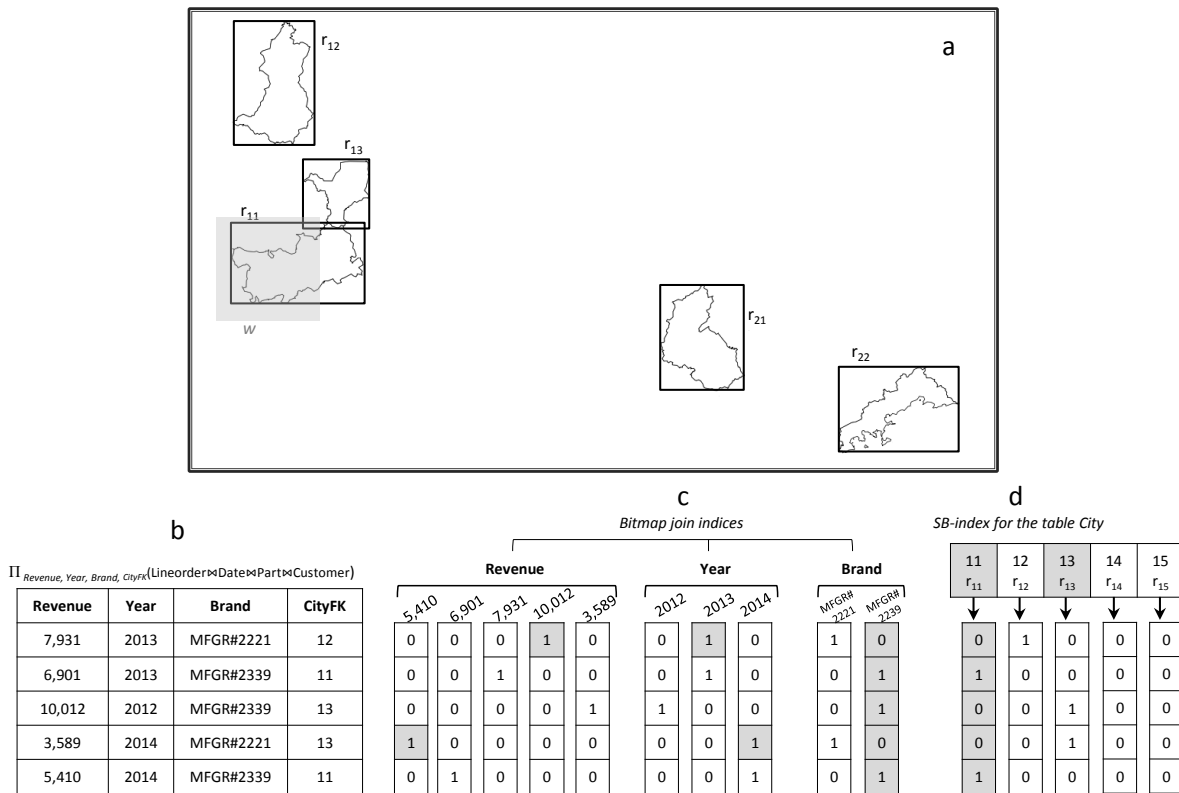


Figure 3.9: An example of SB-index. (a) Cities, their MBRs and a spatial query window. (b) SDW data. (c) Bitmap join indices. (d) Data structure of the SB-index.

join indices are built for the SDW data shown in Figure 3.9b and the bit-vectors are shown in Figure 3.9c.

The SB-index depicted in Figure 3.9d is used to process the query described in Listing 2.1, considering the spatial query window w shown in Figure 3.9a, as follows. A sequential scan is performed on the SB-index’ sequential file. The MBR r_{11} of the first entry is tested against w and, since they intersect, the key value 11 is added to the collection of candidates. Further, the MBR r_{12} of the second entry is tested against w and, since they are disjoint, the entry is skipped. Further, the MBR r_{13} of the third entry is tested against w and, since they intersect, the key value 13 is added to the collection of candidates. Subsequently, the MBRs of the fourth and fifth entries of the sequential file are tested against w and, since they are disjoint, the entries are skipped.

After the sequential scan, the collection of candidates has the key values 11 and 13. In the refinement step, the geometries with these key values are fetched in table City, but only the city with key value 11 really intersects the spatial query window w , as shown in Figure 3.9a. Then, the key value is added to the set of answers.

After the refinement step, the collection of answers has the key value 11, which is used to

replace the spatial predicate “ST.Intersects(...)” by the conventional predicate “CityFK = 11” in the query show in Listing 2.1. The rewritten query is then processed by the SB-index’ bitmap join indices together with bitmap join indices built for the other attributes of the query. Only the SB-index’ bit-vector where CityPK=11 is accessed to execute a bit-wise AND with the bit-vector for Brand=‘MFGR#2239’. As a result, only the first and the fourth entries of the bit-vectors for Revenue and Year, where there is a bit 1, must be aggregated. The result set is sorted and then returned. The SB-index was developed also for SDW schemata with redundancy of spatial data in tables (SIQUEIRA et al., 2008, 2009).

3.3.3 Discussion

The aR-tree has a hierarchical data structure and a tree-based search in the filter step comparable to the R-tree’s, while the SB-index has a sequential data structure and a sequential search in the filter step. They use a single conservative approximation for crisp spatial data: the MBR (minimum bounding rectangle). The aR-tree’s tree-based search can prune the traversal and avoid unnecessary disk accesses specially if the spatial query window is not very large and the input MBRs do not have overlap one each other with a high rate. In such conditions, the traversal does not reach many leaf-nodes and is able to provide answers by accessing aggregated multidimensional arrays pointed to by entries in non-leaf nodes. The SB-index’ sequential scan has a fixed cost and often performs unnecessary disk accesses. On the other hand, it is not affected by a large spatial query window and neither by overlapping among MBRs. While the aR-tree is able to identify answers for intersection range queries and containment range queries already in the filter step, the SB-index is only able to identify answers for containment range queries in the filter step.

The refinement step of the aR-tree is required if an entry of a leaf-node has an MBR that is considered a candidate. Then, the refinement step tends to be less costly as more answers of the spatial predicate are identified when accessing non-leaf nodes, e.g. when the MBR of an entry in a non-leaf node is within the spatial query window. The refinement step of the SB-index happens after the filter step identified candidates. Finally, conventional predicates, aggregation and sorting are processed by the aR-tree manipulating multidimensional arrays, while the SB-index reuse bitmap join indices to process them.

3.4 Indices for Vague Regions

The concern about the performance to process queries against vague spatial data, due to the complexity of vague spatial objects, has motivated the development of indices. This section surveys related work that describe indices for vague regions. Two main indices have been found in the literature, such that Section 3.4.1 summarizes the Vague R-tree designed for simple vague regions (PETRY; LADNER; SOMODEVILLA, 2007), while Section 3.4.1 addresses the Fuzzy MBR R-tree designed for simple fuzzy regions (SOMODEVILLA; PETRY, 2004; PETRY; LADNER; SOMODEVILLA, 2007). Finally, Section 3.3 discusses and compares the surveyed related work. Section 3.4.3 discusses and compares the surveyed related work. Existing work in the literature that focus on probabilistic models and continuous fields are not surveyed (TAO et al., 2005; KALASHNIKOV et al., 2006; LI et al., 2007; ZINN; BOSCH; GERTZ, 2007; YUEN et al., 2010).

3.4.1 Vague R-tree

The vague R-tree is an index for processing point queries against vague regions. A point query can be interpreted as a range query such that the vertices of the rectangular query window are all the same. The vague R-tree considers a pair of MBRs for each vague region. I is the MBR that approximates the kernel (using VASA's nomenclature) and is called inner MBR. O is the MBR that approximates the vague region and is called outer MBR. Figure 3.10a shows the pairs of MBRs for the vague regions A and B.

The vague R-tree extends the R-tree by using the pair of MBRs instead of the well-known MBR. For the sake of simplicity, the size in bytes of an entry of Vague R-tree is two times the size in bytes of an entry of R-tree. Considering the node capacity $M = 2$ for the R-tree, then a single entry of Vague R-tree is supported per node. Every entry in a non-leaf node maintains an inner MBR, an outer MBR and a pair of pointers to subtrees. The inner MBR circumscribes the inner MBRs of the subtree. Analogously, the outer MBR circumscribes the outer MBRs of the subtree. Figure 3.10b shows a pair of MBR such that the inner MBR I_{AB} circumscribes the inner MBRs I_A and I_B , while the outer MBR O_{AB} circumscribes the outer MBRs O_A and O_B .

A leaf node has strictly inner MBRs or outer MBRs. If it has inner MBRs, these are circumscribed by the inner MBR of the parent node. Conversely, if it has outer MBRs, these are circumscribed by the outer MBR of the parent node. Figure 3.10c depicts the data structure of the Vague R-tree for $M = 2$ and the pair of MBRs shown in Figure 3.10b. Point queries are supported and exemplified in the following.

A query asks to identify to which vague region the point p_1 belongs, as shown in Fig-

ure 3.10b. Traversing the index, p_1 intersects O_{AB} but is disjoint from I_{AB} . Then, the leaf node containing outer MBRs is accessed, but p_1 is disjoint from both O_A and O_B . The answer for the query is that p_1 does not belong to any vague region.

Other query asks to identify to which vague region the point p_2 belongs. Traversing the index, p_2 intersects both O_{AB} and I_{AB} . Then, the leaf node containing inner MBRs is accessed, but p_2 is disjoint from both I_A and I_B . The answer for the query is that p_2 does not belong to any vague region.

Finally, another query asks to determine where is the point p_3 . Traversing the index, p_3 intersects both O_{AB} and I_{AB} . Then, the leaf node containing inner MBRs is accessed, but p_3 is disjoint from both I_A and I_B . Further, the leaf node containing outer MBRs is accessed and p_3 intersects O_B . The answer for the query is that p_3 *possibly* belongs to the vague region B.

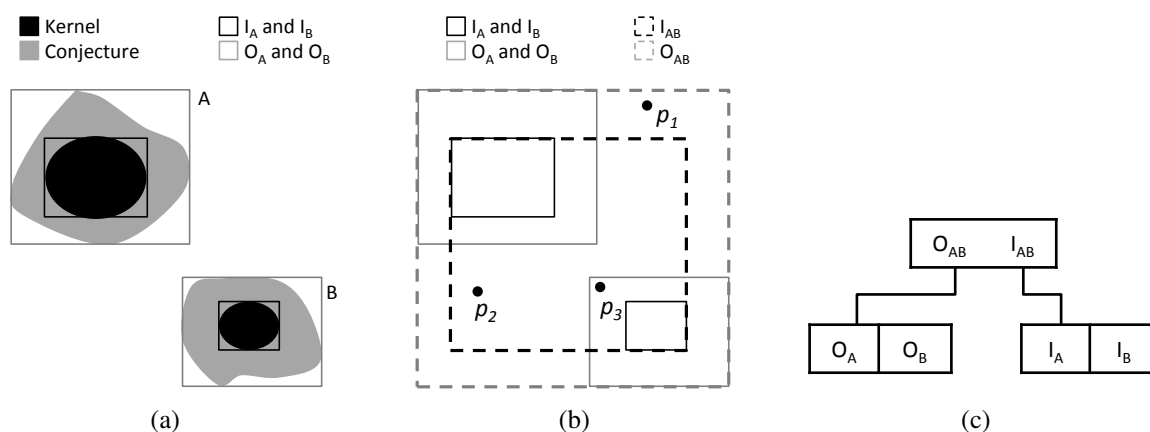


Figure 3.10: An example of vague R-tree. (a) Vague regions and their pair of MBRs. (b) Vague R-tree and three point queries. (c) Vague R-tree's data structure.

Although the Vague R-tree supports point queries it does not have a detailed query processing algorithm. Details are also missing about the necessity of a refinement step against kernels and vague regions to conclude the resolution of the spatial predicate. Besides, the Vague R-tree was not assessed through a performance evaluation.

3.4.2 FMBR R-tree

The Fuzzy MBR R-tree, or simply FMBR R-tree, is an index for searching crisp spatial objects located within a fuzzy region. The data structure of a Fuzzy MBR R-tree built on a fuzzy region encompasses: (i) the list of α MBR-cuts obtained from the fuzzy MBR built on a fuzzy region; and (ii) a R^* -tree for each α MBR-cut. The list is sorted by ascending membership value. Each R^* -tree indexes crisp spatial objects located within a given α MBR-cut.

Figure 3.11a depicts the frontier of a fuzzy region, points within the fuzzy region, and the FMBR R-tree built for the fuzzy region and the points. The fuzzy MBR is composed of the following α MBR-cuts: α MBR₁, α MBR₂, α MBR₃, α MBR₄, and α MBR₅, such that α MBR₁ is the inscribed rectangle corresponding to the core where the membership value is 1.0 and α MBR₅ is the circumscribing rectangle where the membership values are the lowest. For each α MBR-cut, a R*-tree is built using the points within the fuzzy region as input. The data structure of the FMBR R-tree is shown in Figure 3.11b and comprises the sorted list of α MBR-cuts in ascending order of membership values, such that each entry of the list points to a R*-tree.

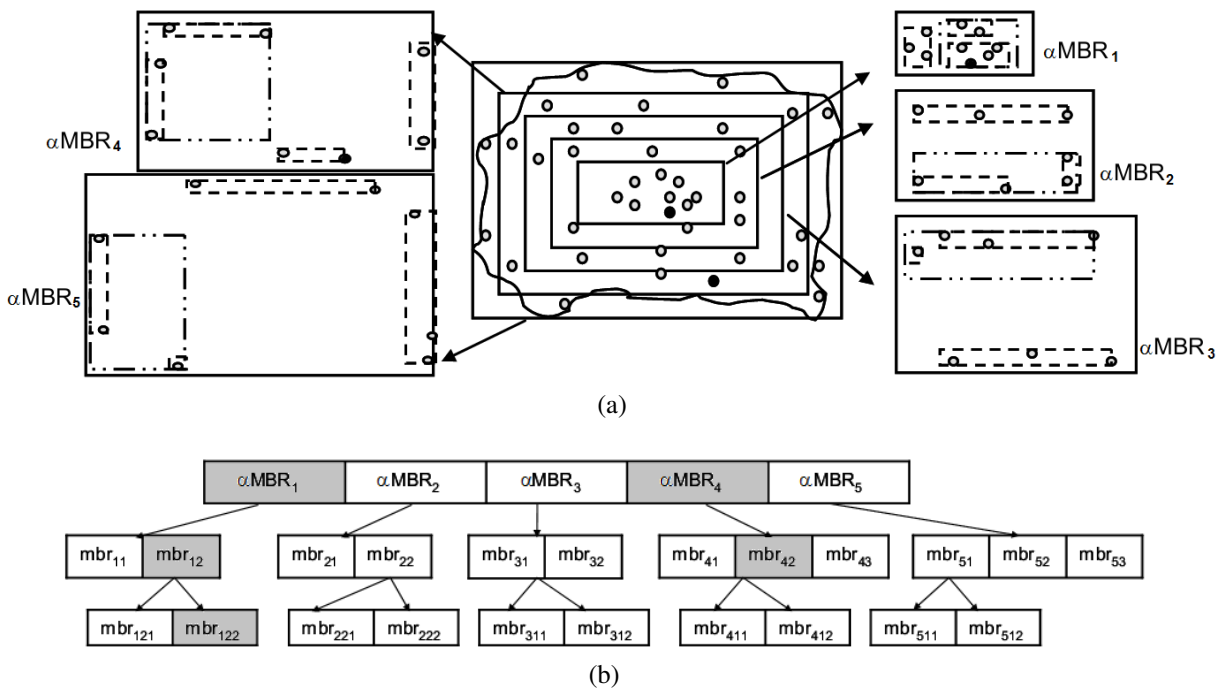


Figure 3.11: An example of FMBR R-tree (adapted from Petry, Ladner & Somodevilla (2007)). (a) A fuzzy region and points within it. (b) Data structure.

An example of containment range query using the FMBR R-tree shown in Figure 3.11 is: are there black points within the fuzzy region? The search starts at the first entry of the list and verifies whether there is a black point within α MBR₁. The test yields true and R*-tree pointed to by α MBR₁ is searched. In the root node, the search concludes that there is no black point within mbr₁₁, but there is a black point within mbr₁₂ and then the subtree is accessed. Although there is no black point within mbr₁₂₁, there is a black point within mbr₁₂₂. Such black point is added to the result set. The search then continues throughout the list, but there is no black point within α MBR₂ and neither within α MBR₃. Since there is a black point within α MBR₄, the subtree is accessed. Although there is no black point within mbr₄₁, there is a black point within mbr₄₂. Such black point is added to the result set. The search then continues throughout the list and finishes after yielding that there is no black point within α MBR₅. The result set is returned

and prioritizes the black point within mbr_{122} rather than the black point within mbr_{42} , since the former is within αMBR_1 that has membership value 1.0 while the latter is within αMBR_4 that has a membership value lesser than 1.0.

Although FMBR R-tree's authors claim that the index supports spatial range queries, distance queries, spatial joins, and nearest neighbor queries, detailed algorithms for processing these spatial predicates were not provided. Another issue that was not clarified is whether spatial objects that are not points, e.g. lines and regions, can also be queried within a fuzzy region. The example previously described provides a list of crisp points to be queried within a list of spatial query windows. The list is composed of the αMBR -cuts of the fuzzy MBR created for a fuzzy region. In fact, the FMBR R-tree does not query vague spatial data, but queries crisp spatial data within a fuzzy region. The FMBR R-tree was not assessed through a performance evaluation. There is a concern about the scalability of the FMBR R-tree if the volume of fuzzy regions is huge, such as in a SDW, because each fuzzy region would demand the creation of one FMBR R-tree.

3.4.3 Discussion

The vague R-tree is an index for processing point queries against vague regions designed according to exact models. However, the absence of detailed algorithms derail its implementation and utilization. The Fuzzy MBR R-tree is an index for searching crisp spatial objects located within a vague region. Although the Fuzzy MBR R-tree does not aim at querying vague spatial data, the characteristic of prioritizing results of a query that refer to higher membership values in the vague region is a feature that can be explored to query vague SDWs.

3.5 Summary

Existing conceptual models for SDWs reuse multidimensional modeling, define spatial attributes in dimensions or as measures in a fact of a data cube, and enable SOLAP. A spatial dimension provides a geographical perspective of analysis for the subject, while a spatial measure is the subject of the analysis itself. Hierarchies associate spatial attributes and constrain the cardinality of the relationship and the topological relationships that exist among instances of attributes from distinct levels of granularity. Aggregation functions for spatial measures perform geometric operations. The logical design of SDWs mainly produces a relational schema either by applying mapping rules over a conceptual schema or not.

Section 3.1 has surveyed existing work on conceptual models for SDWs and logical design for SDWs. The MultiDim conceptual model was proposed by Malinowski & Zimányi (2009) and extended by Vaisman & Zimányi (2014b). Both Pinet & Schneider (2010) and Boulil, Bimonte & Pinet (2015) described UML profiles. The SDWM metamodel was defined by Cuzocrea & Fidalgo (2012) to provide logical schemata of SDWs. The comprehension of those work is a prerequisite to elaborate a conceptual model for SDWs characterized by spatial vagueness and to design their logical schemata. Table 3.1 compares those work to major contributions of this thesis regarding conceptual modeling and logical design of vague SDWs, as described in Chapters 4 and 5, respectively.

Multidimensional models prevail in conceptual modeling of SDWs. Commonly, graphical notations for the concepts of the data cube are provided to enable the creation of diagrams. The MultiDim model's notation resembles the E-R model, while UML profiles extends the UML class diagram. Even the SDWM metamodel that addresses the logical design of SDWs also extends the UML class diagram. Spatial data are modeled either as spatial objects or continuous fields, but the latter are supported only by the MultiDim model. Pictograms are utilized by the MultiDim model to illustrate both spatial data types and topological constraints. Pinet & Schneider (2010) used OCL to specify a topological constraint. Mapping rules analogous to ER to relational mapping transform the conceptual schema according to the MultiDim model into a relational logical schema, while the UML profiles reused object-relational mapping for the same purpose. Topological constraints were graphically annotated by the MultiDim model and implemented as triggers in the underlying DBMS.

Table 3.1: Comparing existing work on SDW design to conceptual modeling and logical design of vague SDWs introduced in this thesis.

Features	MultiDim	UML profiles	SDWM	This
Is a multidimensional model	Yes	Yes	No	Yes
Enables the creation of diagrams	Yes	Yes	Yes	Yes
Supports spatial objects	Yes	Yes	Yes	Yes
Supports continuous fields	Yes	No	No	Yes
Has pictograms for spatial data types	Yes	No	Yes	Yes
Specifies topological constraints	Yes	Yes	No	Yes
Provides mapping rules to transform a conceptual schema into a logical schema	Yes	Yes	No	Yes
Produces a relational schema	Yes	Yes	Yes	Yes
Specifies topological constraints	Yes	Yes	No	Yes
Implements topological constraints	Yes	No	No	Yes
Addresses spatial vagueness	No	No	No	Yes

Section 3.2 has surveyed existing work that address SDW design considering spatial vague-

ness. Perez, Somodevilla & Pineda (2007, 2010) proposed the Fuzzy SDW. Jadidi et al. (2013, 2014) focused on coastal erosion risk assessment and proposed frameworks for conceptual modeling of a SDW and for creation of fuzzy regions. Edoh-Alove, Bimonte & Bédard (2014), Edoh-Alove et al. (2014) described the RADSOLAP method that provides users with several prototypes of the SDW conceptual schema, according to user's tolerance to vague spatial data in the SDW schema. Table 3.2 distinguishes those work from the design of vague SDWs described in Chapters 4 and 5 of this thesis.

Table 3.2: Comparing existing work on design of SDWs characterized by spatial vagueness to the design of vague SDWs described in this thesis.

Features	Fuzzy SDW	Frameworks	RADSOLAP	This
Supports vague spatial data types	No	No	Yes	Yes
Supports fuzzy spatial data types	Yes	Yes	No	Yes
Supports implementations for fuzzy spatial data types	Yes	Yes	No	Yes
Allows spatial vagueness in dimensions	No	No	Yes	Yes
Allows spatial vagueness in measures	No	No	No	Yes
Queries with vague SOLAP operations	No	No	No	Yes
Defines vague topological constraints	No	No	No	Yes

Different from related work, the design of vague SDWs described in this thesis addresses data types defined by exact models, fuzzy models, and implementations for fuzzy models. In contrast with the Fuzzy SDW and the frameworks, vague spatial attributes can be defined in dimensions as they are essential to evaluate spatial predicates on the fly to query the SDW. Different from the RADSOLAP method, vague spatial attributes can also be defined as measures in a fact to allow the aggregation of vague spatial data. Partial results of this thesis were used to design the logical schema of SDW used as case study for the RADSOLAP method. This fact corroborates the importance of designing vague spatial data warehouses and amplifies the challenges to be outperformed in the design of vague spatial data warehouses. Furthermore, two contributions of this thesis were not addressed by existing work in the literature: vague SOLAP operations to query the vague SDW and vague topological constraints to ensure the integrity of vague spatial data.

None of the aforementioned related work focused on physical design of SDWs. Then, Sections 3.3 and 3.4 have addressed physical design and surveyed indices for SDW and indices for vague regions, respectively, and outlined their data structures and query processing algorithms. Table 3.3 compare those indices to the Vague Spatial Bitmap Index (VSB-index) that is one of the contributions of this thesis, as described in Chapter 6. Existing indices use the MBR as conservative approximation, while the VSB-index has been prototyped and evaluated using

the MBR. The FMBR R-tree uses an inscribed rectangle corresponding to the core of the fuzzy MBR as progressive approximation, while the VSB-index has been prototyped and evaluated using the maximum area inscribed polygon described in Chapter 6. Both the VSB-index and existing indices approximate a vague region using a MBR and then they can process spatial predicates against vague regions, e.g. an intersection range query. On the other hand, only the VSB-index processes spatial predicates against elements of vague regions, i.e. parts that *certainly* belong to the vague region and parts that *possibly* belong to the vague region. Although most existing indices are able to identify answers of the spatial predicate already in the filter step, the VSB-index drastically enhances the performance to process queries and reduces the query response time as described in Chapter 6. The VSB-index also resolves conventional predicates, aggregation and sorting, like existing indices for SDWs.

Table 3.3: Comparing indices for SDW and indices for vague regions to the VSB-index.

Features	aR-tree	SB-index	Vague R-tree	FMBR R-tree	VSB-index
Conservative approximation	Yes	Yes	Yes	Yes	Yes
Progressive approximation	No	No	No	Yes	Yes
Designed for vague regions	No	No	Yes	Yes	Yes
Resolution of spatial predicates against vague regions	Yes	Yes	Yes	Yes	Yes
Resolution of spatial predicates against elements of vague regions	No	No	Partially	No	Yes
Identification of answers of the spatial predicate already in the filter step	Yes	Partially	No	Yes	Yes
Resolution of conventional predicates, aggregation and sorting	Yes	Yes	No	No	Yes

Chapter 4

CONCEPTUAL DESIGN OF VAGUE SPATIAL DATA WAREHOUSES

Conceptual modeling is crucial for designing databases as it focus on user's requirements, facilitates the communication between users and designers, and omit details of the underlying implementation platform. Conceptual modeling of multidimensional databases such as data warehouses converges on multidimensional modeling to elaborate a conceptual database schema known as data cube. Conceptual modeling of spatial data warehouses involves not only the design of a data cube, but also representing spatial data as objects or continuous fields.

This chapter addresses conceptual modeling of spatial data warehouses whose spatial data are partially or completely affected by spatial vagueness, i.e. vague spatial data warehouses. Two conceptual models are described: the Vague Spatial Cube (VSCube) and the Vague Spatial MultiDim (VSMultiDim). First, an overview of conceptual modeling of vague spatial data warehouses is detailed in Section 4.1. Then, the VSCube conceptual model is described in Sections 4.2 to 4.8, as follows. Section 4.2 describes attribute types and highlights the vague spatial attribute. Section 4.3 focuses on hierarchies that associate instances of attributes. Section 4.4 addresses the data cube assuming vague spatial data in dimensions and measures. Section 4.5 describes vague spatial predicates for selecting vague spatial objects. Section 4.6 details aggregation functions for vague spatial data. Section 4.7 introduces the vague spatial OLAP. Section 4.8 summarizes how vague spatial data designed according to existing models and implementations for spatial vagueness can be reused in the VSCube conceptual model. Furthermore, the VSMultiDim conceptual model is described in Section 4.9. Finally, Section 4.10 summarizes this chapter. In order to corroborate the applicability of both the VSCube and the VSMultiDim conceptual models and illustrate the use of each defined concept, examples have been elaborated considering the pest control and the HLB case studies.

4.1 Conceptual Modeling of Vague Spatial Data Warehouses

A vague spatial data warehouse (vague SDW) is a spatial data warehouse whose spatial data are partially or completely affected by the imperfection of spatial vagueness. A vague spatial object is an object from the real world that has at least one vague spatial attribute. A vague spatial attribute is the representation in space for an object from the real world, such that the representation is affected by spatial vagueness. In this thesis, the location and shape of a vague spatial object do not change through time. Multidimensional modeling of a vague SDW produces a data cube as conceptual schema, which supports vague spatial dimensions as well as vague spatial measures. A vague spatial dimension has at least one vague spatial attribute, while a vague spatial measure is a vague spatial attribute in a fact. The data cube of a vague SDW has not only vague spatial data, but also conventional data and crisp spatial data.

For example, consider a conceptual schema of a vague SDW built according to the requirements of the pest control case study. The data cube comprises the dimensions *Pesticide*, *Date*, and *Crop*. The amount of applied pesticides in tons is a numeric measure, while the areas where pesticides were applied to are vague spatial objects. The dimension *Crop* has vague spatial attributes to spatially describe crops and agricultural lands, as well as a crisp spatial attribute to address watersheds, among other conventional attributes. Conversely, the dimensions *Pesticide* and *Date* have only conventional attributes.

Conceptually, a vague spatial object has been represented using an exact model if it has elements that *certainly* belong to it and elements that *possibly* belong to it, or using a fuzzy model if its elements have a quantifiable degree of membership in $]0, 1]$. On the other hand, a conceptual schema of a vague SDW may not or cannot fulfill the user's requirements if a single model is chosen to design every vague spatial attribute. In other words, the use of a single exact model or a specific fuzzy model might not be sufficient to design all the vague spatial attributes in dimensions and all the vague spatial measures. Opting exclusively for an exact model constrains the vague SDW to contain vague spatial objects whose elements do not have a quantifiable membership degree. Opting strictly for a fuzzy model assumes that the vague SDW does not allow vague spatial objects to have elements whose degree of membership are not or cannot be estimated. Furthermore, neglecting that fuzzy sets are infinite and often require a finite implementation may cause difficulties to map the conceptual schema of vague SDW into a feasible logical schema of vague SDW. Therefore, conceptual modeling of vague SDWs must allow vague spatial attributes to be designed according to different exact models and fuzzy models and also comply with some characteristics of implementations for fuzzy models.

For example, consider the aforementioned data cube regarding pest control. It is not possible to assign membership values to parts that possibly belong to crops, as shown in Figure 1.1b. On the other hand, membership values are assigned to parts of areas where pesticides were applied to. Since it is not adequate to design crops using fuzzy models and neither applied areas using exact models, it is clear that the vague SDW conceptual schema call for both representations of vague spatial data. Furthermore, a discretization of the continuous surface where pesticides were applied to may be necessary, as shown in Figure 1.2d, and must also be supported.

A vague spatial attribute must clearly differentiate a valid vague spatial object from an invalid vague spatial object, but must also be general enough to enable the reuse of existing models for designing vague spatial objects. Besides, it should not allow ambiguity, but must omit some details that could impair the communication between users and designers. The vague shape of a vague spatial object is composed of two disjoint sets. One set has elements whose membership to the shape is confirmed with *certitude*¹, while for elements of the other set there is *dubiety*² about the membership to the shape. For these reasons, the former set is called *certitude* and the latter set is called *dubiety*.

Certitude elements belong to the vague spatial object, while dubiety elements either have an undetermined degree of membership or a quantifiable membership degree. The certitude generalizes the component that certainly belong to a vague spatial object as defined by exact models, as well as the subsets of a fuzzy spatial object whose membership value is 1.0 as defined by fuzzy models. Conversely, the dubiety generalizes the component that possibly belong to a vague spatial object as defined by exact models, as well as the subsets of a fuzzy spatial object whose membership value is in $]0, 1[$ as defined by fuzzy models. The certitude and the dubiety must be disjoint to prevent ambiguity. The shape of a certitude element and the shape of a dubiety element are approximated using geometries, comparable to implementations for fuzzy models.

For example, the certitude elements of the crop C_1 shown in Figure 1.2b are c_{11} and c_{12} , while the single dubiety element is c_{13} as shown in Figure 1.2c. Also, the certitude of the applied area A_1 shown in Figure 1.2a comprises the subset of the fuzzy region whose membership value is 1.0, while the dubiety encompasses the subset of the fuzzy region whose membership values are in $]0, 1[$. If the fuzzy set of A_1 is discretized as shown in Figure 1.2d, then the certitude comprises the parts where the possibility of application is 100%, while the dubiety encompasses the remaining parts.

¹Certitude: the state of being or feeling certain (<http://www.merriam-webster.com/dictionary/certitude>)

²Dubiety: the state or quality of being doubtful (<http://www.oxforddictionaries.com/definition/english/dubiety>)

Vague spatial attributes are defined in dimensions. Dimensions often encompass hierarchies associating attributes with different granularity. Specially when associating a pair of vague spatial attributes, the hierarchy must clearly state the cardinality of the relationship and topological constraints involving vague spatial objects. The topological constraints, then, relate the certitudes and the dubieties of these vague spatial objects. Not only vague spatial attributes may be associated, but also crisp spatial attributes and conventional attributes.

For example, consider a hierarchy between crops and agricultural lands. A crop is associated to at most one agricultural land, while an agricultural land maintains several crops. Since both are vague spatial attributes, a topological constraint specify which topological relationships are admissible among: (i) certitude elements of an agricultural land and certitude elements of a crop; (ii) certitude elements of an agricultural land and dubiety elements of a crop; (iii) dubiety elements of an agricultural land and certitude elements of a crop; and (iv) dubiety elements of an agricultural land and dubiety elements of a crop. As another example, consider a hierarchy between watersheds and agricultural lands, where the former are represented by a crisp spatial attribute and the latter are denoted by a vague spatial attribute. Then, a topological constraint specify which topological relationships are admissible among: (i) a watershed and certitude elements of an agricultural land; and (ii) a watershed and dubiety elements of an agricultural land.

A vague spatial measure is the subject of analysis in a fact. The aggregation of vague spatial objects is essential to allow summarization. Aggregation functions for vague spatial objects that have certitude and dubiety must perform not only geometric operations, but also consider undetermined degree of membership or membership values in $]0, 1[$ for the dubiety elements. A measure value in a crisp spatial data warehouse is associated to the entire vague spatial object whose vague spatial attribute is in a dimension of the data cube. Conversely, in the vague SDW, a value of measure can be assigned to certitude elements and dubiety elements of a vague spatial object whose vague spatial attribute is in a dimension of the data cube.

Vague spatial predicates allow selecting vague spatial objects that satisfy to a set of criteria. These criteria refer to the location and the shape that describe the vague spatial object. Additionally, the criteria involve the location and shape that describe the certitude elements and the dubiety elements of the vague spatial object. The criteria can also involve membership values of the elements.

It is noteworthy that the vague spatial attributes being hierarchically associated might comply with different models, i.e. exact or fuzzy. Furthermore, the aggregation of vague spatial objects must support vague spatial objects designed according to exact or fuzzy models. More-

over, the vague spatial predicates must select vague spatial objects designed according to exact or fuzzy models. The compliance to these requirements enables vague spatial online analytical processing (vague SOLAP) to be performed over the data cube that represents the conceptual vague SDW schema, with roll-up, drill-down, slice-and-dice, and pivot operations. For example, the data cube regarding pest control can be queried as shown in Table 1.1.

Although multidimensional modeling produces data cubes as the conceptual schema for a vague SDW, it does not enable the elaboration of diagrams or the use of symbols to represent concepts. Then, conceptual modeling of vague SDWs should also provide graphical representations for the concepts and enable the elaboration of diagrams to facilitate the communication between users and designers. To comply with such requirement, it is crucial to introduce a graphical notation for vague spatial data types and for topological constraints involving vague spatial attributes.

The overview regarding conceptual modeling of vague SDWs described in this section has guided the development of the VSCube and VSMultiDim conceptual models for vague SDWs. The VSCube is detailed in Sections 4.2 to 4.8 and focuses on multidimensional modeling, while the VSMultiDim is detailed in Section 4.9, reuses concepts from the VSCube and focuses on the graphical representation of the concepts and the elaboration of diagrams.

4.2 Attributes

An attribute describes the characteristics of a real world object. Given a real world object, the domain of an attribute is the set of values allowed for that attribute. Furthermore, an instance of an attribute is a valid value assigned to that attribute. The VSCube conceptual model classifies attributes according to the UML class diagram shown in Figure 4.1. These classes of attributes are essential to represent data in a variety of formats in a vague SDW. Recall that abstract classes are not instantiable, but concrete classes are. *Attribute* is an abstract class whose descendants are the concrete class Conventional Attribute and the abstract class *Spatial Attribute*. The descendants of the latter are the concrete class Non-geometric Spatial Attribute, the abstract class *Vague Spatial Attribute*, and the concrete class Crisp Spatial Attribute. The concrete classes Partially Vague Spatial Attribute and Completely Vague Spatial Attribute inherit the abstract class *Vague Spatial Attribute*.

Definition 4.2.1. A *conventional attribute* is an attribute whose domain has only numeric, alphanumeric or date values and, therefore, has a total order relation. A label specifying the data type of a conventional attribute optionally precedes its name, i.e. *numeric*, *date*, or *string*.

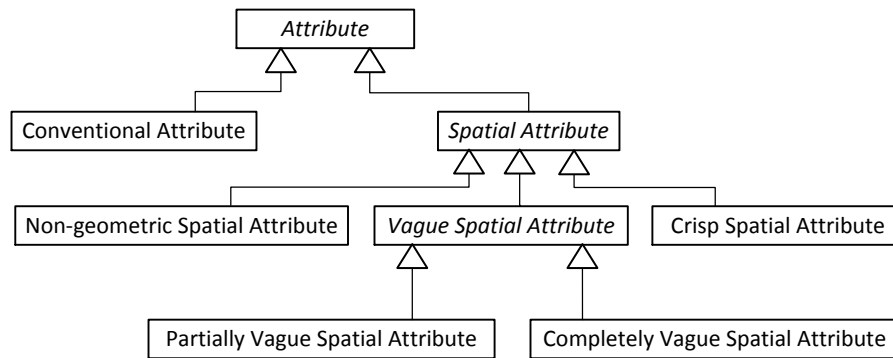


Figure 4.1: The types of attributes defined in the VSCube Conceptual Model.

Example 4.2.1. The conventional attributes $^{numeric}PesticideId$, $^{string}PesticideName$ and $^{date}Date$ refer to an identifier for a pesticide, a name of pesticide, and a date, respectively. Also, their domains are numeric, alphanumeric and the set of valid dates, respectively.

Definition 4.2.2. A *non-geometric spatial attribute* describes existing locations without providing geometric features or reference coordinates. The label nongeog optionally precedes the name of a non-geometric spatial attribute.

Example 4.2.2. The attributes $^{nongeog}Address$ and $^{nongeog}CityName$ are non-geometric spatial attributes referring to civic addresses and names of cities, respectively. Instances of $^{nongeog}Address$ are ‘1160 10th Street’ and ‘Avenue F.D. Roosevelt’, while instances of $^{nongeog}CityName$ are ‘Namur’ and ‘Brussels’. The domains of both $^{nongeog}Address$ and $^{nongeog}CityName$ are alphanumeric.

Definition 4.2.3. A *crisp spatial attribute* is a geometric feature assuming exact location and well-known shape and boundaries for a given phenomenon in the 2D Euclidean space. The label crisp optionally precedes the name of a crisp spatial attribute.

Example 4.2.3. The crisp spatial attribute $^{crisp}Address$ locates an address with one point. An instance is shown in Figure 4.2a. In addition, $^{crisp}Watershed$ describes a watershed using one (complex) region whose boundaries are crisp. An instance is illustrated in Figure 4.2b.

Different from a crisp spatial attribute, a vague spatial attribute provides a representation for the spatial vagueness that affects a real world object.

Definition 4.2.4. A *vague spatial attribute* is a composite attribute denoted $v = \langle certitude, dubiety \rangle$, where:

- $certitude = geo_{certitude}$, where $geo_{certitude}$ is a simple or a complex crisp geometric feature in the 2D Euclidean space;

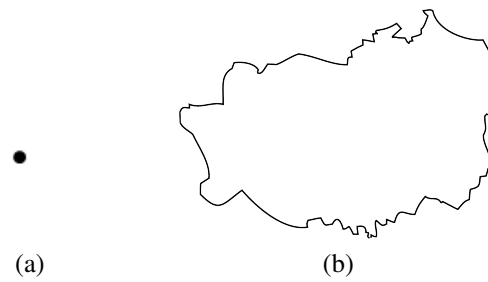


Figure 4.2: Instances of crisp spatial attributes: (a) An address. (b) A watershed.

- $dubiety = \langle geo_{dubiety}, [mval] \rangle$, where $geo_{dubiety}$ is a simple or a complex crisp geometric feature in the 2D Euclidean space, and $mval$ is an optional membership value associated to $geo_{dubiety}$; and
- both the *certitude* and *dubiety* are multivalued.

The *certitude* describes the features of a given phenomenon that are assumed to have exact location and well-known shape and boundaries, while the *dubiety* describes the vagueness of the shape of such phenomenon. Therefore, the *certitude* represents the space where a given phenomenon certainly occurs, while the *dubiety* represents the space where the phenomenon may occur according to a membership degree. The representation of the vague spatial attribute v according to the E-R notation of Elmasri & Navathe (2010) is shown in Figure 4.3.

Since the vague spatial attribute is a composite attribute comprising a multivalued *certitude* and a multivalued *dubiety*, an instance z of a vague spatial attribute is composed of *certitude elements* and *dubiety elements*. A *certitude element* is a geometry, while a *dubiety element* is a pair comprising a geometry and its corresponding membership value. Accessors provide access to elements from *certitude* and *dubiety* of a given instance z . The following accessors are provided by the VSCube conceptual model:

- the *certitude geometry accessor* returns all geometries that compose the *certitude* of z and is $z.certitude.geo$.
- the *certitude element geometry accessor* retrieves the i -th geometry of the *certitude* and is $z.certitude[i].geo$.
- the *dubiety geometry accessor* returns the collection of geometries that compose the *dubiety* of z and is $z.dubiety.geo$.
- the *dubiety memberships accessor* returns the collection of membership values of the elements in the *dubiety* of z and is $z.dubiety.mval$.

- the *dubiety element geometry accessor* retrieves the j -th geometry of the dubiety and is $z.dubiety[j].geo$.
- the *dubiety element membership accessor* retrieves the membership associated to the j -th geometry of the dubiety and is $z.dubiety[j].mval$.

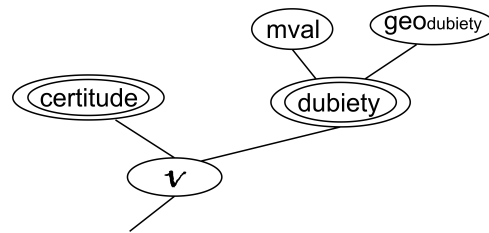


Figure 4.3: The vague spatial attribute v is a composite attribute with a multivalued certitude denoted by geometries and a multivalued dubiety denoted by geometries and their membership values.

The following constraints apply to every instance z of a given vague spatial attribute v , in order to allow aggregation of vague spatial data (Section 4.6) and vague spatial predicates (Section 4.5):

- the interior of $geo_{certitude}$ is disjoint from the interior of $geo_{dubiety}$.
- $\forall j \geq 0$, only one of the following conditions is valid:
 1. $z.dubiety[j].mval$ is in $]0, 1[$; or
 2. $z.dubiety[j].mval$ is determined by a function whose codomain is $]0, 1[$; or
 3. $z.dubiety[j].mval$ is *null* and means *maybe*.

A membership value in $]0, 1[$ indicates that z was designed according to a fuzzy model and then discretized. A membership value defined by a function denotes that z was designed according to a fuzzy model and then represented by a continuous surface or by a linear feature with gradual transitions. A null membership value associated to the dubiety indicates that z was designed according to an exact model and complies with the 3-valued logic (*true*, *maybe*, *false*). All instances of a vague spatial attribute must belong to the same data type to maintain consistency. A discussion on the reuse of data types from existing models is detailed in Section 4.8.

As shown in Figure 4.1, vague spatial attributes are distinguished. A *completely vague spatial attribute* has all instances with non-empty dubiety. Its name is optionally preceded by the label *vague*●. Conversely, a *partially vague spatial attribute* has some instances not affected

by vagueness, and then with empty dubiety, i.e. these instances are crisp. Its name is optionally preceded by the label *vague* \bullet .

Example 4.2.4. Consider the pest control case study. The completely vague spatial attribute *vague* \bullet *AppliedArea* take into account the spatial vagueness affecting areas where pesticides were applied to. Figure 4.4a exemplifies the vague region A_1 , which has the region A_{11} composing its certitude, and the regions A_{12} , A_{13} and A_{14} composing its dubiety, with the membership values 0.8, 0.5 and 0.3, respectively. The partially vague spatial attribute *vague* \blacktriangleright *Crop* describes crops, such that not every crop is affected by spatial vagueness, as shown in Figure 1.1b. In Figure 4.4b, the vague region C_1 has the regions c_{11} and C_{12} composing its certitude, and the region c_{13} as its dubiety.

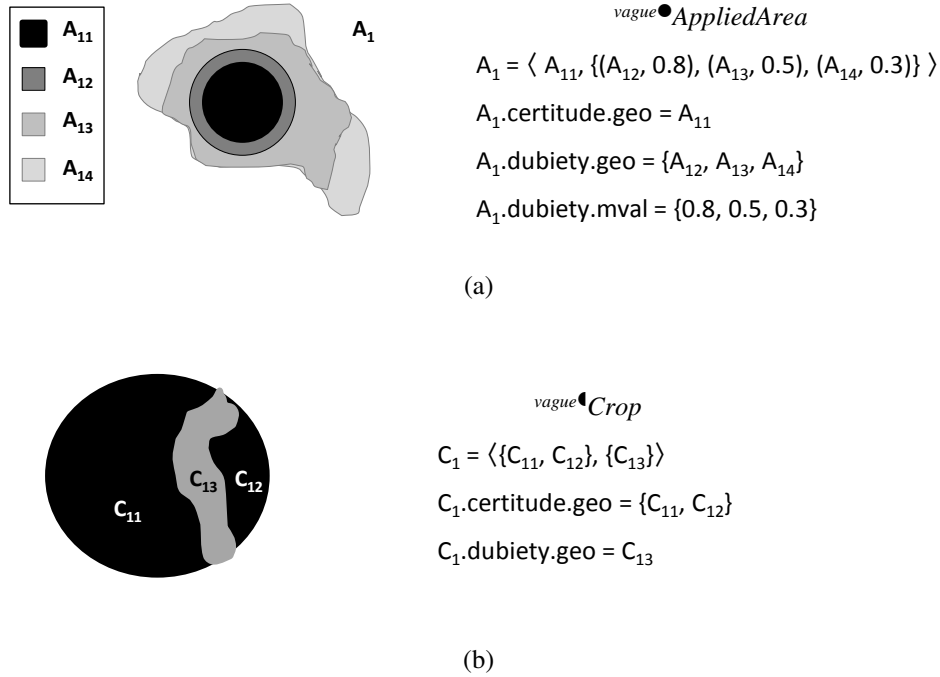


Figure 4.4: Vague spatial attributes and instances for the pest control case study. (a) An applied area. (b) A crop.

The vague spatial attribute is a concept and therefore is independent of implementation. It supports geometric shapes and the corresponding membership values. In detail, it supports data types defined by both exact models and implementations for fuzzy models. As a result, it provides expressiveness to represent vague spatial data. The crisp spatial attribute and the vague spatial attribute differ. If an attribute is crisp it cannot become vague, and vice-versa. In addition, shapes of vague spatial objects are assumed to remain unchanged through time. The logical design of the vague spatial attribute is tackled in Sections 5.2 and 5.3.

4.3 Hierarchies

Hierarchies characterize how instances of attributes are associated and guides data aggregation according to finer or coarser levels of granularity. The hierarchy operator formalized in Section 4.3.1 associates two attributes. The properties detailed in Section 4.3.2 enable hierarchies to associate several attributes. Finally, hierarchies are categorized in Section 4.3.3 according to their associated attributes.

4.3.1 Hierarchy Operator

To motivate the existence of an operator that hierarchically associates a pair of attributes, consider the pest control case study. Not only an agricultural land comprises several crops, but they are also topologically related. Figure 4.5 illustrates an agricultural land and crops. Note that elements from the certitude and the dubiety of both the agricultural land and the crops are related through the topological relationships shown in Figure 4.5. Therefore, a hierarchy operator must relate a pair of attributes according to a cardinality and specify the topological relationships allowed among their instances, taking into account elements from both certitude and dubiety.

Definition 4.3.1. The operator \preceq imposes a total order that associates a pair of attributes x_i and x_{i+1} according to a cardinality and restricts the topological relationships allowed among the instances of x_i and x_{i+1} , i.e.

$$\begin{array}{c}
 \text{cardinality} \\
 x_i \preceq x_{i+1} \\
 R_{(c,c)} \\
 R_{(c,d)} \\
 R_{(d,c)} \\
 R_{(d,d)}
 \end{array}$$

such that:

- *cardinality* is 1:1 or 1:N;
- $R_{(c,c)}$ is a set of topological relationships allowed among certitude elements of instances from x_i and certitude elements of instances from x_{i+1} ;
- $R_{(c,d)}$ is a set of topological relationships allowed among certitude elements of instances from x_i and dubiety elements of instances from x_{i+1} ;

- $R_{(d,c)}$ is a set of topological relationships allowed among dubiety elements of instances from x_i and certitude elements of instances from x_{i+1} ;
- $R_{(d,d)}$ is a set of topological relationships allowed among dubiety elements of instances from x_i and dubiety elements of instances from x_{i+1} ; and
- $R_{(c,c)}$, $R_{(c,d)}$, $R_{(d,c)}$, and $R_{(d,d)}$ are subsets of the set of topological relationships given by $\{\textit{meets, contains, inside, equals, overlaps, intersects, covers, covered by and disjoint}\}$.

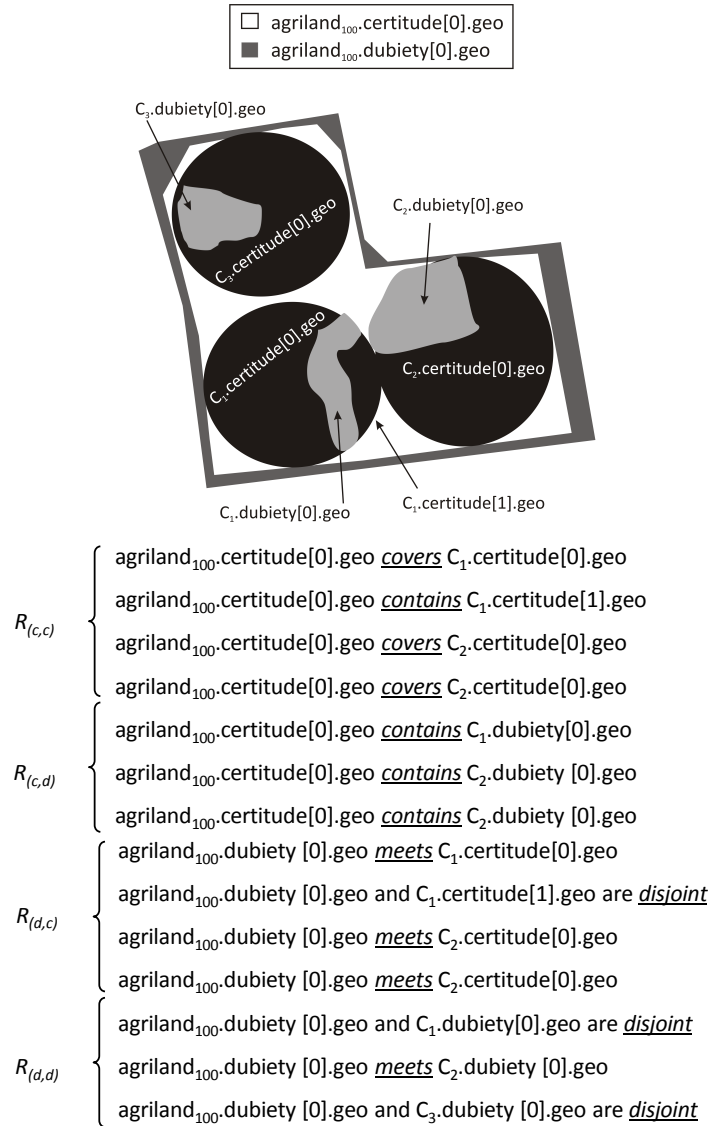


Figure 4.5: The topological relationships among agricultural lands and crops.

Table 4.1 shows the types that the attributes x_i and x_{i+1} can assume in a $x_i \preceq x_{i+1}$ statement. The existence of the sets of topological relationships $R_{(c,c)}$, $R_{(c,d)}$, $R_{(d,c)}$, and $R_{(d,d)}$ is assured only if both x_i and x_{i+1} are vague spatial attributes (line 1 in Table 4.1). None of these sets exist if one of the attributes is conventional or non-geometric (lines 3, 4, 7-16 in Table 4.1) since

there are no geometries involved in such types of attributes. As a result, all these sets are empty and can be omitted for the sake of simplicity. If x_i is crisp and x_{i+1} is vague, then only $R_{(c,c)}$ and $R_{(c,d)}$ exist (line 5 in Table 4.1), since x_i holds only the certitude. If x_i is vague and x_{i+1} is crisp, then only $R_{(c,c)}$ and $R_{(d,c)}$ exist (line 2 in Table 4.1) since x_{i+1} holds only the certitude. Finally, and if both x_i and x_{i+1} are crisp (line 6 in Table 4.1), then only $R_{(c,c)}$ exists since both x_i and x_{i+1} hold only the certitude.

Table 4.1: Types that the attributes x_i and x_{i+1} can assume.

#	x_i	x_{i+1}
1	Vague Spatial	Vague Spatial
2	Vague Spatial	Crisp Spatial
3	Vague Spatial	Non-geometric Spatial
4	Vague Spatial	Conventional
5	Crisp Spatial	Vague Spatial
6	Crisp Spatial	Crisp Spatial
7	Crisp Spatial	Non-geometric Spatial
8	Crisp Spatial	Conventional
9	Non-geometric Spatial	Vague Spatial
10	Non-geometric Spatial	Crisp Spatial
11	Non-geometric Spatial	Non-geometric Spatial
12	Non-geometric Spatial	Conventional
13	Conventional	Vague Spatial
14	Conventional	Crisp Spatial
15	Conventional	Non-geometric Spatial
16	Conventional	Conventional

Example 4.3.1. The spatial vagueness is represented in agricultural lands by the attribute $vague \bullet AgriLand$ and in crops by the attribute $vague \blacktriangleright Crop$. The types of these attributes are those of line 1 in Table 4.1. Based on the topological relationships shown in Figure 4.5, the hierarchy relating agricultural lands and crops is:

$$\begin{array}{lcl}
 & 1 : N & \\
 AgriLand & \preceq & Crop \\
 R_{(c,c)} & = & \{covers, contains\} \\
 R_{(c,d)} & = & \{contains\} \\
 R_{(d,c)} & = & \{meets, disjoint\} \\
 R_{(d,d)} & = & \{disjoint, meets\}
 \end{array}$$

Example 4.3.2. Watersheds are spatially represented by the attribute $crisp Watershed$. A watershed encompasses several agricultural lands. The types of these attributes are those of line 5 in Table 4.1. Based on the topological relationships shown in Figure 4.6, the hierarchy relating watersheds and agricultural lands is:

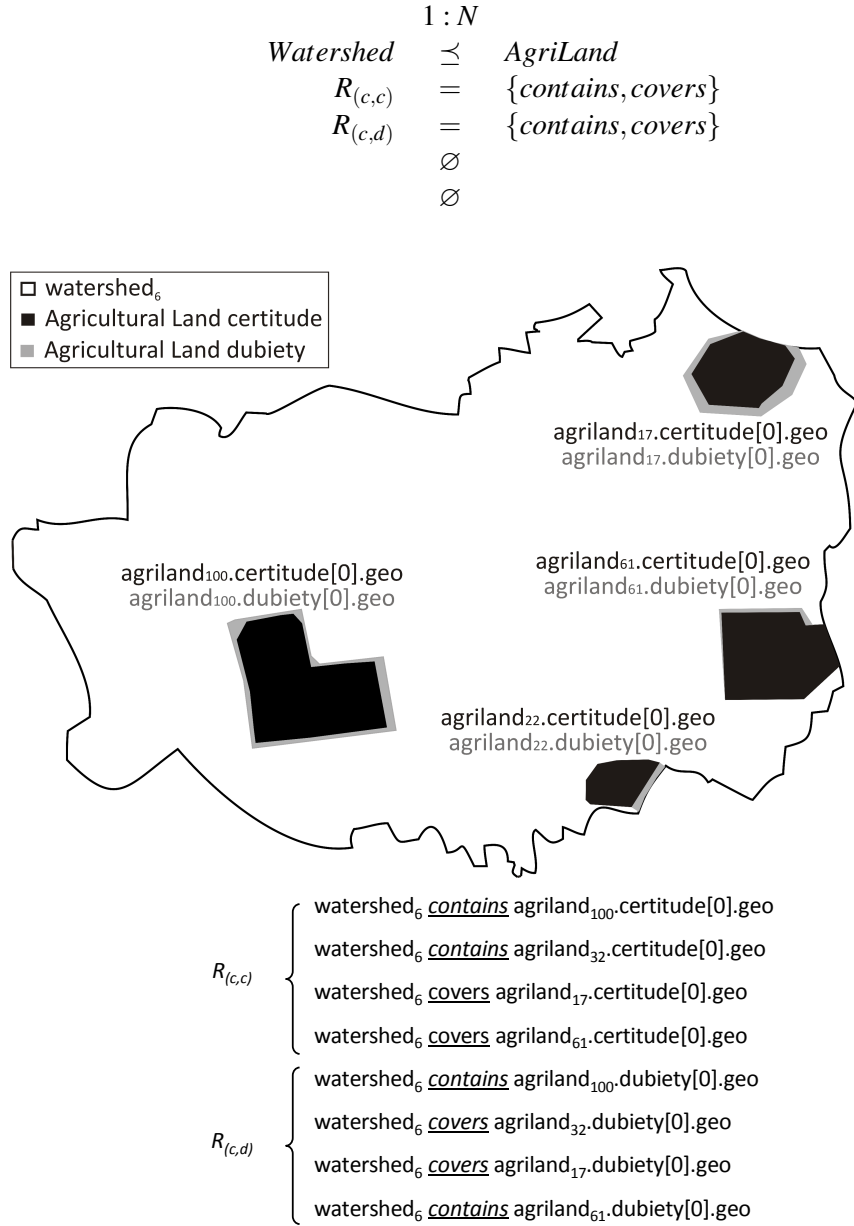


Figure 4.6: The topological relationships among watersheds and agricultural lands.

4.3.2 Properties of Hierarchies

A hierarchy comprises a set of pairwise associations and some properties are held to ensure them, as follows.

Definition 4.3.2. A hierarchy H is a total order on a set of attributes and is denoted $x_0 \preceq \dots \preceq x_n$, where x_i and x_n are attributes and $0 \leq i < n$. Furthermore:

- if $x_i \preceq x_{i+1}$ and $x_{i+1} \preceq x_i$ then $x_i = x_{i+1}$ means that cardinality is 1:1 and that the granularity of x_i and x_{i+1} refer to the same level of detail; (antisymmetry).

- if

$$\begin{array}{ccccc}
 & cij & & cjk & & cik \\
 x_i & \preceq & x_j & x_j & \preceq & x_k & x_i & \preceq & x_k \\
 R_{ij(c,c)} & & & R_{jk(c,c)} & & & R_{ik(c,c)} & & \\
 R_{ij(c,d)} & & \text{and} & R_{jk(c,d)} & & \text{then} & R_{ik(c,d)} & & \\
 R_{ij(d,c)} & & & R_{jk(d,c)} & & & R_{ik(d,c)} & & \\
 R_{ij(d,d)} & & & R_{jk(d,d)} & & & R_{ik(d,d)} & &
 \end{array}$$

where:

- cik is 1:1 if both cij and cjk are 1:1, otherwise cik is 1:N;
 - $R_{ik(c,c)} = \{disjoint\} \wedge R_{ik(c,d)} = \{disjoint\} \wedge R_{ik(d,c)} = \{disjoint\} \wedge R_{ik(d,d)} = \{disjoint\}$ is not allowed if both x_i and x_k are vague spatial attributes;
 - $R_{ik(c,c)} = \{disjoint\} \wedge R_{ik(c,d)} = \{disjoint\}$ is not allowed if x_i is a crisp spatial attribute and x_k is a vague spatial attribute;
 - $R_{ik(c,c)} = \{disjoint\} \wedge R_{ik(d,c)} = \{disjoint\}$ is not allowed if x_i is a vague spatial attribute and x_k is a crisp spatial attribute;
 - $R_{ik(c,c)} = \emptyset \wedge R_{ik(c,d)} = \emptyset \wedge R_{ik(d,c)} = \emptyset \wedge R_{ik(d,d)} = \emptyset$ if, and only if both x_i and x_k are not (vague or crisp) spatial attributes; (transitivity).
- $x_i \preceq x_{i+1}$ or $x_{i+1} \preceq x_i$ (totality).

Although two attributes can be considered in the same level of granularity (antisymmetry), a hierarchy imposes that one precedes the other (totality). In addition, transitivity is only held if the geometries of different attributes are not all disjoint. When transitivity is applied to relate a pair of attributes, the valid topological relationships need to be identified, as it is not possible to automatically determine them.

Example 4.3.3. Example 4.3.2 has related watersheds to agricultural lands, while Example 4.3.1 related agricultural lands to crops. Then, by transitivity:

$$\begin{array}{ccc}
 & 1:N & \\
 Watershed & \preceq & Crop \\
 R_{(c,c)} & = & \{contains, covers\} \\
 R_{(c,d)} & = & \{contains, covers\} \\
 & \emptyset & \\
 & \emptyset &
 \end{array}$$

Example 4.3.4. Based on Examples 4.3.1 to 4.3.3, the hierarchy $H_{WaterSupply}$ relates watersheds, agricultural lands, and crops:

$$\begin{array}{ccccccc}
 H_{WaterSupply} = & \text{crisp}Watershed & & 1 : N & & \text{vague}\bullet AgriLand & & 1 : N & & \text{vague}\blacktriangleright Crop \\
 & & & \preceq & & & & \preceq & & \\
 & & & \{contains, covers\} & & & & \{covers, contains\} & & \\
 & & & \{contains, covers\} & & & & \{contains\} & & \\
 & & & \emptyset & & & & \{meets, disjoint\} & & \\
 & & & \emptyset & & & & \{disjoint, meets\} & &
 \end{array}$$

Figure 4.7 illustrates instances of the attributes $\text{crisp}Watershed$, $\text{vague}\bullet AgriLand$, and $\text{vague}\blacktriangleright Crop$ that comply with the hierarchy $H_{WaterSupply}$.

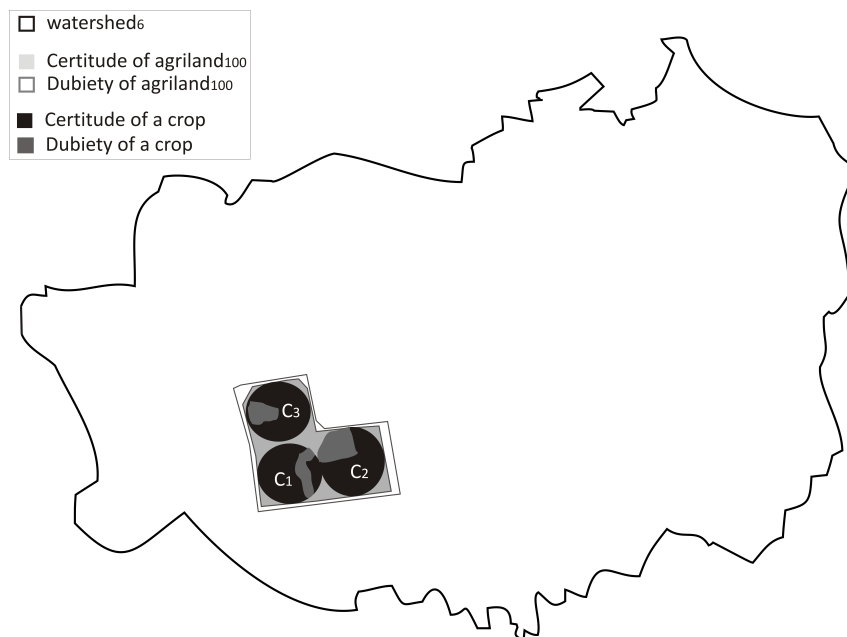


Figure 4.7: A watershed, an agricultural land and crops.

Not only vague spatial attributes and crisp spatial attributes are related by hierarchies in the VSCube model. The following example addresses a hierarchy involving other attribute types.

Example 4.3.5. Political boundaries are set by the hierarchy $H_{Location}$ such that the sets $R_{(c,c)}$, $R_{(c,d)}$, $R_{(d,c)}$, and $R_{(d,d)}$ are omitted as all of them are empty:

$$H_{Location} = \text{crisp}Country \preceq 1 : N \text{nongeo} StateName \preceq 1 : N \text{crisp}City \preceq 1 : 1 \text{numeric} CityCode$$

Then, by transitivity:

Table 4.2: Categories of hierarchies of the VSCube conceptual model.

Category of hierarchy	Vague spatial attributes	Crisp spatial attributes	Non-geometric spatial attributes	Conventional attributes
Hybrid hierarchy	✓	✓	✓	✓
Completely spatial hierarchy	✓	✓		
Partially vague spatial hierarchy	✓	✓	✓	
	✓	✓		✓
	✓		✓	✓
	✓		✓	
Partially crisp spatial hierarchy		✓	✓	✓
		✓	✓	
		✓		✓
Partially non-geometric spatial hierarchy			✓	✓
Vague spatial hierarchy	✓			
Crisp spatial hierarchy		✓		
Non-geometric spatial hierarchy			✓	
Conventional hierarchy				✓

$$\begin{aligned}
 & \text{crispCountry} \stackrel{1:N}{\preceq} \text{crispCity} \\
 R_{(c,c)} &= \{\text{contains, overlaps}\} \\
 & \emptyset \\
 & \emptyset \\
 & \emptyset
 \end{aligned}$$

Note that the set $R_{(c,c)}$ is the only non-empty set, since countries and cities are crisp and thus do not have dubiety. According to Table 4.1, countries and names of states are associated according to line 7, names of states and cities are associated according to line 10, a city's territory and code are associated according to line 8, and territories from countries and their cities are associated according to line 6 in Table 4.1.

4.3.3 Categories of Hierarchies

The hierarchies of the VSCube conceptual model are categorized in Table 4.2 according to the types of attributes they associate. For instance, a hybrid hierarchy holds attributes of all types, while a completely spatial hierarchy holds only vague spatial attributes and crisp spatial attributes.

Example 4.3.6. As examples of different categories of hierarchies, $H_{WaterSupply}$ described in Example 4.3.4 is a completely spatial hierarchy, and $H_{Location}$ described in Example 4.3.5 is a partially crisp spatial hierarchy.

4.4 Multidimensional Cube with Vague Spatial Data

The following sections contextualize vague spatial data in the multidimensional cube. Section 4.4.1 describes dimensions, Section 4.4.2 details measures, Section 4.4.3 addresses the cube, Section 4.4.4 characterizes the vague spatial fact, and Section 4.4.5 focuses on the lattice of cuboids.

4.4.1 Dimensions

Dimensions match the axes of a multidimensional data cube and are composed of attributes, which are often associated through hierarchies.

Definition 4.4.1. A dimension D is a tuple denoted $TypeDim D = \langle TypeAttr_0, \dots, TypeAttr_v \rangle$, where:

- x_i is an attribute and $0 \leq i \leq v$;
- $TypeAttr_i \in \{numeric, string, date, nongeo, crisp, vague \blacktriangleright, vague \bullet\}$ is the type of the attribute, i.e. numeric, alphanumeric (string), date, non-geometric spatial, crisp spatial, partially vague spatial or completely vague spatial, respectively;
- $TypeDim \in \{conv, nongeo, crisp, vague\}$ refers to the type of the dimension, as follows:
 - a *conventional dimension* has only conventional attributes;
 - a *non-geometric spatial dimension* has at least one non-geometric spatial attribute and optional conventional attributes;
 - a *crisp spatial dimension* has at least one crisp spatial attribute and no vague spatial attributes; and
 - a *vague spatial dimension* has at least one vague spatial attribute.
- if x_i univocally identifies one instance of the dimension, then it is considered a key and is underlined.

Example 4.4.1. The following dimensions provide perspectives of analysis for the pest control case study:

$${}^{conv}D_{Pesticide} = \langle {}^{numeric} \underline{PesticideId}, {}^{string} PesticideName, {}^{string} PesticideType \rangle$$

$$\text{conv}D_{Date} = \langle \text{date } \underline{Date}, \text{numeric } Month, \text{string } Quarter, \text{numeric } Year \rangle$$

$$\text{vague}D_{Crop} = \langle \text{vague } \blacktriangleright \underline{Crop}, \text{string } PlantName, \\ \text{vague } \bullet \text{ AgriLand}, \text{string } AgriLandOwner, \\ \text{crisp } Watershed, \text{string } WatershedName \rangle$$

Furthermore, the following instances of the dimension $\text{vague}D_{Crop}$ are illustrated in Figure 4.7:

$\langle C_1, \text{'Sugar cane'}, \text{agriland}_{100}, \text{watershed}_6 \text{'Piracicaba'} \rangle$

$\langle C_2, \text{'Sugar cane'}, \text{agriland}_{100}, \text{watershed}_6 \text{'Piracicaba'} \rangle$

$\langle C_3, \text{'Sugar cane'}, \text{agriland}_{100}, \text{watershed}_6 \text{'Piracicaba'} \rangle$

4.4.2 Measures

Measures are the subject of analysis in a data cube and often denote business scores and performance. In a vague SDW, measures can assume conventional values or (vague) spatial values.

Definition 4.4.2. A measure TypeMea_m is an attribute whose type is $\text{TypeMea} \in \{\text{numeric}, \text{crisp}, \text{vague } \blacktriangleright, \text{vague } \bullet\}$.

Example 4.4.2. To comply with the pest control case study, the quantity of pesticides applied in tons is addressed by the measure $\text{numeric}AppliedTons$. Furthermore, the vague regions where pesticides were applied to are addressed by the vague spatial measure $\text{vague } \bullet AppliedArea$. Note that $AppliedTons = 0.30$ and $AppliedArea = A_1$ as illustrated in Figure 1.2b.

4.4.3 Cube

A cube is a multidimensional view whose axes are the dimensions. The values of an axis are the values assigned to attributes of the dimension, while the values of measures are projected on axes. A fact associates values of attributes in dimensions to values of measures. Since (vague) spatial data can be addressed on dimensions and facts, the cube provides not only a multidimensional view, but also a geographic view of data. The measure values of a fact are often summarized using aggregation functions. In the VSCube conceptual model, numeric measures are aggregated by existing aggregation functions, e.g. SUM, AVG, MIN, MAX, and COUNT. Furthermore, crisp spatial measures are aggregated by reusing existing spatial aggregation functions, such as Union. Moreover, vague spatial measures are aggregated by aggregation functions such as $VUnion$, $VIntersection$ and $VSDifference$, which are introduced in Section 4.6.1.

Definition 4.4.3. A cube C is a tuple denoted $C = \langle D_0, \dots, D_p, {}^{TypeMea0}m_0, g_0, \dots, {}^{TypeMeaq}m_q, g_q \rangle$, where D_i is a dimension, ${}^{TypeMeaj}m_j$ is a measure, and g_j is an aggregation function applied to ${}^{TypeMeaj}m_j$, for $0 \leq i \leq p$ and $0 \leq j \leq q$.

The cube comprises dimensions and all their attributes, and therefore provides several levels of granularity, according to hierarchies that relate the attributes. The cube represents the conceptual schema of a vague SDW modeled according to the VSCube model. Attributes of a given dimension that do not participate in the hierarchy belong to the same level of granularity of the key of the dimension, such as the attribute ${}^{string}PlantName$ in ${}^{vague}D_{Crop}$ described in Example 4.4.1.

A cuboid maintains zero or one attribute of each dimension and then denotes a distinct level of granularity of the cube. Measures and aggregation functions of a cuboid are those defined in its cube.

Definition 4.4.4. A cuboid c is a tuple denoted $c = \langle C, {}^{TypeAttr0}x_0, \dots, {}^{TypeAttrt}x_t \rangle$, where:

- C is the cube that defines c ;
- $0 \leq i \leq t$;
- x_i is an optional attribute of a dimension in C ;
- $\forall s \leq i, j \leq t : i \neq j \Rightarrow x_i \neq x_j$ (there are no repeated attributes in c); and
- $TypeAttri \in \{numeric, string, date, nongeog, crisp, vague \blacktriangleright, vague \bullet\}$.

A fact assigns values of measures for each value of the attributes in a cuboid.

Definition 4.4.5. A fact f is a set denoted $f = \{c, a_0, \dots, a_p, value_0, \dots, value_q\}$ or $f = \{value_0, \dots, value_q\}$ where:

- c is a cuboid;
- x_i is an attribute of the cuboid c , for $0 \leq i \leq p$;
- $a_i \in Dom(x_i)$;
- m_j is a measure held by the cube C where the cuboid c is defined, for $0 \leq j \leq q$; and
- $value_j \in Dom(m_j)$.

The general case adopts the definition of fact as $f = \{a_0, \dots, a_p, value_0, \dots, value_q\}$. On the other hand, the distinguished cuboid c_{all} described in Section 4.4.5 does not hold any attributes, and then values of measures are associated to it by the fact denoted $f = \{value_0, \dots, value_q\}$. Therefore, c_{all} holds the aggregated values of the measures over all dimensions.

Example 4.4.3. Let $C_{PesticideApplication}$ be a cube and c_0 be a cuboid:

$$C_{PesticideApplication} = \langle \overset{conv}{D}_{Pesticide}, \overset{conv}{D}_{Date}, \overset{vague}{D}_{Crop}, \\ \overset{conv}{AppliedTons}, \text{SUM}, \overset{vague}{AppliedArea}, \text{VSUnion} \rangle$$

$$c_0 = \langle C_{PesticideApplication}, \overset{numeric}{PesticideId}, \overset{date}{Date}, \overset{vague}{Crop} \rangle$$

The cuboid c_0 has the attributes that are necessary to provide an answer for the query PC1 listed in Table 1.1. The multidimensional and geographic view provided by the cuboid c_0 , which holds the keys of all dimensions in $C_{PesticideApplication}$, is depicted in Figure 4.8. In, c_0 , each fact associates values of measures to values of keys of pesticides (121, 122), dates ('2012-10-01' to '2012-10-04'), and crops (C_1, C_2, C_3). The fact highlighted in Figure 4.8 is $\{121, '2012-10-04', C_1, 0.30, A_1\}$.

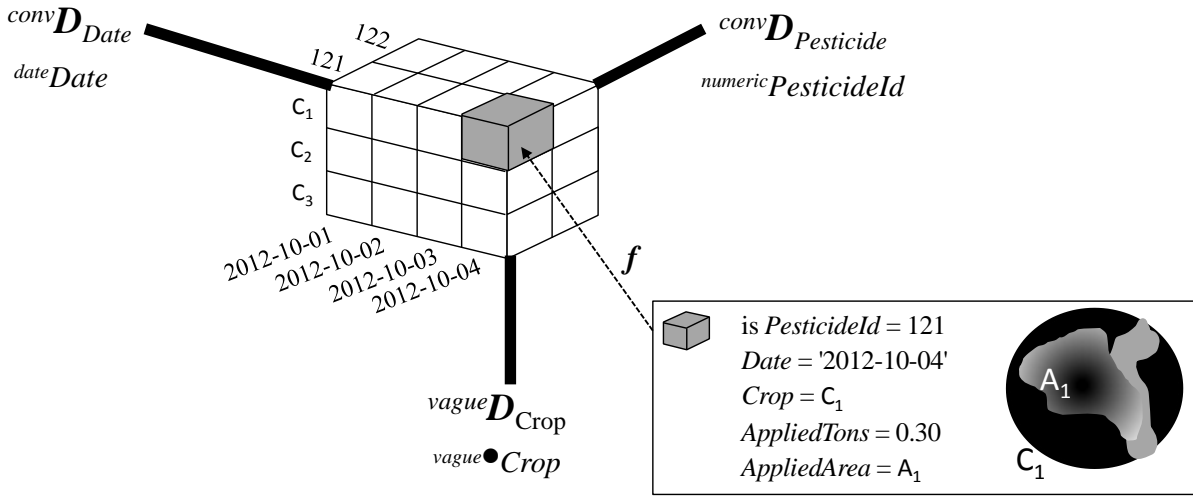


Figure 4.8: The multidimensional and geographic views provided by the cuboid c_0 , and the fact f .

4.4.4 Vague Spatial Fact

A vague spatial fact has a finer granularity than a fact and enables the analysis of measures at the grains of certitude and dubiety when a vague spatial attribute is addressed by the cuboid. A vague spatial fact is defined for each vague spatial attribute in a given cuboid.

Definition 4.4.6. A vague spatial fact $\overset{vague}{f}$ is a $(|v.certitude.geo| + |v.dubiety.geo|) \times M$ matrix for each fact where:

- v is a vague spatial attribute addressed by the cuboid c of the cube C ;
- $|v.certitude.geo|$ is the cardinality of $v.certitude.geo$;
- $|v.dubiety.geo|$ is the cardinality of $v.dubiety.geo$;
- M is the quantity of measures addressed in cube C ;
- the cells of the matrix are $cell_{i,j}$, for $0 \leq i < M$ and $0 \leq j < |v.certitude.geo| + |v.dubiety.geo|$;
- each cell $cell_{i,j}$, with $0 \leq j < |v.certitude.geo|$, is the value of the i -th measure of the cube C regarding the j -th geometry of $v.certitude.geo$;
- each cell $cell_{i,j}$, with $|v.certitude.geo| \leq j < |v.certitude.geo| + |v.dubiety.geo|$, is the value of the i -th measure of the cube C regarding the $(j - |v.certitude.geo|)$ -th geometry of $v.dubiety.geo$;
- the use of the corresponding aggregation function on the i -th measure, to aggregate all $cell_{i,j}$ with $0 \leq j < |v.certitude.geo|$ returns the aggregated measure for the certitude of the instance; and
- the use of the corresponding aggregation function on the i -th measure, to aggregate all $cell_{i,j}$ with $|v.certitude.geo| \leq j < |v.certitude.geo| + |v.dubiety.geo|$, returns the aggregated measure for the dubiety of the instance.

Example 4.4.4. Consider the crop C_1 and the applied area A_1 as shown in Figure 1.2a-c, where $C_1.certitude[0].geo = C_{11}$, $C_1.certitude[1].geo = C_{12}$ and $C_1.dubiety[0].geo = C_{13}$. Consider also the fact highlighted in Figure 4.8. The vague spatial fact illustrated in Figure 4.9 associates partial values of the measures $^{numeric}AppliedTons$ and $^{vague}AppliedArea$ to the certitude and the dubiety of $^{vague}Crop = C_1$, where $PesticideId = 121$ and $Date = '2012-10-04'$. In the matrix, the row $cell_{0,j}$ denotes how many tons of pesticides were applied ($^{numeric}AppliedTons$), while the row $cell_{1,j}$ denotes the extent of the applied area over the corresponding part of C_1 ($^{vague}AppliedArea$). Therefore, $cell_{0,0}$ is 0.25 and $cell_{1,0}$ is the extent of A_1 that intersects C_{11} , as shown in Figure 1.2b. Furthermore, $cell_{0,1}$ is 0.00 and $cell_{1,1}$ is empty since A_1 does not intersect C_{12} , as shown in Figure 1.2b. Moreover, $cell_{0,2}$ is 0.05 and $cell_{1,2}$ is the extent of A_1 that intersects C_{13} , as shown in Figure 1.2c. Applying the aggregation function SUM to $cell_{0,0}$ and $cell_{0,1}$ produces $AppliedTons = 0.25$ for the certitude of C_1 . Conversely, applying SUM to $cell_{0,2}$ produces $AppliedTons = 0.05$ for the dubiety of C_1 . In addition, applying vague spatial union $VUnion$ to $cell_{1,0}$ and $cell_{1,1}$ produces the vague region of the applied area regarding the certitude of C_1 . Conversely, applying $VUnion$ to $cell_{1,2}$ produces the vague region of the applied area over the dubiety of C_1 .

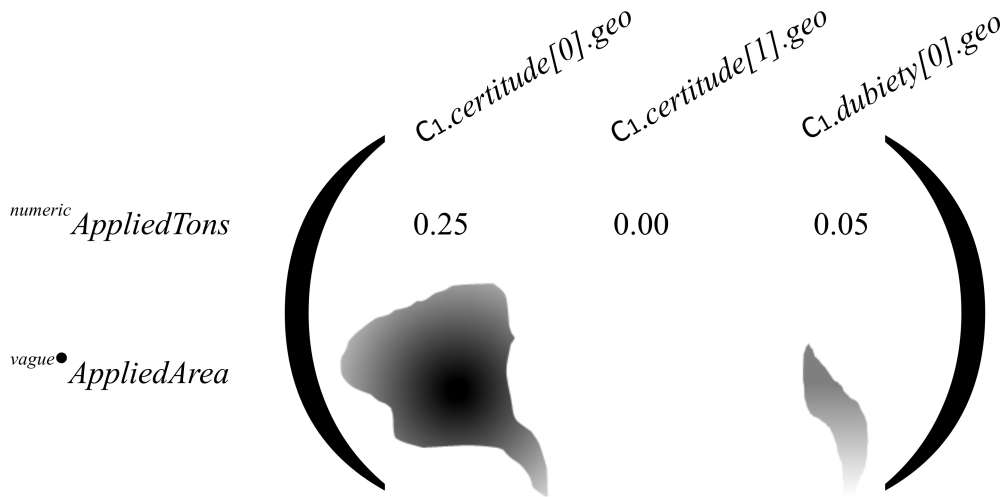


Figure 4.9: The matrix of a vague spatial fact.

Example 4.4.5. The vague spatial fact depicted in Figure 4.9 can aid to answer the query PC2 described in Table 1.1. If parts that certainly belong to the crop C_1 are required, then $cell_{0,0}=0.25$ and $cell_{0,1}=0.00$ provide the amount of pesticides, while $cell_{1,0}$ and $cell_{1,1}$ provide the areas of application. Conversely, if parts that possibly belong to the crop C_1 are required, then $cell_{0,2}$ supplies the amount of pesticides, while $cell_{1,2}$ supplies the area of application.

The logical design of the vague spatial fact is described in Section 5.7.

4.4.5 Lattice of Cuboids

Cuboids with distinct granularities are organized in a lattice to enable the aggregation of the values of measures. The lattice of cuboids provides levels of summarized data, such that hierarchies play a key role to determine such organization.

Definition 4.4.7. A lattice of cuboids L is a directed acyclic graph that imposes a partial order on the set of cuboids of a given cube, and is a tuple denoted $L = \langle C, c_0, \dots, c_z, E \rangle$ where:

- C is the cube that defines the lattice;
- c_i is a cuboid of C , and therefore is a node of the lattice ($0 \leq i \leq z$);
- $c_i = c_{bottom}$ is the bottom-level node, i.e. a cuboid at the finest granularity that holds the keys of dimensions;
- $c_i = c_{all}$ is the top-level node, i.e. a cuboid at the coarser granularity that does not hold attributes;

- E is a set of edges that link the nodes (cuboids);
- $c_i \rightarrow c_j$ means that there is an edge that links c_i to c_j and that the granularity of c_i is finer than the granularity of c_j ;
- the reflexive and transitive closure of \rightarrow is denoted by \rightarrow^* ;
- $c_i \rightarrow^* c_k$ determines that aggregation functions are applied to values of measures in c_i to obtain summarized values in c_k ;
- \rightarrow^* has a unique bottom level c_{bottom} and a unique top level c_{all} ;
- all cuboids c_i are organized in the graph such that $c_{bottom} \rightarrow^* c_i$ and $c_i \rightarrow^* c_{all}$; and
- each hierarchy $x_0 \preceq \dots \preceq x_n$ is a total order encompassed by L , where $0 \leq b < n$ and c_b and c_{b+1} are distinct cuboids of L , such that

- $x_b \in c_b$,
- $x_{b+1} \in c_{b+1}$,
- $c_{b+1} \rightarrow c_b$, and
- $c_{b+1} \rightarrow^* c_{all}$.

The visual representation for a lattice depicts each cuboid as a node. In addition, the order relation that enables data aggregation is represented by a directed edge.

Example 4.4.6. Let $L_{PesticideApplication}$ be a lattice of cuboids regarding the cube $C_{PesticideApplication}$. A subset of $L_{PesticideApplication}$ is depicted in Figure 4.10, considering only the keys of the dimensions in $C_{PesticideApplication}$. The cuboid with the finest granularity is:

$$c_{bottom} = \langle C_{PesticideApplication}, \overset{numeric}{PesticideId}, \overset{date}{Date}, \overset{vague}{Crop} \rangle.$$

Note that $c_{bottom} = c_0$ as described in Example 4.4.3 and that the cuboid *all* does not hold attributes.

Example 4.4.7. Figure 4.11a illustrates another subset of $L_{PesticideApplication}$. Each edge traverses a hierarchy, aggregates values of measures using aggregation functions and points to a cuboid with a coarser granularity. The multidimensional view provided by each cuboid is shown in Figure 4.11b, where the visual effect of data aggregation correspond to merging cells that represent facts. The geographical view provided by facts of each cuboid is shown in Figure 4.11c. Areas where pesticides were applied to overlap either crops or agricultural lands. For the sake of simplicity, only areas of application of the pesticide with $PesticideId = 121$ are shown. For each cuboid, facts and the aggregation of measure values are listed in Figure 4.11d,

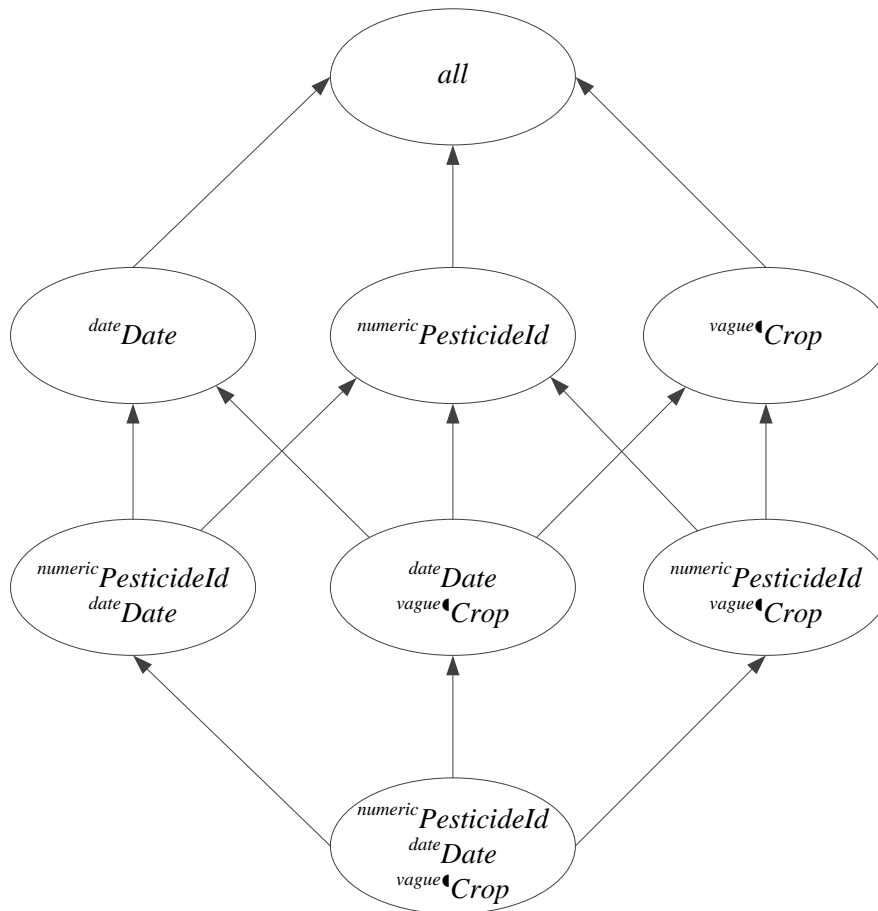


Figure 4.10: Lattice of cuboids $L_{PesticideApplication}$, represented according to the bottom-up notation of Ciferri et al. (2013).

such that $PesticideId = 121$. An edge usually implies aggregating applied areas and applied tons by using $VUnion$ and SUM , respectively. It is noteworthy that the cuboid at the bottom can be used to answer the query PC1 shown in Table 1.1. In addition, the cuboid at the top can be used to answer the queries PC3 and PC4 shown in Table 1.1.

4.5 Vague Spatial Predicates

Range queries had their importance and utility for SDW already discussed in Section 2.1.1.3. They are also important for vague SDW to select vague spatial objects from the dataset according to a topological relationship, as exemplified by the query PC4 shown in Table 1.1. Sections 4.5.1 and 4.5.2 address range queries defined by the VSCube conceptual model. In addition to assess a topological relationship against whole objects, VSCube model's range queries assess objects' certitude, dubiety, certitude elements, dubiety elements, and dubiety elements

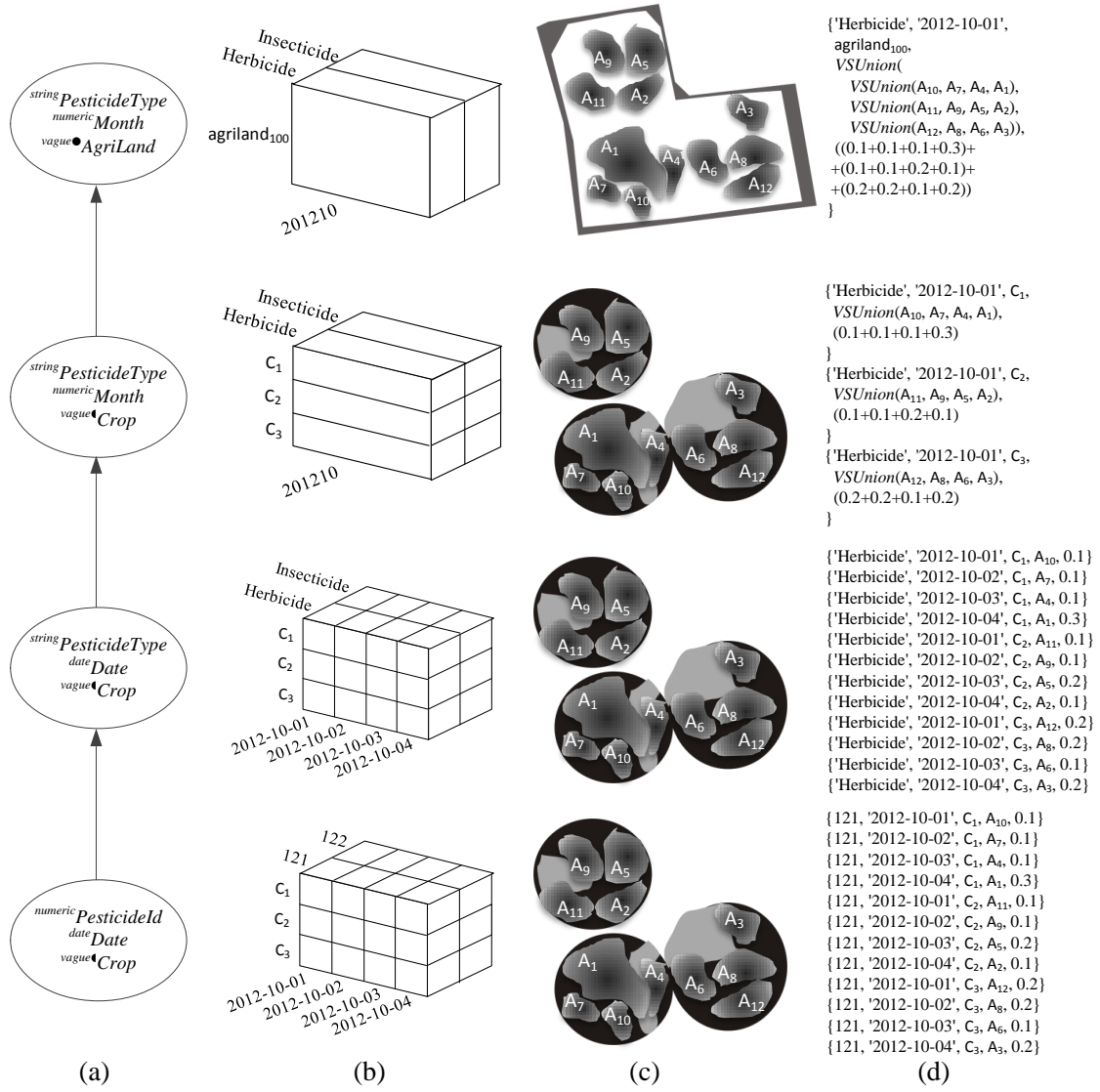


Figure 4.11: A subset of the lattice $L_{PesticideApplication}$. (a) Cuboids. (b) The multidimensional view. (c) The geographic view. (d) Data aggregation.

with their membership values.

4.5.1 Spatial Range Queries

To select vague spatial objects or their elements according to a topological relationship, the VSCube conceptual model provides the following intersection range queries (IRQ) and containment range queries (CRQ):

- IRQ_{object} and CRQ_{object} focus on objects;
- $IRQ_{certitude}$ and $CRQ_{certitude}$ focus on objects' certitude;

- $IRQ_{dubiety}$ and $CRQ_{dubiety}$ focus on objects' dubiety;
- $IRQ_{dubiety-mval}$ and $CRQ_{dubiety-mval}$ checks whether dubiety elements satisfy a condition regarding their membership values;
- $IRQ_{certitude_elements}$ and $CRQ_{certitude_elements}$ retrieve certitude elements that satisfy the topological relationship;
- $IRQ_{dubiety_elements}$ and $CRQ_{dubiety_elements}$ retrieve dubiety elements that satisfy the topological relationship; and
- $IRQ_{dubiety_elements-mval}$ and $CRQ_{dubiety_elements-mval}$ retrieve dubiety elements that satisfy the topological relationship and a condition regarding their membership values.

These queries return a non-empty set if at least one vague spatial object satisfies the topological relationship, or return an empty set otherwise.

Definition 4.5.1. Considering that q is a spatial query window and $dataset$ is a set of vague spatial objects, the range queries issued against whole objects held by the dataset are:

$$IRQ_{object}(q, dataset) = \{z | z \in dataset \wedge z.certitude.geo \cap q \neq \emptyset \vee z.dubiety.geo \cap q \neq \emptyset\};$$

$$and$$

$$CRQ_{object}(q, dataset) = \{z | z \in dataset \wedge z.certitude.geo \cap q = z.certitude.geo \wedge z.dubiety.geo \cap q = z.dubiety.geo\}.$$

Example 4.5.1. In Figure 4.12, let z be an area where pesticides were applied to. An IRQ_{object} retrieves the vague spatial object z as z intersects q . Conversely, a CRQ_{object} returns an empty set because z is not within q .

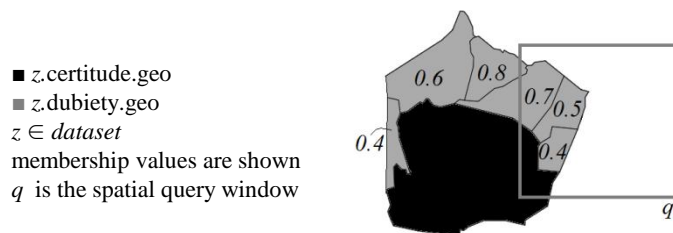


Figure 4.12: The vague spatial object z and the spatial query window q .

Definition 4.5.2. Considering that q is a spatial query window and $dataset$ is a set of vague spatial objects, the range queries issued against the certitude or against the dubiety of vague spatial objects held by the dataset are, respectively:

$$IRQ_{certitude}(q, dataset) = \{z | z \in dataset \wedge z.certitude.geo \cap q \neq \emptyset\};$$

$$IRQ_{dubiety}(q, dataset) = \{z | z \in dataset \wedge z.dubiety.geo \cap q \neq \emptyset\};$$

$$CRQ_{certitude}(q, dataset) = \{z | z \in dataset \wedge z.certitude.geo \cap q = z.certitude.geo\}; \text{ and}$$

$$CRQ_{dubiety}(q, dataset) = \{z | z \in dataset \wedge z.dubiety.geo \cap q = z.dubiety.geo\}.$$

Besides, the membership value α can be considered to filter objects whose dubiety has all elements complying with an expression, for $\Phi \in \{=, \neq, >, <, \leq, \geq\}$:

$$IRQ_{dubiety-mval}(q, dataset, \Phi, \alpha) = \{z |$$

$$z \in dataset \wedge z.dubiety.geo \cap q \neq \emptyset \wedge \forall j \geq 0, z.dubiety[j].mval \Phi \alpha\}; \text{ and}$$

$$CRQ_{dubiety-mval}(q, dataset, \Phi, \alpha) = \{z |$$

$$z \in dataset \wedge z.dubiety.geo \cap q = z.dubiety.geo \wedge \forall j \geq 0, z.dubiety[j].mval \Phi \alpha\}.$$

Example 4.5.2. In Figure 4.12, let z be an area where pesticides were applied to. An $IRQ_{certitude}$ returns the vague spatial object z because q intersects the certitude of z . The result is the same for an $IRQ_{dubiety}$ as q intersects the dubiety of z . On the other hand, suppose an $IRQ_{dubiety-mval}$ is issued with Φ being ‘greater than’ (i.e. $>$) and $\alpha = 0.5$. The result for such $IRQ_{dubiety-mval}$ is empty, since there are dubiety elements of z that do not have membership value greater than 0.5. Moreover, both a $CRQ_{certitude}$ and a $CRQ_{dubiety}$ return an empty set, since z , its certitude, and its dubiety are not within q . Analogously, a $CRQ_{dubiety-mval}$ also returns an empty set independent from the membership value provided for α , since z is not within the q .

Note that $IRQ_{certitude}$ is the vague spatial predicate in the query PC4 listed in Table 1.1. The condition “certainly intersect” implies that an applied area and the spatial query window intersect with *certitude*. To satisfy such condition, it must be ensured that the certitude of the applied area intersects the certitude of the spatial query window. Since the former is vague and the latter is crisp, then it is sufficient to assess whether the certitude of the applied area intersects the spatial query window. Note that other spatial range queries from Definition 4.5.2 could be used instead of $IRQ_{certitude}$, by rewriting the query PC4.

Definition 4.5.3. Considering that q is a spatial query window and $dataset$ is a set of vague spatial objects, the range queries issued against certitude elements or against dubiety elements of vague spatial objects held by the dataset are, respectively:

$$IRQ_{certitude_elements}(q, dataset) = \{o | z \in dataset \wedge o \in z.certitude.geo \wedge o \cap q \neq \emptyset\};$$

$$IRQ_{dubiety_elements}(q, dataset) = \{o | z \in dataset \wedge o \in z.dubiety.geo \wedge o \cap q \neq \emptyset\};$$

$$CRQ_{certitude_elements}(q, dataset) = \{o | z \in dataset \wedge o \in z.certitude.geo \wedge o \cap q = o\}; \text{ and}$$

$$CRQ_{dubiety_elements}(q, dataset) = \{o | z \in dataset \wedge o \in z.dubiety.geo \wedge o \cap q = o\}.$$

Besides, the membership value α can be considered to filter dubiety elements, for $\Phi \in \{=, \neq, >, <, \leq, \geq\}$ and $j \geq 0$:

$$IRQ_{dubiety_elements-mval}(q, dataset, \Phi, \alpha) = \{o \mid z \in dataset \wedge o \in z.dubiety.geo \wedge o = z.dubiety[j].geo \wedge o \cap q \neq \emptyset \wedge z.dubiety[j].mval \Phi \alpha\};$$

$$CRQ_{dubiety_elements-mval}(q, dataset, \Phi, \alpha) = \{o \mid z \in dataset \wedge o \in z.dubiety.geo \wedge o = z.dubiety[j].geo \wedge o \cap q = o \wedge z.dubiety[j].mval \Phi \alpha\}.$$

Example 4.5.3. In Figure 4.12, let z be an area where pesticides were applied to. Suppose an $IRQ_{certitude_elements}$ is issued. Since q intersects the single certitude element of z , this element is returned, as shown in Figure 4.13a. If an $IRQ_{dubiety_elements}$ is issued, four dubiety elements of z are returned, as shown in Figure 4.13b. If an $IRQ_{dubiety_elements-mval}$ is issued with Φ being ‘greater than’ (i.e. $>$) and $\alpha = 0.5$, then the result has two dubiety elements of z , as shown in Figure 4.13c. Furthermore, a $CRQ_{certitude_elements}$ returns an empty set because the single certitude element of z is not within q . A $CRQ_{dubiety_elements}$ returns the elements with membership value 0.5 and 0.4, since their geometries are within q . Finally, a $CRQ_{dubiety_elements-mval}(q, z, \geq, 0.5)$ returns the element with membership value 0.5 since its geometry is within q and complies with the expression.

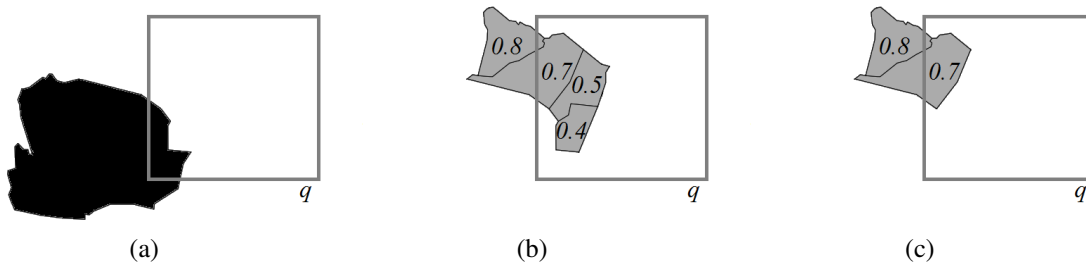


Figure 4.13: The results for $IRQ_{certitude_elements}$, $IRQ_{dubiety_elements}$ and $IRQ_{dubiety_elements-mval}$. (a) $IRQ_{certitude_elements}$. (b) $IRQ_{dubiety_elements}$. (c) $IRQ_{dubiety_elements-mval}$.

The implementation of spatial range queries is described in Section 5.9.2.

4.5.2 The Vague Spatial Range Query

Both IRQs and CRQs explained in Section 4.5.1 considered vague spatial objects from the dataset against the crisp spatial query window q . To allow ranking the results of a query according to relevance, the VSCube conceptual model also enables the use of a vague spatial query window composed of a pair of bi-dimensional concentric rectangles. As a result, the spatial vagueness concerns the spatial query window, while the dataset can be composed of crisp or vague spatial data. The *vague spatial range query* (VSRQ) identifies all crisp spatial objects or vague spatial objects such that:

- the inner spatial query window evaluates a CRQ, to be more restrictive and consequently denote a lower degree of vagueness, producing more relevant results; and
- the outer spatial query window evaluates an IRQ, to be less restrictive and gradually denote a greater degree of vagueness, producing less relevant results by subtracting the results already gathered by the inner spatial query window.

Definition 4.5.4. Considering that *inner* and *outer* are concentric spatial query windows and *dataset* is a set of spatial objects, the vague spatial range query issued against whole objects in the dataset is:

$$VSRQ_{object}(inner, outer, dataset) = CRQ_{object}(inner, dataset) \cup (IRQ_{object}(outer, dataset) - CRQ_{object}(inner, dataset))$$

The $VSRQ_{certitude}$, $VSRQ_{dubiety}$, $VSRQ_{dubiety-mval}$, $VSRQ_{certitude-elements}$, $VSRQ_{dubiety-elements}$ and $VSRQ_{dubiety-elements-mval}$ are defined analogously.

Example 4.5.4. In Figure 4.14a, a $VSRQ_{object}(inner, outer, dataset)$ is issued against watersheds of the pest control case study, considering the spatial query windows *inner* and *outer*. The result set of the $VSRQ_{object}$ is shown in Figure 4.14b, where cities in black are those within *inner* and have a higher priority. Conversely, cities in gray intersect *outer* and are not within *inner*, and thus have a lower priority.

Crisp spatial objects were used as dataset in the example for the sake of simplicity. However, vague spatial objects could be used instead.

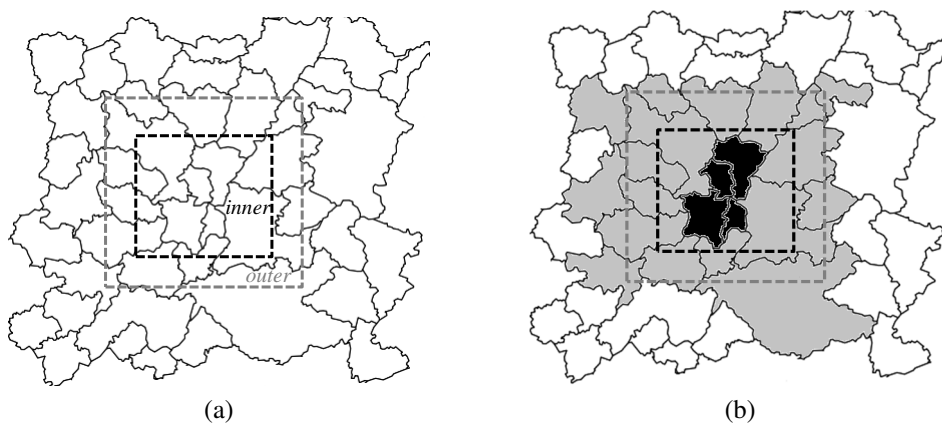


Figure 4.14: A $VSRQ_{object}$ against crisp watersheds. (a) Watersheds and spatial query windows. (b) Result set.

The implementation of the vague spatial range query is addressed in Section 5.9.2.

4.6 Vague Spatial Aggregation Functions

The subjects of analysis in a SDW are mostly numeric measures and crisp spatial measures as explained in Section 2.1, which are summarized by proper aggregation functions, such as SUM for a numeric measure and union for a crisp spatial measure. Additionally, a vague SDW has vague spatial measures detailed in Section 4.4.2 that require vague spatial aggregation functions. Since vague spatial objects consist of certitude and dubiety geometries and dubiety membership values, the vague spatial aggregation functions perform both geometric and numeric aggregations. The VSCube Conceptual Model offers three vague spatial aggregation functions: the vague spatial union is described in Section 4.6.1, the vague spatial intersection is detailed in Section 4.6.2 and the vague spatial difference is addressed in Section 4.6.3.

4.6.1 Vague Spatial Union

The pairwise union of two input vague spatial objects x and y merges their shapes and creates a new vague spatial object z . The certitude of z is composed by the union of the certitudes of both x and y . The dubiety of z comprises the union of the certitudes of both x and y , except any part that already belong to the certitude of z . This exception considers that the certitude of z already contains any intersection between the certitude of one input object with the dubiety of the other input object. If there is intersection between the dubieties of x and y , then such intersection belongs to the dubiety of z . In addition, the greatest value of membership value is yielded. Regarding the portions in the dubiety of x or y that do not intersect any part of the other input object, they also belong to the dubiety of z and maintain their original membership values.

Definition 4.6.1. The vague spatial union of two input vague spatial objects x and y is:

$$\begin{aligned}
 VSUnionXY(x,y) = \{z | \\
 & z.certitude.geo = (x.certitude.geo \cup y.certitude.geo) \wedge \\
 & z.dubiety.geo = (x.dubiety.geo \cup y.dubiety.geo) - z.certitude.geo \wedge \\
 & \text{for } i, j, k \geq 0, z.dubiety[k].mval \text{ is} \\
 & \quad \max(x.dubiety[i].mval, y.dubiety[j].mval), \text{ iff } x.dubiety[i].geo \cap y.dubiety[j].geo \neq \emptyset \\
 & \quad x.dubiety[i].mval, \text{ iff } \forall k \geq 0 \text{ implies } z.dubiety[k].geo \cap y.dubiety[j].geo = \emptyset; \\
 & \quad y.dubiety[j].mval, \text{ iff } k \geq 0 \text{ implies } z.dubiety[k].geo \cap x.dubiety[i].geo = \emptyset; \text{ or} \\
 & \}
 \end{aligned}$$

The vague spatial union of a collection of vague spatial objects composes a single vague spatial object z and processes input vague spatial objects recursively using $VSUnionXY$.

Definition 4.6.2. The vague spatial union of a collection of vague spatial objects is:

$$VSUnion(o_0, \dots, o_n) = VSUnionXY(o_n, VSUnionXY(o_{n-1}, \dots VSUnionXY(o_1, o_0) \dots)),$$

where o_i is a vague spatial object and $0 < i \leq n$.

Example 4.6.1. Let x and y be distinct areas where pesticides were applied to. Both x and y are depicted in Figure 4.15a and $VSUnion(x, y)$ returns a vague spatial object that is assigned to z in Figure 4.15b. Different tones of gray highlight the different membership values associated to the dubiety.

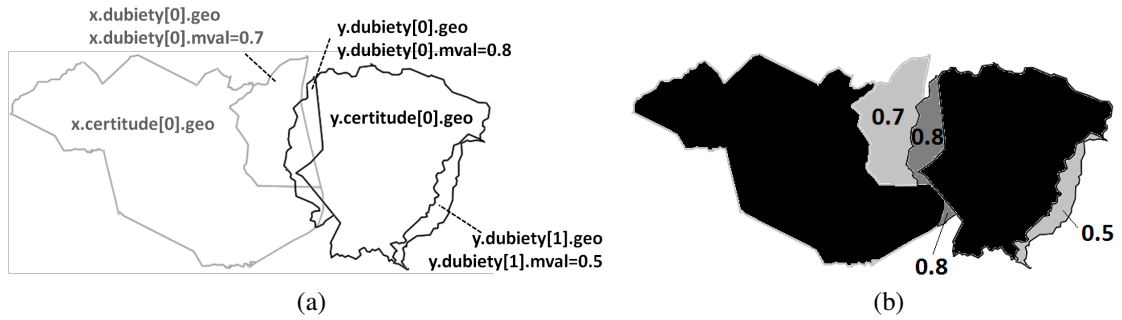


Figure 4.15: The vague spatial union applied to the vague spatial objects x and y . (a) x and y . (b) $z = VSUnion(x, y)$.

Note that $VSUnion$ is used in queries PC2, PC3, and PC4 as shown in Table 1.1 to aggregate areas where pesticides were applied. The implementation of $VSUnion$ is detailed in Section 5.9.3.

4.6.2 Vague Spatial Intersection

The pairwise intersection of two vague spatial objects x and y produces the vague spatial object z containing the extent shared by both x and y . The intersection of the certitudes of x and y belongs to the certitude of z . Conversely, the intersection between the dubiety of x or y with any other part of the other input object belongs to the dubiety of z and holds the lower membership value involved. Finally, the parts of x that do not intersect any part of y , and the parts of y that do not intersect any part of x are discarded. The membership value is assumed to be 1.0 for the certitudes of x and y .

Definition 4.6.3. The vague spatial intersection of two input vague spatial objects x and y is:

$$VSIntersectionXY(x, y) = \{z\}$$

$$z.certitude.geo = (x.certitude.geo \cap y.certitude.geo) \wedge$$

$$z.dubiety.geo = ((x.certitude.geo \cap y.dubiety.geo) \cup$$

$$(x.dubiety.geo \cap y.certitude.geo) \cup$$

$$(x.dubiety.geo \cap y.dubiety.geo) \wedge$$

for $i, j, k \geq 0$, $z.dubiety[k].mval$ is

$$y.dubiety[j].mval, \text{ iff } x.certitude[i].geo \cap y.dubiety[j].geo \neq \emptyset; \text{ or}$$

$$x.dubiety[i].mval, \text{ iff } x.dubiety[i].geo \cap y.certitude[j].geo \neq \emptyset; \text{ or}$$

$$\min(x.dubiety[i].mval, y.dubiety[j].mval), \text{ iff } x.dubiety[i].geo \cap y.dubiety[j].geo \neq \emptyset$$

}

The vague spatial intersection of a collection of vague spatial objects composes a single vague spatial object z and processes input vague spatial objects recursively using $VSIIntersectionXY$.

Definition 4.6.4. The vague spatial intersection of a collection of vague spatial objects is:

$$VSIIntersection(o_0, \dots, o_n) = VSIIntersectionXY(o_n, VSIIntersectionXY(o_{n-1}, \dots, VSIIntersectionXY(o_1, o_0) \dots)),$$

where o_i is a vague spatial object and $0 < i \leq n$.

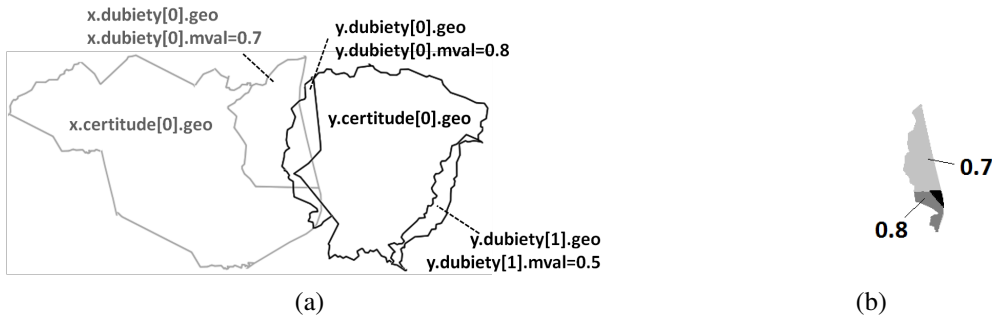


Figure 4.16: The vague spatial intersection applied to the vague spatial objects x and y . (a) x and y . (b) $z = VSIIntersection(x, y)$.

Example 4.6.2. Consider that x and y are distinct areas where pesticides were applied to. Both x and y are depicted in Figure 4.16a and $VSIIntersection(x, y)$ returns a vague spatial object that is assigned to z in Figure 4.16b. Different tones of gray highlight the different membership values associated to the dubiety.

4.6.3 Vague Spatial Difference

The pairwise difference of two vague spatial objects x and y produces the vague spatial object z containing the extent in x that does not belong to y . The portions in the certitude of x that do not intersect any part of y belong to the certitude of z . The portions in the dubiety of x that do not intersect any part of y belong to the dubiety of z and maintain their original membership values. Conversely, the portions in the certitude of x that intersect the dubiety of y belong to the dubiety of z and hold the membership value of 1.0 minus the original membership

value concerning y . In addition, the portions of the dubieties of x and y that intersect belong to the dubiety of z only if the membership value in x minus the membership value in y yields a positive value.

Definition 4.6.5. The vague spatial difference of two input vague spatial objects x and y is:

$$VSDifferenceXY(x,y) = \{z|$$

$$z.certitude.geo = x.certitude.geo - (x.certitude.geo \cap y.certitude.geo)$$

$$- (x.certitude.geo \cap y.dubiety.geo) \wedge$$

$$z.dubiety.geo = (x.dubiety.geo - (x.dubiety.geo \cap y.certitude.geo)$$

$$- (x.dubiety.geo \cap^- y.dubiety.geo))$$

$$\cup (x.certitude.geo \cap y.dubiety.geo) \wedge$$

for $i, j, k \geq 0$, $z.dubiety[k].mval$ is

$$1.0 - y.dubiety[j].mval, \text{ iff } x.certitude[i].geo \cap y.dubiety[j].geo \neq \emptyset; \text{ or}$$

$$x.dubiety[i].mval - y.dubiety[j].mval, \text{ otherwise}$$

$\}$, where the operator \cap^- retrieves the intersections between $x.dubiety.geo$ and $y.dubiety.geo$ for $i, j \geq 0$ and $x.dubiety[i].mval - y.dubiety[j].mval \leq 0$.

The vague spatial difference of a collection of vague spatial objects composes a single vague spatial object z and processes input vague spatial objects recursively using $VSDifferenceXY$.

Definition 4.6.6. The vague spatial difference of a collection of vague spatial objects is:

$$VSDifference(o_0, \dots, o_n) = VSDifferenceXY(o_n, VSDifferenceXY(o_{n-1}, \dots$$

$$VSDifferenceXY(o_1, o_0) \dots)), \text{ where } o_i \text{ is a vague spatial object and } 0 < i \leq n.$$

Example 4.6.3. Consider that x and y are distinct areas where pesticides were applied to. Both x and y are depicted in Figure 4.17a and $VSDifference(x,y)$ returns a vague spatial object that is assigned to z in Figure 4.17b. Different tones of gray highlight the different membership values associated to the dubiety.

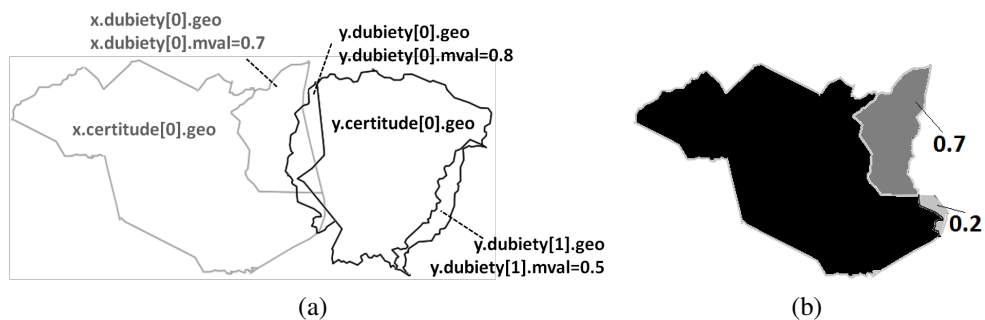


Figure 4.17: The vague spatial difference applied to the vague spatial objects x and y . (a) x and y . (b) $z = VSDifference(x,y)$.

4.7 Vague Spatial Online Analytical Processing

The vague spatial online analytical processing (vague SOLAP) comprises a set of complex multidimensional queries described in this section, extended with vague spatial aggregation functions detailed in Section 4.6 and the vague spatial predicates described in Section 4.5.

The *roll-up* operation progressively fetches data according to attributes of a coarser granularity. It often fetches upper levels of a given hierarchy by requiring attributes of linked cuboids. It relies upon aggregation, and therefore functions detailed in Section 4.6 can be used to aggregate vague spatial data. Conversely, the *drill-down* operation progressively fetches data according to attributes of a finer granularity. It usually fetches lower levels of a given hierarchy by requiring attributes of linked cuboids.

Definition 4.7.1. Let L be a lattice of cuboids such that $L = \langle C, c_0, \dots, c_z, E \rangle$. The *roll-up* operation is denoted $roll-up(c_{finer}, c_{coarser})$ and corresponds to firstly issue a query against the cuboid c_{finer} , and subsequently issue a query against the cuboid $c_{coarser}$, such that there is at least one way whose edges link c_{finer} and $c_{coarser}$, i.e. $c_{finer} \rightarrow^* c_{coarser}$.

Definition 4.7.2. Let L be a lattice of cuboids such that $L = \langle C, c_0, \dots, c_z, E \rangle$. The *drill-down* operation is denoted $drill-down(c_{coarser}, c_{finer})$ and corresponds to firstly issue a query against the cuboid $c_{coarser}$, and subsequently issue a query against the cuboid c_{finer} , such that there is at least one way whose edges link c_{finer} and $c_{coarser}$, i.e. $c_{finer} \rightarrow^* c_{coarser}$.

Roll-up is the inverse of drill-down. Both can be performed not only against attributes that are adjacent in a hierarchy. In addition, they are not limited to geometries of the same dimensionality between the hierarchy levels. Instead, they may change the dimensionality of geometries along the hierarchy of a spatial dimension, e.g. 2D to 0D and vice-versa. Sometimes the change occurs more than once and sometimes with attributes that are not geometry.

Example 4.7.1. Consider the following cuboids shown in Figure 4.11a:

$c_{bottom} = \langle C_{PesticideApplication},^{numeric} PesticideId,^{date} Date,^{vague} \blacktriangleright Crop \rangle$, and

$c_{tma} = \langle C_{PesticideApplication},^{string} PesticideType,^{numeric} Month,^{vague} \bullet AgriLand \rangle$.

Note that $c_{bottom} \rightarrow^* c_{tma}$. Recall that c_{bottom} and c_{tma} can be used to answer the queries PC1 and PC3 shown in Table 1.1, respectively. Then, $roll-up(c_{bottom}, c_{tma})$ corresponds to firstly issue the query PC1 against c_{bottom} and thereafter issue the query PC3 against c_{tma} . Conversely, $drill-down(c_{tma}, c_{bottom})$ consists of corresponds to firstly issue the query PC3 against c_{tma} and thereafter issue the query PC1 against c_{bottom} .

The *slice-and-dice* operation retrieves data that comply with specific criteria. It relies upon

selection, and therefore predicates detailed in Section 4.5 can be used to retrieve vague spatial data.

Definition 4.7.3. Let c be a cuboid such that $c = \langle C, TypeAttr^0 x_0, \dots, TypeAttr^t x_t \rangle$. The *slice-and-dice* operation determines a *slice* to match fixed values of at least one attribute x_i of the cuboid c , and a *dice* to match ranges for at least one of the remaining attributes x_j of the cuboid c ($i \neq j$).

Example 4.7.2. Consider the following cuboid shown in Figure 4.11:

$$c_{tma} = \langle C_{PesticideApplication},^{string} PesticideType,^{numeric} Month,^{vague} \bullet AgriLand \rangle.$$

Recall that c_{tma} can be used to answer the query PC4 shown in Table 1.1. The *slice* on c_{tma} sets the year as 2012 and the pesticide type as herbicide. The *dice* on c_{tma} is a range of vague spatial objects fetched by an $IRQ_{certitude}$ against agricultural lands.

In addition to allow data summarization, detailing, and selection, vague SOLAP also enables the *pivot* operator that aims at visualization, as follows.

Definition 4.7.4. Let c be a cuboid such that $c = \langle C, TypeAttr^0 x_0, \dots, TypeAttr^t x_t \rangle$. The pivot operation provides interactive visualization of the cuboid c by switching the axis of at least two attributes.

Example 4.7.3. In Figure 4.8, the cuboid c_0 has the following axis: $x = PesticideId$, $y =^{vague} \blacktriangleright Crop$ and $z = Year$. A pivot operation was applied on the cuboid c_0 , as shown in Figure 4.18, obtaining the following axis: $x = Year$, $y =^{vague} \blacktriangleright Crop$ and $z = PesticideId$.

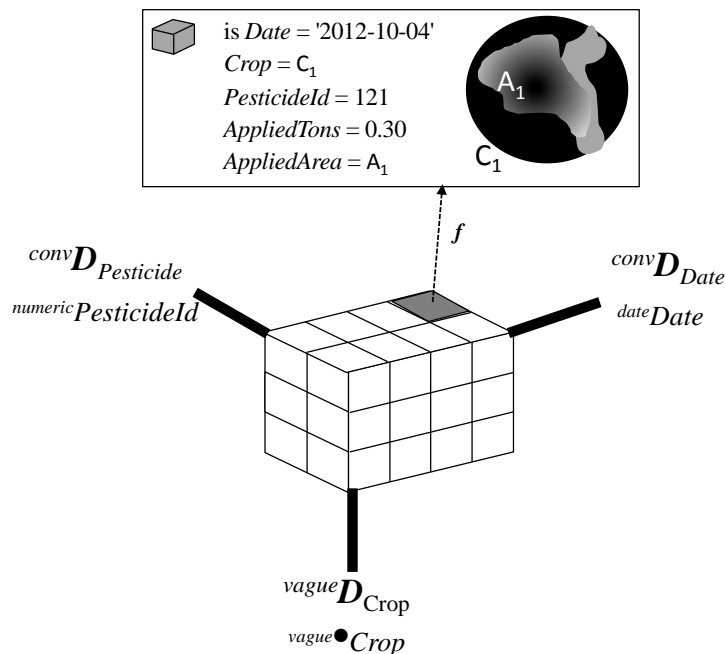


Figure 4.18: A pivoted cuboid.

Queries using the operators described in this section are detailed in Section 5.9.

4.8 Reusing Existing Models and Implementations

Existing models and implementations for vague spatial data surveyed in Section 2.3 are not aimed at vague SDW, but can be reused in the VSCube conceptual model with a few adaptations and according to the shapes they use to represent spatial vagueness, as follows. Section 4.8.1 focuses the representation using arbitrary geometries with: (i) non-disjoint interiors as employed by QMM, RBB and Egg-Yolk; (ii) disjoint interiors as adopted by VASA; (iii) disjoint interiors and a precomputed membership values as tackled by the Plateau Algebra. Section 4.8.2 focuses the Bitmap that uses rectangular grid cells or hexagonal grid cells with disjoint interior, each cell holding a precomputed membership value. Section 4.8.3 focuses on TIN that uses triangles whose interiors are disjoint interiors and whose vertices hold membership values. Finally, Section 4.8.4 focuses linear segments holding membership values.

4.8.1 Arbitrary Geometries

Considering that according to QMM, L_{max} is the maximal extent of a vague spatial object z and L_{min} is the minimal extent of z , then the following assignments create a vague spatial object o compatible with the VSCube conceptual model:

$$o.certitude[0].geo \leftarrow L_{min}, \text{ and}$$

$$o.dubiety[0].geo \leftarrow L_{max} - L_{min}, \text{ where } - \text{ is the geometric difference.}$$

QMM encompasses the RBB and Egg-Yolk and, therefore, the same assignments are applied for these models. The cited subtraction is essential to enable geometric data aggregation of vague spatial data, which play a key role in the processing of values of a vague spatial measure. Note that the aforementioned models do not define operators for union, intersection and difference. The cited subtraction is also essential to enable vague spatial predicates, since those models also do not provide spatial predicates to retrieve parts of vague spatial objects.

If z is an object modeled according to VASA, then *kernel* is the kernel of z , and *conjecture* is the conjecture of z . Two approaches can be used to create a vague spatial object o compatible with the VSCube conceptual model, as follows. The first has the following trivial assignments:

$$o.certitude[0].geo \leftarrow kernel \text{ and}$$

$$o.dubiety[0].geo \leftarrow conjecture.$$

The second pair of assignments identifies i elements in kernel and j elements in conjecture:

$$o.certitude[i].geo \leftarrow kernel[i] \text{ and}$$

$$o.dubiety[j].geo \leftarrow conjecture[j], \forall i, j \geq 0.$$

The first pair of assignments allows querying entire objects, only certitudes and only dubieties, while the second pair assignments allows querying whole objects, only certitudes, only dubieties, elements of the certitude and elements of the dubiety. Both pairs of assignments enable geometric aggregation of vague spatial data.

The aforementioned models do not support membership values, and then spatial predicates involving membership values are not enabled. On the other hand, if z is a vague spatial object designed according to the Plateau Algebra, then it is composed by i pairs of geometry and membership value. If $z[i].geo$ is the i -th geometry and $z[i].mval$ is the i -th membership value, then the following assignments apply to create a vague spatial object o compatible with the VSCube conceptual model to enable both the aggregation of vague spatial data and all vague spatial predicates, for $c, d, i \geq 0$:

$$o.certitude[c].geo \leftarrow z[i].geo \text{ if } z[i].mval = 1.0; \text{ or}$$

$$o.dubiety[d].geo \leftarrow z[i].geo \text{ otherwise.}$$

4.8.2 Bitmaps

The same assignments described for the Plateau Algebra described in Section 4.8.1 are valid if z is a bitmap with i cells $z[i]$ that have a geometry $z[i].geo$ and a corresponding membership value $z[i].mval$. The vague spatial predicates defined the VSCube conceptual model are allowed. The operations of union, intersection and difference adapt the grid of cells of the resultant bitmap. Therefore, the guidelines described by Verstraete, Tré & Hallez (2006) should be incorporated in the aggregation of vague spatial data.

4.8.3 Triangulations

Considering that T is a TIN for a fuzzy region, the corresponding vague spatial object o compatible with the VSCube conceptual model is created as follows. Let $T[i]$ be the i -th triangle of T , such that $T[i]$ is described by $((x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3))$ where x and y are coordinates and z are membership values. Considering that $\Delta((x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3))$ creates the geometry of a 2D triangle with the vertices $((x_1, y_1), (x_2, y_2), (x_3, y_3))$, then the following assignments apply for $c, d, i \geq 0$:

$$o.certitude[c].geo \leftarrow \Delta((x_1, y_1), (x_2, y_2), (x_3, y_3)), \text{ if } z_1 = z_2 = z_3 = 1.0; \text{ and}$$

$$o.dubiety[d].geo \leftarrow \Delta((x_1, y_1), (x_2, y_2), (x_3, y_3)), \text{ otherwise.}$$

Furthermore, the membership value in $o.dubiety[d].mval$ is not precomputed, but is provided by a function whose codomain is $]0, 1]$, as described in Equation 2.1 (Section 2.3.3.4). To comply with this purpose, (x_1, y_1) , (x_2, y_2) and (x_3, y_3) are the coordinates of the vertices of the triangle $T[i]$ and z_1 , z_2 and z_3 are the membership values associated to each vertex in $T[i]$, respectively. As a result, the function determines the membership value for every point (x, y) of a triangle, including its vertices, and does not denote an empty value. Rather, the value is calculated when demanded to process vague spatial predicates and aggregation of vague spatial data.

For instance, consider the TIN depicted in Figure 4.19a, where two triangles are highlighted and the spatial query window w is displayed. The black triangle t_{black} belongs to the certitude of the vague region, while the gray triangle t_{gray} belongs to the dubiety of the vague region. Both t_{black} and t_{gray} have their vertices described by coordinates and membership values in Figure 4.19b. Note that the triangles share one vertex, i.e. $(102, 125, 1.0)$. Before the assignments, A , B , C and D are calculated as follows:

$$A = 125 \times (0.8 - 0.01) + 119 \times (0.01 - 1.0) + 112 \times (1.0 - 0.8) = 3.34$$

$$B = 1.0 \times (112 - 83) + 0.8 \times (83 - 102) + 0.01 \times (102 - 112) = 13.7$$

$$C = 102 \times (119 - 112) + 112 \times (112 - 125) + 83 \times (125 - 119) = -244$$

$$D = -3.34 \times 102 - 13.7 \times 125 + 244 \times 0.01 = -2050.74.$$

The following fractions are also calculated:

$$-\frac{A}{C} = -\frac{3.34}{-244} = 0.013688525$$

$$-\frac{B}{C} = -\frac{13.7}{-244} = 0.056147541$$

$$-\frac{D}{C} = -\frac{-2050.74}{-244} = -8.404672131$$

Finally, the following assignments are valid:

$$o.certitude[c].geo \leftarrow t_{black} \text{ where } (c \geq 0),$$

$$o.dubiety[d].geo \leftarrow t_{gray} \text{ where } (d \geq 0), \text{ and}$$

$$o.dubiety[d].mval \leftarrow 0.013688525x + 0.056147541y - 8.404672131.$$

The vague spatial predicates of the VSCube conceptual model are enabled as follows. Although the following discussion concerns the $IRQs$, the same concepts are applicable to the $CRQs$. When evaluating an IRQ_{object} , if one vertex of a triangle in $o.certitude.geo$ or in $o.dubiety.geo$ intersects the spatial query window w , then the corresponding vague spatial object

o is an answer. The evaluation of $IRQ_{certitude}$ and $IRQ_{dubiety}$ are done analogously considering the intersection of w against $o.certitude.geo$ and the intersection of w against $o.dubiety.geo$, respectively. For instance, consider the TIN of a vague region and the spatial query window w shown in Figure 4.19a. Both t_{black} and t_{gray} indicate that the vague region is an answer of the IRQ_{object} . Also, t_{black} indicates that the vague region is a answer of the $IRQ_{certitude}$ and t_{gray} indicate that the vague region is a answer of the $IRQ_{dubiety}$.

As for $IRQ_{certitude_elements}$, if at least one vertex of a given triangle in the certitude intersects the spatial query window w then the corresponding triangle is an answer. For example, t_{black} satisfies an $IRQ_{certitude_elements}$ against the spatial query window w as shown in Figure 4.19a. Analogously, when evaluating an $IRQ_{dubiety_elements}$, if least one vertex of a given triangle in the dubiety intersects the spatial query window w then the corresponding triangle is an answer. For example, t_{gray} satisfies an $IRQ_{dubiety_elements}$ against the spatial query window w as shown in Figure 4.19a.

As for $IRQ_{dubiety_elements-mval}$, if at least one vertex of a given triangle in the dubiety intersects the spatial query window w and all vertices comply with the conditional involving the membership value, then the corresponding triangle is an answer. For example, t_{gray} does not satisfy an $IRQ_{dubiety_elements-mval}$ against the spatial query window w for membership values greater than 0.6 in Figure 4.19a, as there is a vertex with membership value 0.01. Analogously, a $IRQ_{dubiety-mval}$ would have an empty result set for the same spatial query window w and membership values greater than 0.6.

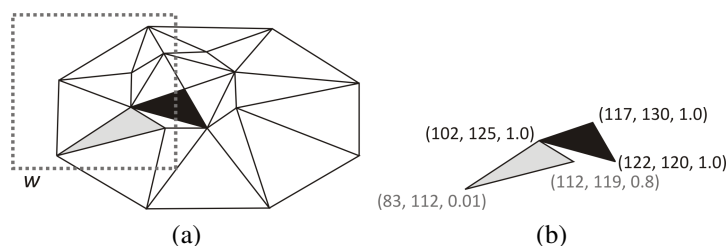


Figure 4.19: A spatial range query against a TIN. (a) A TIN and a spatial query window w . (b) Triangles t_{black} and t_{gray} of the TIN.

The membership values of the vertices of triangles must be calculated not only when evaluating vague spatial predicates on a TIN, but also to allow aggregation of vague spatial data. The operations of union, intersection and difference require retriangulation of the input TINs. Therefore, the guidelines described by Dilo (2006) are incorporated in the aggregation of vague spatial data. Then, the vague spatial object o can be transformed into a TIN using the reverse assignments, considering $T[i]$ is the i -th triangle of a TIN T , such that $T[i]$ is described by $((x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3))$, where x and y are coordinates and z are membership values.

The following reverse assignments to transform a vague spatial object into a TIN apply:

$T[i] \leftarrow ((x_{c1}, y_{c1}, 1.0), (x_{c2}, y_{c2}, 1.0), (x_{c3}, y_{c3}, 1.0))$, where (x_{c1}, y_{c1}) , (x_{c2}, y_{c2}) and (x_{c3}, y_{c3}) are vertices of the triangle in $o.certitude[c].geo$; and

$T[i] \leftarrow ((x_{d1}, y_{d1}, z_1), (x_{d2}, y_{d2}, z_2), (x_{d3}, y_{d3}, z_3))$, where (x_{d1}, y_{d1}) , (x_{d2}, y_{d2}) and (x_{d3}, y_{d3}) are vertices of the triangle in $o.dubiety[d].geo$ and z_1 , z_2 and z_3 are their membership values, respectively.

4.8.4 Lines with Gradual Transitions

Lines with gradual transitions are reused as they implement fuzzy lines. Considering that l is a line with gradual transition, the corresponding vague spatial object o compatible with the VSCube conceptual model is created as follows. Let $l[i]$ be the i -th segment of l , such that $l[i]$ is described by $((x_1, y_1, z_1), (x_2, y_2, z_2))$ where x and y are coordinates and z are membership values. The following assignments apply considering that $Line((x_1, y_1), (x_2, y_2))$ creates the geometry of a rectilinear line with the endpoints $(x_1, y_1), (x_2, y_2)$:

$o.certitude[c].geo \leftarrow Line((x_1, y_1), (x_2, y_2))$, if $z_1 = z_2 = 1.0$; and

$o.dubiety[d].geo \leftarrow Line((x_1, y_1), (x_2, y_2))$, otherwise.

Furthermore, the membership value in $o.dubiety[d].mval$, is not precomputed, but is given by the linear interpolation on $((x_1, y_1, z_1), (x_2, y_2, z_2))$.

The vague spatial predicates of the VSCube conceptual model are enabled like those described in Section 4.8.3. The operations of union, intersection and difference may require splitting of lines used as input. Therefore, the guidelines described by Dilo (2006) are incorporated in the aggregation of vague spatial data. In order to execute such methods, the vague spatial object o can be transformed into a line with gradual transition using the reverse assignments, considering $l[i]$ is the i -th segment of a line l , such that $l[i]$ is described by $((x_1, y_1, z_1), (x_2, y_2, z_2))$, where x and y are coordinates and z are membership values. The following reverse assignments to transform a vague spatial object into a line with gradual transition apply:

$l[i] \leftarrow ((x_{c1}, y_{c1}, 1.0), (x_{c2}, y_{c2}, 1.0))$, where (x_{c1}, y_{c1}) and (x_{c2}, y_{c2}) are endpoints of the segment in $o.certitude[c].geo$; and

$l[i] \leftarrow ((x_{d1}, y_{d1}, z_1), (x_{d2}, y_{d2}, z_2))$, where (x_{d1}, y_{d1}) , (x_{d2}, y_{d2}) and (x_{d3}, y_{d3}) are endpoints of the segment in $o.dubiety[d].geo$ and z_1 and z_2 are their membership values, respectively.

4.9 The Vague Spatial MultiDim Conceptual Model

The VSCube conceptual model has enabled the conceptual modeling of vague SDWs and has provided flexible and multidimensional ways for representing, querying, and aggregating vague spatial data. However, the VSCube conceptual model does not offer graphic representations of the concepts, which could be utilized by database designers to elaborate diagrams. Therefore, the Vague Spatial MultiDim Conceptual Model, or simply VSMultiDim, is proposed to overcome the cited shortcoming.

Before introducing the VSMultiDim conceptual model, the requirements and goals of such a model are outlined and the adopted methods are briefly explained. The first requirement is to address multidimensional modeling. The goal is to reuse the legacy already developed for DW/OLAP, SDW/SOLAP, and vague SDW/vague SOLAP. The second requirement is to enable the elaboration of diagrams and symbols to represent concepts. The goal is to facilitate the communication between designers and users when they create conceptual schemata for vague SDWs. The third requirement is to support vague spatial data types defined by exact models and fuzzy spatial data types defined by fuzzy models. The goal is to enable modeling a wide variety of real-world phenomena.

In order to achieve these goals and fulfill these requirements, the MultiDim conceptual model outlined in Section 3.1.1 has been extended to reuse formal definitions of the VSCube conceptual model. As a result, the VSMultiDim model addresses multidimensional modeling, inherits and extends MultiDim's graphical notations, and applies VSCube's concepts. The *conceptual schema* of a vague SDW designed according to the VSMultiDim model is a diagram that encompasses a set of *facts* and a set of *levels* that represent dimensions. Then, a schema with a single fact in the VSMultiDim model is analogous to a cube modeled according to the VSCube model.

The following sections detail the VSMultiDim model. In order to illustrate the applicability of the VSMultiDim model, examples are provided to describe two different situations according to the case studies. The examples regarding the pest control case study assume that a data cube has already been proposed as conceptual schema using the VSCube conceptual model. Conversely, the examples regarding the HLB case study assume that the conceptual schema is modeled with the VSMultiDim conceptual model to elaborate a diagram that represents the data cube. Section 4.9.1 describes fundamentals of the VSMultiDim model. Section 4.9.2 addresses levels, attributes and members, Section 4.9.3 tackles dimensions and hierarchies, and Section 4.9.4 focuses on fact and measures.

4.9.1 Fundamentals

The VSMultiDim model introduces pictograms for the visual representation of vague spatial data types and fuzzy spatial data types, as described in Section 4.9.1.1. The VSMultiDim model reuses certitude and dubiety, as detailed in Section 4.9.1.2. The VSMultiDim model also introduces pictograms to denote vague topological constraints, as described in Section 4.9.1.3.

4.9.1.1 Data Types

The VSMultiDim model's pictograms that denote data types are listed in Figure 4.20. They represent vague or fuzzy simple types Point, Line, and Region, as well as vague or fuzzy collection types Point Set, Line Set, and Region Set. Pictograms for crisp spatial data types were inherited from the MultiDim model and are also listed, for the sake of comparison.

	Point	Line	Region	Point Set	Line Set	Region Set
Crisp	•	~	◼	••••	~	◼
Vague	◦	~	◻	◦◦◦◦	~	◻
Fuzzy	•	~	◼	••••	~	◼

Figure 4.20: Pictograms denoting data types supported by the VSMultiDim conceptual model.

The pictograms differentiate vague spatial data types from fuzzy spatial data types. Recall that the former are based in a 3-valued logic to express membership as being either *true*, *maybe*, or *false* (Section 2.3.1.1), while the latter are based in fuzzy logic to quantify the membership degree in $[0, 1]$ (Section 2.3.2.1). Pictograms for vague spatial data types have black dashed contour and white fill. Pictograms for fuzzy spatial data types have a gradient where the solid color black represents the membership degree 1, darker tones of gray denote higher membership values, and brighter tones of gray denote lower membership values.

The selection of a pictogram to represent either a vague or a fuzzy spatial data type is done by analogy with the vague spatial attribute of the VSCube model, as follows. Suppose that a vague spatial attribute is defined. If the vague spatial attribute's instances have dubiety elements whose membership values either are in $]0,1[$ or are determined by a function whose codomain is $]0,1[$, then a gradient pictogram for a fuzzy spatial data type must be selected. If the vague spatial attribute's instances have dubiety elements whose membership values are *null* and means *maybe*, then a dashed pictogram for a vague spatial data type must be selected.

The pictograms also unambiguously distinguish data types concerning the shape and the cardinality. On the one hand, vague and fuzzy spatial data types are complex and manipulate

sets (collections) of shapes to ensure the correctness of their operators. On the other hand, conceptual design of SDWs requires simple data types to be specified in a conceptual schema, to exempt the user from understanding complex data types. As a result, the pictograms that denote data types address both the simple types and the collection types.

The selection of a pictogram to represent either a simple or a collection type is also done by analogy with the vague spatial attribute of the VSCube model, as follows. Suppose that a vague spatial attribute is defined. If the vague spatial attribute's instances have at most one dubiety element, then a simple type must be selected. If the vague spatial attribute's instances have more than one dubiety element, then a collection type must be selected.

A dashed point depicts a simple vague point either with known location and undetermined membership degree or with an unknown location inside an area. A dashed line represents a simple vague line with broad boundaries and/or interior. A dashed polygon denotes a simple vague region composed of a pair of regions. Examples of simple vague points are shown in Figure 2.8a,b,e, examples of simple vague lines are shown in Figure 2.8f-o, and examples of simple vague regions are shown in Figure 2.8q-s,u. Multiple dashed points depict a vague point set as exemplified in Figure 2.8c-d. Multiple dashed lines represent a vague line set as exemplified in Figure 2.8p. Multiple dashed regions denote a vague region set as exemplified in Figure 2.8t,v.

Furthermore, a gradient point depicts a simple fuzzy point, which is either a pair of coordinates with membership value or a buffer whose points have membership values assigned. A gradient line represents a simple fuzzy line that is a linear object whose membership values vary gradually. A gradient region denotes a surface whose points have membership values assigned. Examples of simple fuzzy points are shown in Figure 2.10a,b,d, an example of simple fuzzy line is shown in Figure 2.10e, and examples of simple vague regions are shown in Figure 2.10h-i. Multiple gradient points depict a fuzzy point set as exemplified in Figure 2.10c. Multiple gradient lines represent a fuzzy line set as exemplified in Figure 2.10f-g. Multiple gradient regions denote a fuzzy region set as exemplified in Figure 2.10j-l.

It is noteworthy that both VSCube and VSMultiDim models support vague spatial data types defined by exact models. However, the VSMultiDim model supports fuzzy spatial data types defined by fuzzy models, while the VSCube model supports implementations for fuzzy spatial data types (Section 2.3.3). As a result, the VSMultiDim model delegates the implementation choice to be made in the logical design of the vague SDW.

4.9.1.2 Certitude and Dubiety

Let o be an object assuming a vague spatial data type (Section 2.3.1.1) and \tilde{o} be an object assuming a fuzzy spatial data type (Section 2.3.2.1). Then, $Certitude(o)$ and $Certitude(\tilde{o})$ are subsets whose memberships to o and \tilde{o} , respectively, are certain. These subsets are called certitude of o and certitude of \tilde{o} , respectively. The certitude of o consists of the geometric shapes whose membership to o is *true*. The certitude of \tilde{o} is the fuzzy set $\{c \in \mathbb{R}^n \mid \mu_{\tilde{o}}(c) = 1.0\}$. The certitude of a crisp spatial object is the object itself.

Furthermore, $Dubiety(o)$ and $Dubiety(\tilde{o})$ are subsets for which there is dubiety about their memberships to o and \tilde{o} , respectively. These subsets are called dubiety of o and dubiety of \tilde{o} , respectively. The dubiety of o consists of the geometric shapes whose membership to o is *maybe*. The dubiety of \tilde{o} is the fuzzy set $\{d \in \mathbb{R}^n \mid 0 < \mu_{\tilde{o}}(d) < 1.0\}$. The dubiety of a crisp spatial object is empty.

4.9.1.3 Vague Topological Constraints

Recall that a topological relationship indicates how the locations of two spatial objects are related. These relationships are typically used for expressing topological constraints. Reusing topological relationships from existing exact models (Section 2.3.1.3) or fuzzy models (Section 2.3.2.3) to express topological constraints in a vague SDW could substantially increase the complexity of conceptual modeling. It would require to consider relationships that are fully satisfied and yield *true* or possibility 1, as well as and relationships that are partially satisfied and yield *maybe* or possibility in $]0,1[$.

Therefore, only relationships that are fully satisfied are considered in the VSMultiDim model. In addition, the sets of topological relationships $R_{(c,c)}$, $R_{(c,d)}$, $R_{(d,c)}$, and $R_{(d,d)}$ are reused from the VSCube model, as described in Section 4.3.1. Then, a vague topological constraint $T_{X,Y}$ between the sets of objects X and Y lists all topological relationships allowed between elements of x_i and elements of y_j , where $x_i \in X$ and $y_j \in Y$:

$$T_{X,Y} = \{R_{(c,c)}, R_{(c,d)}, R_{(d,c)}, R_{(d,d)}\} \quad (4.1)$$

Figure 4.21 lists the pictograms that have been created to denote topological relationships. Note that contour and fill colors are standardized, such that black refers to certitude and gray refers to dubiety. In a diagram, it is infeasible to represent a topological constraint graphically with all the allowed relationships. Rather, one representative from each set $R_{(c,c)}$, $R_{(c,d)}$, $R_{(d,c)}$, and $R_{(d,d)}$ is chosen to be graphically represented by a pictogram. The representative is the

more usual topological relationship, for the sake of simplicity.

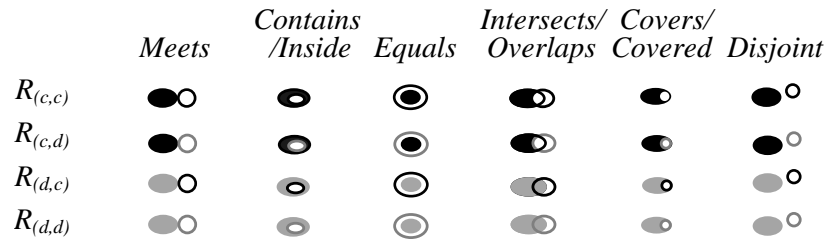


Figure 4.21: Pictograms denoting topological relationships of the VSMultiDim conceptual model.

4.9.2 Attributes, Levels and Members

In the VSCube model, a dimension comprises several attributes of distinct granularities, as described in Section 4.4.1. In the MultiDim model, a level is equivalent to an entity type of the ER model, as discussed in Section 3.1.1. As for the VSMultiDim model, a *level* represents a set of real-world concepts that have similar attributes. Additionally, a level is a subset of the attributes of a given dimension modeled according to the VSCube conceptual model, such that the subset belong to the same entity type. A level also has a type, which can be one of the types already defined for dimensions in Section 4.4.1:

- a *conventional level* has only conventional attributes and its members are called *conventional members*;
- a *non-geometric spatial level* has at least one non-geometric spatial attribute and optional conventional attributes, and its members are called *non-geometric spatial members*;
- a *crisp spatial level* has at least one crisp spatial attribute and no vague spatial attributes, and its members are called *crisp spatial members*; and
- a *vague spatial level* has at least one vague spatial attribute and its members are called *vague spatial members*.

Conventional levels and non-geometric spatial levels do not require a spatial representation and are treated as they already are in the MultiDim model. Also, crisp spatial levels are like spatial levels already defined by the MultiDim model. On the other hand, both the vague spatial

level and the vague spatial attribute require one pictogram to determine the data type, as shown in Figure 4.20. The pictogram is placed to the right of the name of a level or to the right of the name of an attribute.

Example 4.9.1. Regarding the pest control case study, Figure 4.22a illustrates the dimension $vague D_{Crop}$ modeled according to the VSCube in Example 4.4.1. Levels have been obtained by identifying subsets of attributes that belong to the same entity type. Crop is a vague spatial level that has a surrogate key on CropId, the conventional attribute PlantName, and an omitted vague spatial attribute that denotes a crop as a simple vague region. Agricultural Land is a vague spatial level that has a surrogate key on AgriLandId, the conventional attribute AgriLandOwner, and an omitted vague spatial attribute that denotes an agricultural lands as a simple vague region. Watershed is a crisp spatial level that has a surrogate key on WatershedId, the conventional attribute WatershedName, and an omitted crisp spatial attribute that denotes a watershed as a simple crisp region.

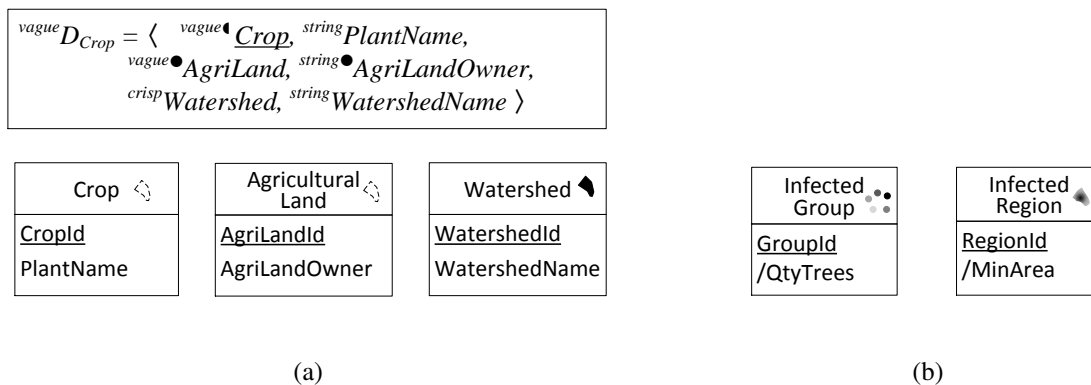


Figure 4.22: Vague spatial levels: (a) Obtained from a dimension of the pest control case study. (b) Modeled to comply with the HLB case study.

Example 4.9.2. Regarding the HLB case study, the vague spatial level Infected Group shown in Figure 4.22b has a surrogate key on GroupId, the derived attribute QtyTrees, and an omitted vague spatial attribute that denotes an infected group as a fuzzy point set. The quantity of trees in a group is supplied by QtyTrees. A member of this level describes a set of locations such that each location has an associated possibility of infection in]0,1]. Members of the level Infected Group are obtained using distance calculations, as discussed in Section 1.2. Therefore, a value for QtyTrees can only be calculated after the corresponding fuzzy point set is created. For instance, a vague spatial member of the level Infected Group has GroupId=11, the red point set shown in Figure 1.3c, and QtyTrees=7. Another member of Infected Group has GroupId=12, the orange point set shown in Figure 1.3c, and QtyTrees=7.

Example 4.9.3. Regarding the HLB case study, the vague spatial level `InfectedRegion` shown in Figure 4.22b has a surrogate key on `RegionId`, the derived attribute `MinArea`, and an omitted vague spatial attribute that denotes an infected region as a simple fuzzy region. A member of this level describes a continuous extent where the possibility of infection by HLB varies in $]0,1]$. The area in square meters where the possibility of infection by HLB is 1 is supplied by `MinArea`. Members of the level `InfectedRegion` are obtained from clustering, as discussed in Section 1.2. Therefore, values for `MinArea` can only be calculated after the corresponding fuzzy region is created. For instance, a vague spatial member of the level `InfectedRegion` has `RegionId=1`, the fuzzy region shown in Figure 1.3d, and `MinArea=628`.

In the following, Section 4.9.3 addresses hierarchies, which relate levels of distinct granularities. The logical design of the vague spatial attribute is tackled in Sections 5.2 and 5.3. The logical design of the vague spatial level is addressed in Section 5.4.

4.9.3 Dimensions and Hierarchies

Initially, consider both the VSCube and the MultiDim models for the sake of comparison. In the VSCube model, a hierarchy is a total order on a set of attributes as described in Section 4.3. Each pair of attributes with distinct granularities is associated through a cardinality and holds a vague topological constraint involving their certitude and dubiety elements. In the MultiDim model, a hierarchy relates several levels as discussed in Section 3.1.1. Each pair of crisp spatial levels with distinct granularity is associated through a cardinality and holds a topological constraint among their members.

As for the VSMultiDim model, a *dimension* encompasses at least one level and often has hierarchies. A *hierarchy* is defined by a set of associations between two levels, where one is the parent level that has a coarser granularity, and the other is the child level and has a finer granularity. The association has a cardinality and determines a topological constraint.

The cardinality of the parent-child association is either one-to-one, one-to-many, or many to many. A topological constraint relates members from the associated levels. In the diagram, four pictograms are placed between two vague spatial levels L_{parent} and L_{child} parallel to the connection used for the association. Figure 4.21 lists the pictograms that can be used. The pictogram for $R_{(c,c)}$ is the one closest to the top-left level, while $R_{(d,d)}$ is the one closest to the right-bottom level. When a given set is empty, e.g. due to an empty dubiety in a crisp spatial level, the symbol \emptyset replaces the pictogram. Also, if one of the levels is conventional, neither pictograms nor the symbol \emptyset are drawn.

Example 4.9.4. Figure 4.23 illustrates the hierarchy $H_{WaterSupply}$ from Example 4.3.4 as defined using the VSCube conceptual model. The corresponding graphical representation of such hierarchy has been created with the VSMultiDim model and shown in Figure 4.23. Watershed is a crisp spatial level while Agricultural Land is a vague spatial level. Although $R_{(c,c)} = \{contains, covers\}$, a single pictogram for *covers* has been used in the graphical representation because the latter is more frequent. The pictogram for *contains* was analogously chosen for $R_{(c,d)}$. On the other hand, the sets $R_{(d,c)}$ and $R_{(d,d)}$ are empty because watersheds are crisp and do not have dubiety. Thus, the symbol \emptyset was employed instead of a pictogram. The relationships among watersheds and agricultural lands have already been illustrated in Figure 4.6. Pictograms for the relationships among agricultural lands and crops were also analogously selected. Such relationships have already been illustrated in Figure 4.5.

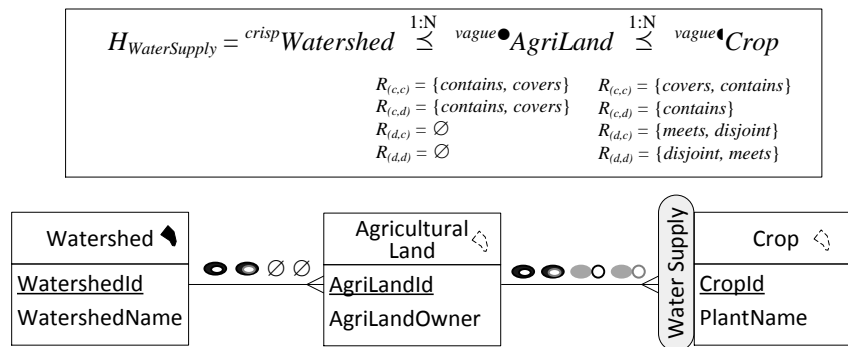


Figure 4.23: A hierarchy of the pest control case study represented according to both the VSCube model and the VSMultiDim model.

Concerning the HLB case study, note that queries HLB5 and HLB7 listed in Table 1.2 require infected regions, but clustering such regions on the fly is infeasible due to the high computation cost. In addition, those queries can alternatively refer to infected groups. Clearly, the spatial vagueness that is intrinsic of groups and regions must be represented in the conceptual schema of the SDW.

Example 4.9.5. Consider the HLB case study. Figure 4.24 illustrates the hierarchy Spread composed of the vague spatial levels Infected Group and Infected Region. The hierarchy Spread is encompassed by the dimension Infection, which provides a perspective of analysis that considers the intrinsic spatial vagueness of both infected groups and infected regions. Then, the spread of the disease can be analyzed by group containing a set of locations with associated possibilities of infection, or by region affecting a wider extent and encompassing several groups. An infected group is associated to at most one infected region, while an infected region comprises one or

more infected groups. According to the pictograms shown, a certitude element of a group is usually inside of a certitude element of a region and disjoint from a dubiety element of a region. Also, a dubiety element of a group is usually disjoint from a certitude element of a region and inside of a dubiety element of a region. These topological relationships hold for the two groups, red and orange, associated to a region in Figure 1.3d.

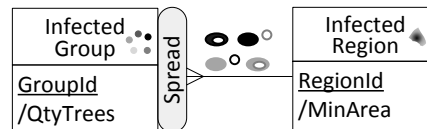


Figure 4.24: A hierarchy of the HLB case study represented according to the VSMultiDim model.

Example 4.9.6. Consider the HLB case study. Figure 4.25 shows the hierarchies Citriculture, Personnel, and Calendar. They belong to respective homonym dimensions. The hierarchy Citriculture is described by the crisp spatial levels Tree, Plot, Farm, and City. A member of Tree denotes a tree, is spatially represented as a simple crisp point, and has an identifier in *Treeld*, a planting date in *PlantingDate*, and an initially unknown eradication date in *EradicationDate*. A member of Plot denotes a plot, is spatially represented as a simple crisp region that contains one or more trees, has an identifier in *PlotId* and the name of the planted trees in *PlantName*. A member of Farm denotes a farm, is spatially represented as a crisp region that covers one or more plots, has an identifier in *FarmId* and its owner's name in *Owner*. A member of City denotes a city, is spatially represented by a crisp region set that covers one or more farms, is identified by its name *CityName* and has its elevation represented by a continuous field in *Elevation*. The hierarchy Personnel is described by the conventional levels Inspector and Team. A member of Inspector denotes an inspector whose identification and name are in *InspectorId* and *InspectorName*, respectively. A member of Team denotes a team composed of one or more inspectors and its name is given by *TeamName*. The hierarchy Calendar is described by the conventional levels Month, Quarter, and Year. A member of Month denotes a month identified by its number *MonthNo* and whose name is given by *MonthName*. A member of Quarter denotes a quarter comprising three months that is identified by its number *QuarterNo*. A member of Year denotes an year composed of four quarters that is identified by its number *YearNo*.

Consider the hierarchies described in Examples 4.9.5 and 4.9.6 regarding the HLB case study. Note that an infected group has a set of trees' locations plus the possibility of infection, as shown in Figure 1.3c. Also, an infected group's extent overlaps one or more plots, as shown in

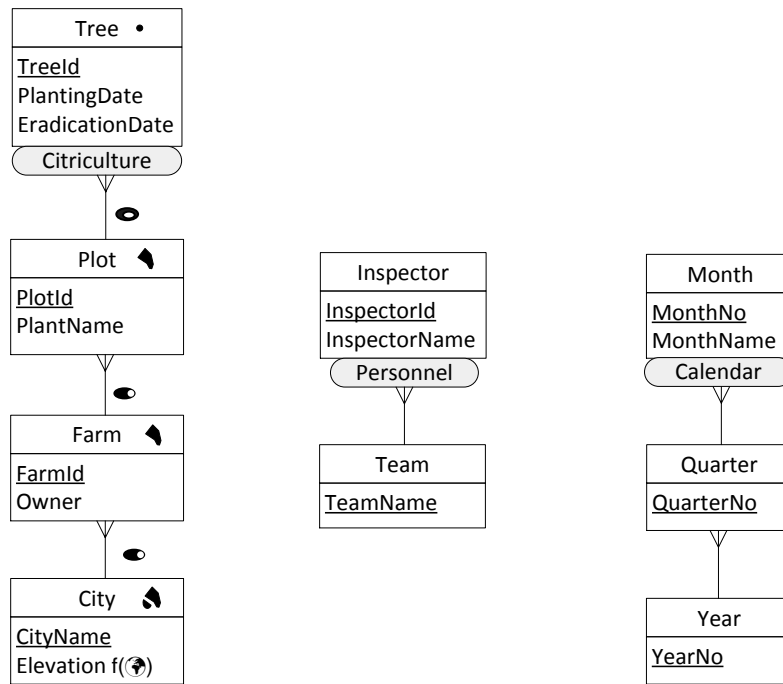


Figure 4.25: Hierarchies of the HLB case study represented according to the VSMultiDim model.

Figure 1.3d. Indeed, an infection affects trees and spreads over plots. However, the topological relationships that relate HLB infections to citricultural trees and plots are not sufficient to justify a single hierarchy, because the concepts are distinct. Therefore, the dimensions Infection and Citriculture are separated, as detailed in those examples. Analogously, the hierarchies Spread and Citriculture are also distinct, as shown in Figures 4.24 and 4.25. The relationship between an infection and a tree is provided by a fact.

The logical design of hierarchies in vague SDWs is addressed in Section 5.5. In the following, Section 4.9.4 tackles the fact and measures. The latter are aggregated along different levels of a hierarchy.

4.9.4 Fact and Measures

A *fact* relates levels and often has measures. A *fact member* is an instance of a fact. A *spatial measure* is a spatial attribute in a fact whose values provide spatial representations for the relationship among members of the related levels. The value of a spatial measure is the result an observation, e.g. a topological relationship satisfied by the related members.

A *conventional measure* is usually numeric and can be aggregated using SUM, for instance. A *crisp spatial measure* is a crisp spatial attribute whose values can be aggregated using geo-

metric union, for instance. A *vague spatial measure* is a vague spatial attribute whose values can be aggregated using existing operations outlined in Sections 2.3.1.2 and Section 2.3.2.2 or the aggregation functions introduced by the VSCube conceptual model in Section 4.6.

To create a fact using a cube modeled according to the VSCube, each pair (*measure, function*) is extracted from the cube and the name of the cube is assigned to the name of the fact. The type of a spatial measure is specified by a pictogram placed on the right of the measure's name. The pictogram is one of those shown in Figure 4.20. The aggregation function used to summarize values of a measure can also be specified on the right of the pictogram's name.

Example 4.9.7. Figure 4.26 describes the cube $C_{PesticideApplication}$ from Example 4.4.3 as modeled according to the VSCube model. The corresponding fact Pesticide Application modeled according to the VSMultiDim model is obtained by extracting the pairs (*AppliedTons*, SUM) and (*AppliedArea*, VSUnion) from $C_{PesticideApplication}$. The fact Pesticide Application relates the levels Pesticide, Date, and Crop. The conventional measure AppliedTons is numeric and aggregated using SUM, while AppliedArea is a vague spatial measure denoting simple fuzzy regions that can be aggregated using VSUnion. Each fact member indicates the quantity in tons and the location where pesticide application occurred by pesticide by date by crop, as required by the query PC1 listed in Table 1.1.

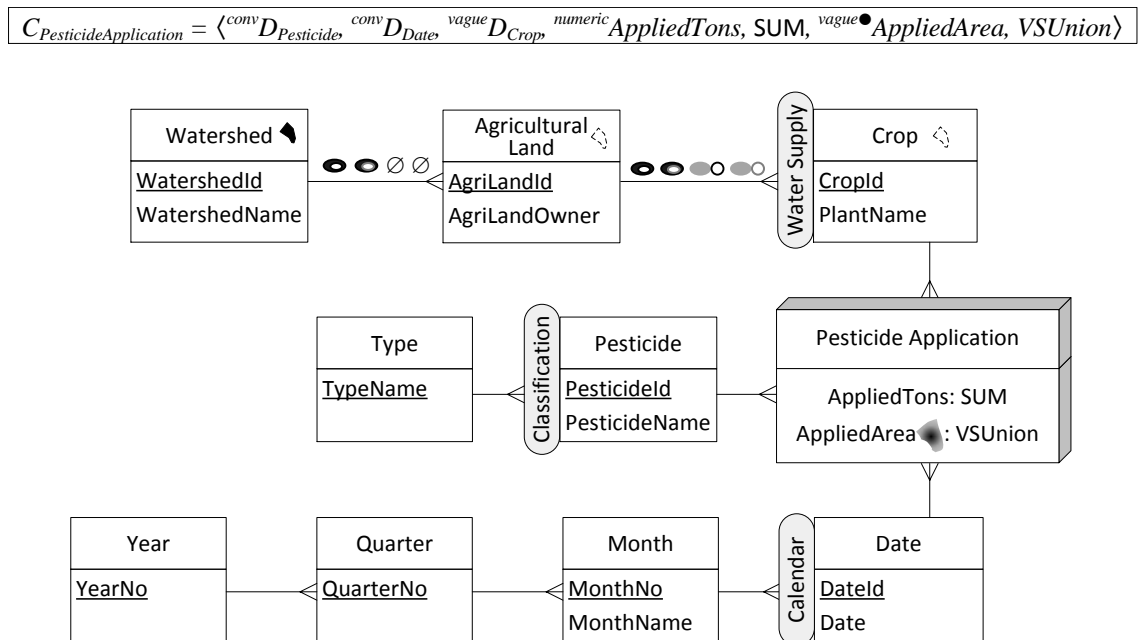


Figure 4.26: A conceptual schema of vague SDW for the pest control case study.

Example 4.9.8. Figure 4.27 illustrates the conceptual schema of vague SDW created using the VSMultiDim model for the HLB case study. The fact HLB Control relates the crisp spatial level

Tree, the vague spatial level Infected Group, and the conventional levels Month and Inspector. The numeric measure SymptomSeverity denotes a value in $[0,9]$ and is aggregated using AVG. The vague spatial measure InfectedTree represents the finest grain of an infection location, which coincides with a tree's location, and quantifies the possibility of infection in $[0,1]$. Therefore, its spatial representation is given by a simple fuzzy point. The vague spatial measure InfectedTree is aggregated using *VUnion*. The measure NeedsEradication? is not aggregated and allows an annotation whether eradication is needed. Each fact member indicates the symptom severity and location and possibility of infection by tree by infected group by month by inspector, as required by the query HLB1 listed in Table 1.2. An example of fact member is SymptomSeverity=8, the red point shown in Figure 1.3c representing InfectedTree with possibility 1, and NeedsEradication=True. Another example of fact member is SymptomSeverity=1, the topmost bright red point shown in Figure 1.3c representing InfectedTree with possibility 0.3, and NeedsEradication=False.

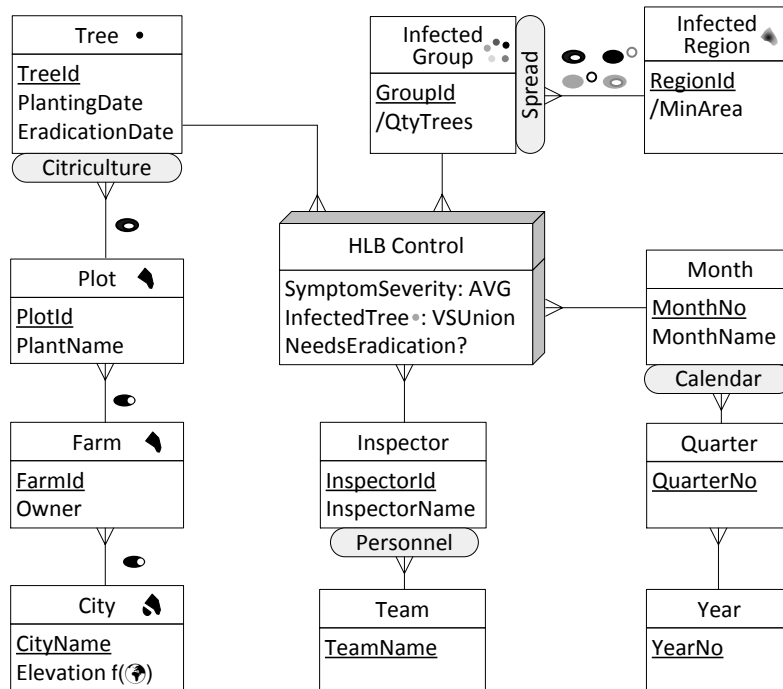


Figure 4.27: A conceptual schema of vague SDW for the HLB case study.

Before proceeding with the VSMultiDim model description, some aspects related to modeling are discussed. To illustrate the discussed topics, the vague SDW modeled for the HLB case study and shown in Figure 4.27 is used as example.

A member of Infected Group has a fuzzy point set composed of several simple fuzzy points

from all related infected trees described by fact members. Therefore, every member of Infected Group is related to one or several fact members. This relationship allows associating the HLB infection to citricultural trees and plots. Such relationship was not feasible considering exclusively the hierarchies, as discussed in Section 4.9.3.

Furthermore, the measure *InfectedTree* describes the possibility of infection at a given location. Thus, each fact member has a simple fuzzy point that is a fuzzy set whose membership function is given in Equation 2.3.2.1 (Section 2.3.2.1). The membership value quantifies the degree to which an element of the set belongs to the fuzzy set. Since the fuzzy set is a singleton, the sum of the membership values is 1 when the unique element has the membership value 1. Otherwise, the sum is greater than 0 and lesser than 1. A simple fuzzy point representing an infection is a poorly-defined object, as discussed in Section 2.3.

An alternative to represent the possibility of infection would be to add the measure *Infection Possibility* with numeric values in $]0,1]$ into the fact *HLB Control*. Then, each fact member would reference a member of the level *Tree* and inherit the tree's location. The query *HLB2* listed in Table 1.2 would require to apply *MAX* on *Infection Possibility* and geometric union on points representing trees. However, it is meaningless to obtain the numeric value of the maximum infection possibility by month by plot without a location. Unlike the numeric measure *Infection Possibility*, the proposed vague spatial measure *InfectedTree* represents the finest grain of an infection with both the numeric possibility and the location where the infection occurs.

Another alternative would be to assume randomness and stipulate chances of (non) occurrence of an infection by creating the uncertain measure *Infection Probability*, as described in Section 2.2.1.2. Using the uncertain domain $\{\text{Yes}, \text{No}\}$, a discrete probability distribution function could provide a pair of probabilities to associate each fact member with a chance of occurrence of infection. Considering Example 4.9.8, those fact members would have $\text{Infection Probability} = \{\text{Yes}/1.0, \text{No}/0.0\}$ and $\text{Infection Probability} = \{\text{Yes}/0.3, \text{No}/0.7\}$. The feasible scenarios for that pair of fact members would be (Yes, Yes), (Yes, No), (No, Yes), and (No, No). Also, each fact member would reference a member of the level *Tree* and inherit the tree's location.

Nevertheless, the interpretation of the uncertain measure *Infection Probability* differs from the interpretation of the vague spatial measure *InfectedTree*, as follows. The vague spatial measure indicates that the point (represented by a pair of coordinates) is an infected tree with a given possibility. Conversely, the uncertain measure indicates that, for a given fact member, the probability that it represents an infection is x while the probability that it does not represent an infection is $1 - x$. With the uncertain measure each fact member holds a pair of probabilities whose sum is 1, in contrast with a single membership value in $]0,1]$ required by the proposed

vague spatial measure. Furthermore, the uncertain measure neglects the location of the infection, which is intrinsic of a vague spatial measure. The query HLB2 listed in Table 1.2 would require to apply MAX on the probability associated with the element ‘Yes’ of the uncertain measure, in addition to a spatial union of trees’ locations.

After discussing some modeling aspects, the description of the VSMultiDim conceptual model is resumed. The VSMultiDim conceptual model does not provide it any graphical representation for the vague spatial fact, which has been introduced by the VSCube model in Section 4.4.4. Nevertheless, it is implicit that partial values of measures in a fact can be assigned to elements of vague spatial members from vague spatial levels related by the fact.

A *spatial fact* is a fact relating two or more spatial levels. For each pair of related spatial levels, one of the following situations occur:

- both are crisp spatial levels and topological constraints apply;
- there is one crisp spatial level and one vague spatial level and vague topological constraints apply; or
- both are vague spatial levels and vague topological constraints apply.

The constraints of a spatial fact are not depicted as pictograms in the fact in order to avoid excessive pictograms for visualization. For example, a pair of crisp spatial levels requires one pictogram, while a pair of vague spatial levels require four pictograms. Then, the number of required pictograms can increase significantly if there are several crisp or vague spatial levels related to the fact and impair readability and comprehension of the schema.

Example 4.9.9. In the schema shown in Figure 4.27, the spatial fact in HLB Control associates the crisp spatial level Tree and the vague spatial level Infected Group. The vague topological constraint between these levels is $\{R_{(c,c)}, R_{(c,d)}, \emptyset, \emptyset\}$, since members of Tree are crisp and have an empty dubiety. Also $R_{(c,c)} = \{intersects\}$ and $R_{(c,d)} = \{intersects\}$, as a tree location intersects an infected group’s set of locations.

The logical design for facts and vague spatial measures in vague SDWs is addressed in Section 5.6, while the logical design for the vague spatial fact is addressed in Section 5.7.

4.10 Summary

In this chapter, the VSCube conceptual model has been introduced to enable representing and querying vague spatial data in multidimensional databases. It produces a data cube as the

conceptual schema to enable flexible and multidimensional ways for representing, querying and aggregating vague spatial data. It also comprises attribute types, hierarchies and their categories, dimensions, measures, fact, vague spatial fact, cube and lattice of cuboids, and supports conventional data, non-geometric spatial data, crisp spatial data and, mainly, vague spatial data. Vague spatial data types defined by exact models and based on geometric features with an associated membership in $\{true, maybe, false\}$ are supported. Implementations of fuzzy spatial data types and based on geometric features with an associated (precomputed) membership in $]0,1[$ are also supported.

Furthermore, the VSCube conceptual model also defines vague spatial aggregation functions (vague spatial union, vague spatial intersection, and vague spatial difference) and vague spatial predicates (intersection range queries, containment range queries and vague spatial range queries). Moreover, the vague SOLAP and the multidimensional operations of slice-and-dice, drill-down, roll-up and pivot have been described. In order to corroborate the applicability of the VSCube conceptual model, examples have been elaborated based on the pest control case study. The VSCube conceptual model allows the analysis of business scores related to vague spatial data, and therefore improves the decision-making process. With the VSCube conceptual model, spatial vagueness has been introduced in SDWs and culminate in the vague SDW.

This chapter has also detailed the VSMultiDim conceptual model for vague SDWs that, differently from the VSCube conceptual model, offers graphic notations of the concepts and enables the elaboration of diagrams. The VSMultiDim mainly tackles the representation of spatial vagueness in attributes, levels, members, dimensions, hierarchies, fact, measures and schema of the vague SDW. It introduces pictograms to differentiate data types and to represent vague topological constraints. Vague spatial data types defined by exact models are supported. Fuzzy spatial data types defined by fuzzy models and based on fuzzy sets whose membership functions assign membership values in $[0,1]$ are also supported. All pictograms enable an easy hand-drawing and implementation in tools for computer-aided software engineering. Examples have been elaborated based on the pest control case study and on the HLB case study to corroborate the applicability of the VSMultiDim conceptual model. With the MultiDim conceptual model, the communication between users and designers are benefited by the enabled creation of diagrams for the vague SDW conceptual schema.

Conceptual modeling of vague SDWs, however, does not enable the elaboration of a vague SDW schema underlying a particular DBMS, which is intrinsic of the logical design of vague SDWs addressed in Chapter 5.

Chapter 5

LOGICAL DESIGN OF VAGUE SPATIAL DATA WAREHOUSES

The logical design of a database commonly maps the conceptual database schema into a logical database schema underlying a particular database management system (DBMS). The goals of the logical design are mainly to preserve the database integrity, to facilitate writing queries, to reduce the cost for querying the database, and to analyze the costs for storing the database and for maintaining database's constraints. The logical design of data warehouses and spatial data warehouses has been mainly tackled by storing data in relational databases, extending SQL and providing specific access methods to efficiently implement the multidimensional data model and the corresponding operations.

This chapter details mapping rules that transform a conceptual schema of vague spatial data warehouse, modeled according to the VSCube and the VSMultiDim models, into a relational logical schema. Furthermore, it explains the constraints of the vague spatial data warehouse and provides their implementations. Moreover, it enables querying the relational logical schema of the vague spatial data warehouse and performing vague spatial online analytical processing.

An overview of the relational representation of vague spatial data warehouses is outlined in Section 5.1. Section 5.2 details and discusses alternative implementations for the vague spatial attribute. Section 5.3 selects one implementation for the vague spatial attribute and describes its design. Section 5.4 addresses the vague spatial level and the vague spatial member. Section 5.5 focuses on hierarchies. Section 5.6 stands for the fact and vague spatial measures. Section 5.7 addresses the vague spatial fact, which allows the assignment of measure values to certitude elements and dubiety elements of vague spatial members. Section 5.8 tackles vague topological constraints. Section 5.9 focuses on query processing and vague spatial online analytical processing and describes how SQL has been extended to query vague spatial data warehouses.

Finally, Section 5.10 summarizes the chapter.

5.1 Relational Representation of Vague Spatial Data Warehouses

A vague spatial data warehouse (vague SDW) has at least one vague spatial attribute in a dimension or as a measure in a fact. A relational vague SDW assumes that there is at least one table holding a vague spatial attribute. The starting point of the logical design for the vague SDW as a relational database consists of defining how a vague spatial attribute is designed in a table. The VSCube conceptual model has defined a vague spatial attribute as a composite attribute whose data fields are certitude and dubiety. Certitude is a multivalued data field of geometries, while dubiety is composed by both a multivalued data field of geometries and a multivalued data field of membership values.

However, the original relational model has well-known shortcomings, e.g. supporting only atomic, monovalued and conventional attribute types. Therefore, current relational DBMSs preserves foundations of the relational model and additionally offer a variety of extensions that can be used to improve the logical design of databases demanded by more complex system requirements. Spatial extensions, object-relational features and support for multivalued attributes are seen as the most promising resources of current DBMSs that can be investigated for the provision of the logical design for vague SDWs based on the relational model.

Spatial extensions provides spatial data types, spatial operators, spatial functions and spatial indices. Spatial data types enable storing a collection of geometries in a single row, which can be particularly useful for storing certitude elements and dubiety elements of a vague spatial object. Spatial operators and spatial functions can be reused to develop aggregation functions for vague spatial data and vague spatial predicates as well as to check topological constraints. Spatial indices can be built to speed up the resolution of vague spatial predicates. Spatial data types, spatial operators and spatial functions comply with the OGC standard. This compliance should be maintained in the logical design of the vague SDW to ensure the validity of data and compatibility with higher-level application programming interfaces.

Furthermore, object-relational features allow preserving foundations of the relational model while data is organized using an object model. The extensibility feature can be explored for extending SQL and implementing user-defined functions (UDFs). Aggregation functions for vague spatial data, vague spatial predicates, and also constraints of the vague SDW can be implemented as UDFs. Moreover, the support for multivalued attributes is particularly useful

to tackle membership values of elements from a vague spatial object. Since a collection of geometries can be stored in a single row, multiple membership values should also be stored in a single row, for example. Note that a correspondence between a geometry and a membership value is mandatory.

Obviously, more than one implementation for the vague spatial attribute might be possible. However, only some of them may be feasible. Comparisons between different approaches should be done to identify advantages and shortcomings of each one. In addition, differently from conceptual modeling that is independent of implementation details, the logical design depends on the underlying DBMS. As a result, the flexibility of representing both exact models and fuzzy models may not be achieved in the logical design, differently from the VSCube and the VSMultiDim conceptual models. The limitations must be reported to prevent designers from creating logical schemata that do not comply with the requirements of the application.

Once vague spatial attributes can be defined in tables, the representation of dimensions, hierarchies and facts is essential for obtaining a logical schema for the vague SDW. Dimensions may have one or more hierarchies associating levels, thus requiring several tables to be represented. Facts may have vague spatial measures that comply with the implementation of vague spatial attributes. Furthermore, values of measures can be associated to elements of vague spatial members and then demand a particular design of the fact table. As a result, the *logical schema* of a vague SDW designed as a relational database consists of all the tables accordingly created.

The vague SDW requires the definition of constraints with different purposes. Some constraints are related to the implementation of the vague spatial attribute itself. Other constraints restrict the allowed topological relationships among vague spatial objects of different granularities, for example. Constraints must be specified and implemented in the logical design of the vague SDW.

Once a vague SDW has a logical schema implemented in a DBMS, it can be queried to benefit decision-making. The vague spatial online analytical processing (vague SOLAP) comprises queries that require extending the SQL with operations implemented as UDFs. These operations have a variety of purposes, such as accessing elements of vague spatial objects, resolving vague spatial predicates, and processing vague spatial aggregation functions. Moreover, slice-and-dice, roll-up and drill-down reuse these operations to carry on different analyses of the data cube implemented as a relational database.

This chapter addresses the relational representation of vague SDWs. On the one hand, the implementations have been specified using PostgreSQL and its spatial extension PostGIS.

On the other hand, efforts have been taken to follow SQL and OGC standards to allow the reproduction of the implementations in other DBMSs by performing minor adaptations.

5.2 Implementations for the Vague Spatial Attribute

The definition of a vague spatial attribute into an arbitrary table T may require one or more columns and even additional tables to enable the representation of each vague spatial object as one or more rows. In the context of a vague SDW, the table T is a fact table, a dimension table or a level table, as discussed in Section 2.1.2. Independently from the table where the vague spatial attribute is defined, its logical design should:

- comply with the OGC standard and allow the registration of geometry columns in OGC's metadata;
- fit multivalued certitude and dubiety as well as monovalued certitude and dubiety;
- minimize the number of joins between tables when processing queries;
- enable the use of existing indices implemented by the DBMS;
- minimize the number of calls to DBMS' internal functions when processing queries;
- sort elements of the vague spatial object in ascending or descending order of membership values; and
- store the geometric union of the elements of the vague spatial object.

The compliance with the OGC standard and the registration of geometry columns in OGC's metadata are essential to ensure the compatibility with end-user applications, as described in Section 2.1.2. The adequacy to multivalued certitude and dubiety as well as monovalued certitude and dubiety ensures the compatibility with the vague spatial data types. The other requirements aims at improving the performance to process queries. The benefits of using indices have been summarized in Section 2.1.3. The number of calls to internal functions of the DBMS should be minimum to avoid additional overheads. The retrieval of elements with a low or high membership value can be faster if these elements are by default at the beginning or at the end of a list, for example. The geometric union of the elements of a vague spatial object is a geometry whose number of vertices tends to be smaller than the total number of vertices required by the collection of elements. Since the number of vertices of a geometry can impair the performance to process queries, as indicated in Section 2.1.3.2, the precomputation and storage of

the geometric union aims at speeding up the resolution of spatial predicates, such as IRQ_{object} , CRQ_{object} , and $VSRQ_{object}$ that fetch objects instead of their elements.

In the following sections, different approaches for the logical design of the vague spatial attribute are described. Nevertheless, these approaches achieve the aforementioned requirements only partially. Thus, the main advantages and disadvantages of each approach are duly discussed. Section 5.2.1 describes the use of separate tables for certitude and dubiety, while the approach detailed in Section 5.2.2 employs a single table to store the certitude and the dubiety. Differently from Sections 5.2.1 and 5.2.2, which adopt monovalued and atomic attributes to comply with the relational model, Section 5.2.3 designs the vague spatial attribute according to an object-relational approach and the definition of an UDT. In Section 5.2.4, the vague spatial attribute is denoted as a pair of columns of type array. Section 5.2.5 describes the use of one column for geometries and one column for the array of membership values, i.e. it details the vague spatial attribute as described in Section 5.3. Differently from Sections 5.2.1 to 5.2.5, Section 5.2.6 specifically addresses the logical design of vague spatial attributes whose both the certitude and the dubiety are monovalued. Finally, in contrast with Sections 5.2.1 to 5.2.6, Section 5.2.7 tackles the use of geometries with 2D plus measure and geometries with 3D.

5.2.1 Separate Tables for Certitude and Dubiety

The logical design of the vague spatial attribute described in this section corresponds to a logical design that has been published (SIQUEIRA et al., 2012a). An E-R representation of the vague spatial attribute has been illustrated in Figure 4.3. As for the logical design, the E-R to relational mapping performed with adaptations on the vague spatial attribute is shown in Figure 5.1. In addition to the columns produced by the E-R to relational mapping, the table T also holds the column `MergedElementsGeo` that stores the geometric union of the certitude and the dubiety of each object. The column `MergedElementsGeo` can be indexed by an spatial index and also registered in the OGC's metadata. There is a physical separation of the certitude and the dubiety of the objects, since the certitude is stored in the table `Certitude` while the dubiety is stored in the table `Dubiety`.

The table `Certitude` stores the elements that belong to the certitude of the objects. As a result, each object has at most one row in the table T and possibly several rows in the table `Certitude`. The column `TFK` specifies the object that owns an element of the certitude, since a foreign key defined on `TFK` references the table T. In addition, each element of the certitude has an identifier to distinguish it from other elements of the certitude provided by the column `CertitudeElementId`. The composite primary key of the table `Certitude` comprises the columns

TFK and CertitudeElementId. The geometric feature associated to an element of the certitude is denoted by the column CertitudeElementGeo, which can be indexed by a spatial index and also registered in the OGC's metadata. There is not a column to specify the membership values associated to elements of the certitude, since these are assumed to have membership value equal to 1.0.

Furthermore, the table Dubiety stores the elements that belong to the dubiety of the objects. The table Dubiety is designed analogously to the table Certitude. The identifier for elements of the dubiety is provided by the column DubietyElementId, a foreign key on the column TFK references the corresponding object defined in the table T, and the primary key encompasses the columns TFK and DubietyElementId. Also, the column DubietyElementGeo holds the geometric feature associated to the element of the dubiety, can be indexed by a spatial index, and can be registered in the OGC's metadata. Additionally, the column DubietyElementMval denotes the membership value associated to the element of the dubiety. If the membership value is a degree of membership, then its value is in $]0,1[$. Otherwise, if it means *maybe*, a default value is assigned, e.g. -1 . For example, Figure 5.2 highlights a vague region and its storage in separate tables created for certitude and dubiety.

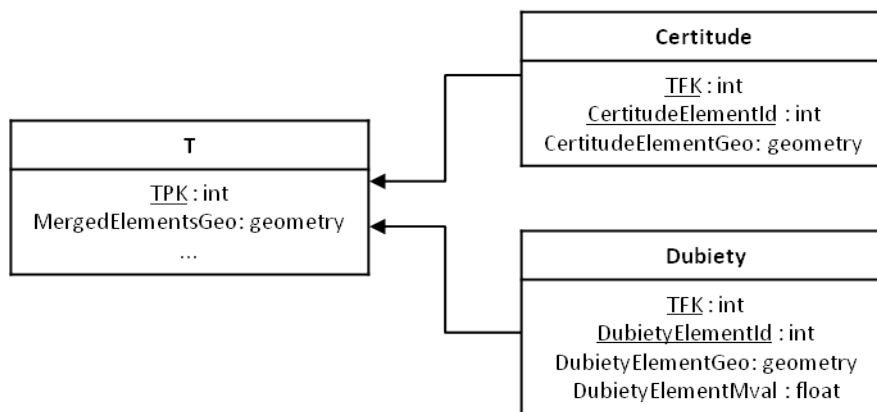


Figure 5.1: Separate Tables for Certitude and Dubiety.

The logical design for the vague spatial attribute detailed in Figure 5.2.1 and exemplified in Figure 5.2 has the following positive aspects. The column MergedElementsGeo is particularly beneficial for the computation of vague spatial predicates such as IRQ_{object} and CRQ_{object} , since it prevents joining the tables Certitude or Dubiety to the table T. In addition, note that the elements of the certitude and the dubiety are stored individually in the tables Certitude and Dubiety, respectively. Therefore, it is not necessary to use internal functions of the DBMS to differentiate certitude elements from dubiety elements when retrieving individual elements.

On the other hand, the shortcomings are the following. The existence of three tables im-

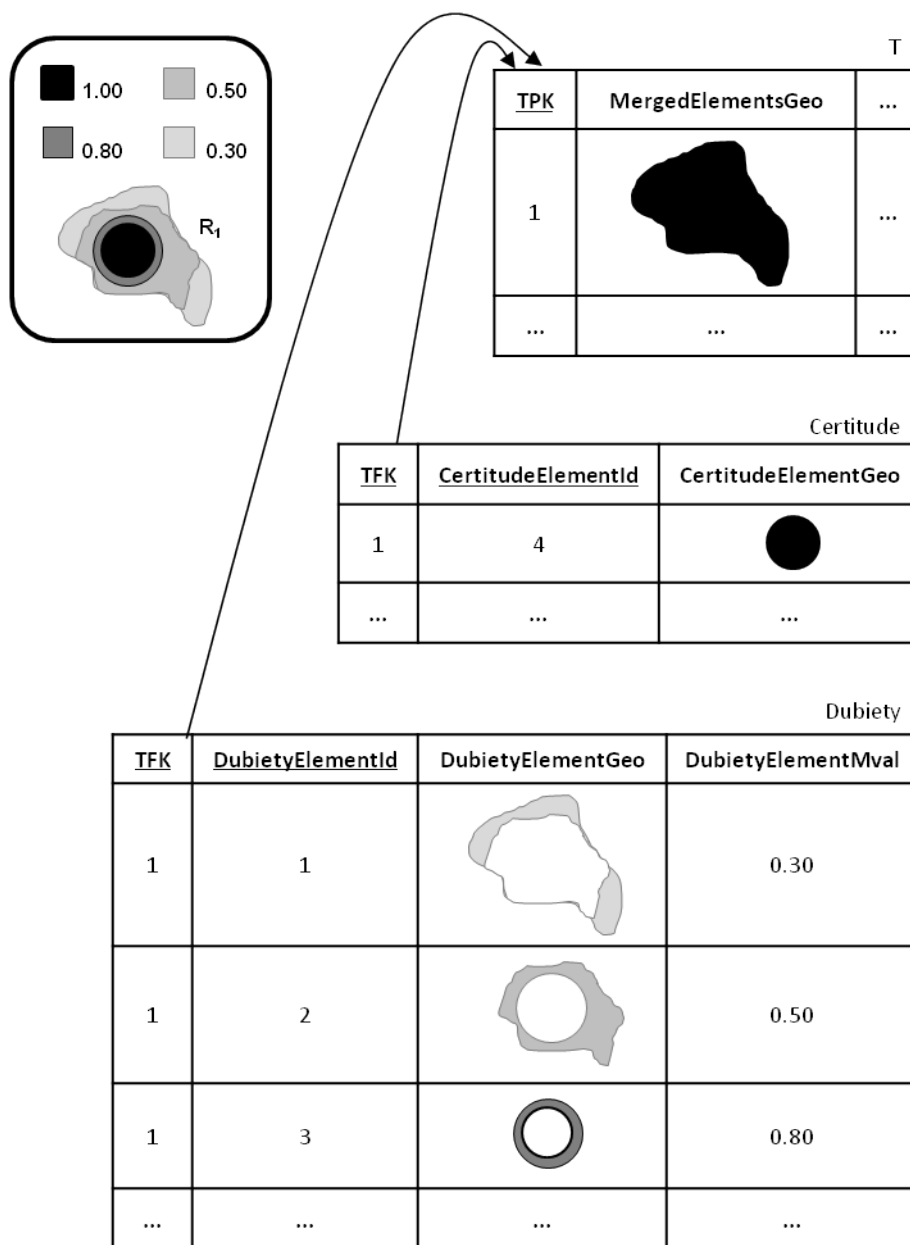


Figure 5.2: A vague region and its storage in separate Tables for Certitude and Dubiety.

poses the computation of joins to process vague spatial predicates that fetch the certitude or the dubiety, as well as to process vague spatial aggregation functions. Besides, the separation of the elements in distinct tables is unnecessary for a vague spatial attribute whose both the certitude and the dubiety are monovalued, as each object maintains one row in the table T, one row in the table Certitude and one row in the table Dubiety. Figure 5.3 not only illustrates the described situation but also indicates how the column DubietyElementMval becomes useless when the default value -1 is used to denote *maybe* and therefore every row assumes DubietyElementMval=-1.

Clearly, the separation of certitude and dubiety into two different tables can deteriorate the query processing performance due to the computation of joins. This shortcoming motivates an

alternative logical design for the vague spatial attribute, which is explained in Section 5.2.2.

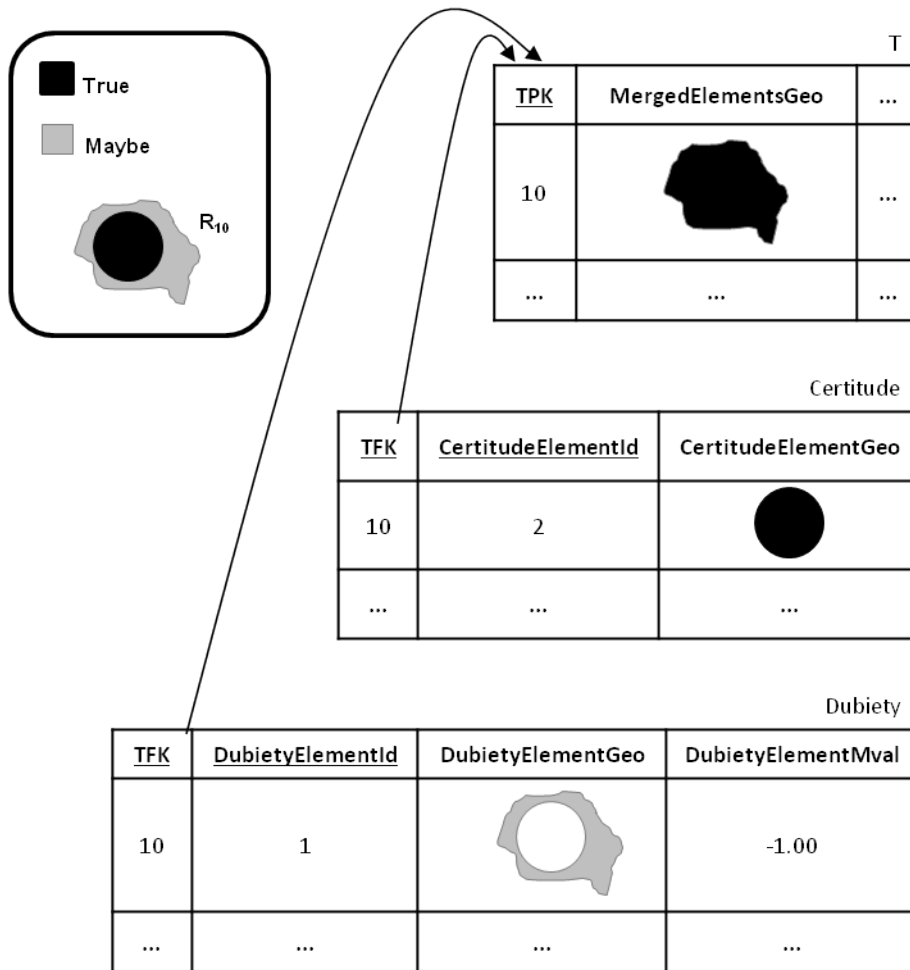


Figure 5.3: A vague region and its storage in separate Tables for Certitude and Dubiety considering a vague spatial attribute whose both the certitude and the dubiety are monovalued.

5.2.2 A Single Table for Certitude and Dubiety

In order to avoid processing joins between tables as discussed in Section 5.2.1, an alternative logical design for the vague spatial attribute considers a single table to store the certitude and the dubiety, as shown in Figure 5.4. The table T holds the column MergedElementsGeo to store the geometric union of the certitude and the dubiety. The table Element maintains elements of both the certitude and the dubiety of the objects. The column ElementGeo addresses the geometries of the elements, while the column ElementMval denotes the membership value associated to an element. If the element belongs to the certitude, then ElementMval=1.0. Otherwise, the element belongs to the dubiety and ElementMval is either in $]0, 1[$ to denote a degree of membership, or is a default value that means *maybe*, e.g. -1. Each one of the columns MergedElementsGeo and ElementGeo can be registered in OGC's metadata and have a spatial

index built.

Figure 5.5 exemplifies the storage of one vague region in these tables. The vague spatial attribute exemplified has both the certitude and the dubiety multivalued. Conversely, it is straightforward to note that at most one row in the table T and two rows in the table Element are required if the vague spatial attribute has both the certitude and the dubiety monovalued.

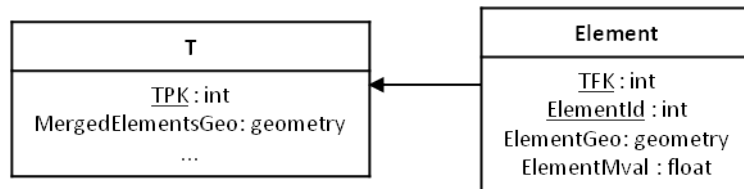


Figure 5.4: The vague spatial attribute represented by a single table.

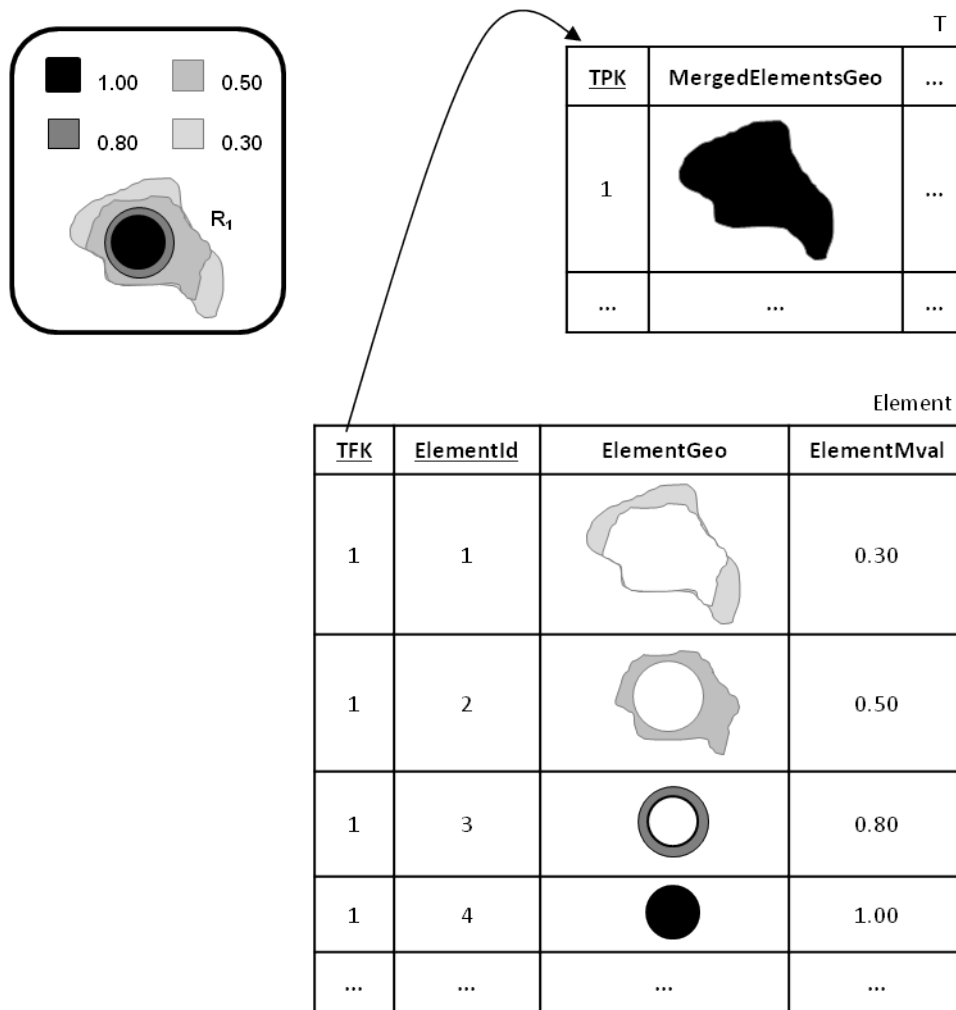


Figure 5.5: An example of vague region according to the vague spatial attribute represented by a single table.

The logical design that considers one single table to store the elements of the certitude

and of the dubiety has the following advantages over the use of separate tables described in Section 5.2.1. First, it demands the creation of one additional table instead of two, i.e. Element rather than Certitude and Dubiety. Second, it requires one column for the geometries of elements instead of two, i.e. ElementGeo. Third, to process vague spatial predicates, it imposes at most one additional join between tables instead of two, i.e. between the tables T and Element. Fourth, to process vague spatial aggregation functions, it does not require any additional join between tables rather than two, since only the table Element can be accessed. Like separate tables, the approach of the single table prevents the use of internal DBMS functions to extract the elements of the object, as they are individually stored in the table Element.

In the table T, a spatial index created on the column MergedElementsGeo indexes the geometric union of the certitude and the dubiety of an object. Conversely, in the table Element, a spatial index created on the column ElementGeo indexes all the geometries of elements belonging to the objects. For example, considering the dataset shown in Figure 5.5, the spatial index created on the column ElementsGeo has one entry for the geometry where $TPK=1$, while the spatial index created on the column ElementGeo has one entry for each geometry where $TFK=1$, i.e. a total of four entries.

There is one issue concerning the spatial index created on the column ElementGeo, as follows. If the elements of an object are very close one to each other, the MBRs of the elements might overlap one each other, as shown in Figure 5.6. Recall that MBRs that overlap can impair the performance of the spatial index to process queries, as discussed in Section 2.1.3.2. In addition, the spatial index is potentially voluminous as it has one entry for each element.

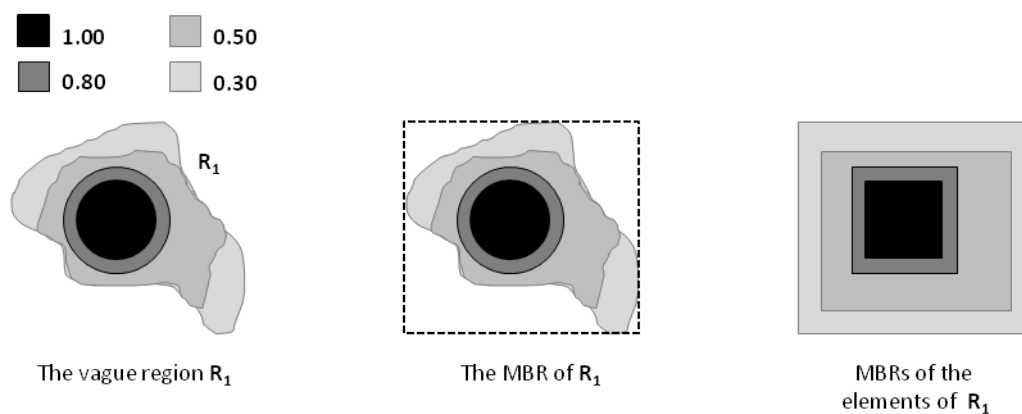


Figure 5.6: A vague region, its MBR and the MBRs of its elements.

One join between the tables T and Element may be required to process some queries against the logical design illustrated in Figure 5.4. This fact motivates the design of a unique table T as shown in Figure 5.7, where a composite primary key encompasses the columns TPK and ElementID. However, the following issues raise. First, the use of a composite primary key induce

other tables that reference T to have a composite foreign key. As a result, joins between T and any table that reference it might have the performance impaired. Second, if the table T denotes a level and is referenced by a fact table, the grain of the fact that should be *by vague spatial level* becomes *by element of the vague spatial level*. The last grain is expected for the vague spatial fact, whose concept was previously detailed in Section 4.4.4 and whose logical design is discussed in Section 5.7. Therefore, the minimization of the joins imposed by the logical design depicted in Figure 5.4 should not be achieved by using the logical design shown in Figure 5.7.

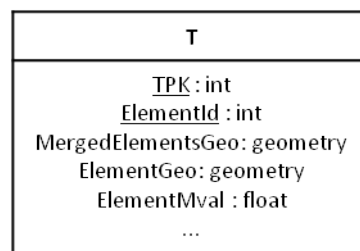


Figure 5.7: A single table with a composite primary key.

Finally, although the logical design shown in Figure 5.4 has advantages over the separate tables described in Section 5.2.1, the following shortcomings still remain. The column ElementID is completely artificial and has no meaning for users, as these will not query the vague SDW to retrieve the identifiers of the objects. Besides, one additional join between the tables T and Element may be required to process vague spatial predicates. The additional join should be eliminated to benefit the performance to process queries. As a result, a solution that completely embeds the vague spatial attribute in the table T is required. Thus, it becomes infeasible to adopt strictly monovalued and atomic columns to comply with a purely relational approach.

5.2.3 User Defined Types

Sections 5.2.1 to 5.2.2 have focused the logical design according to the relational model. In this section, the logical design of the vague spatial attribute is discussed according to an object-relational approach.

In this sense, the classes `VSElementType` and `VSAtributeType` play key roles. The class `VSElementType` represents elements of the certitude and of the dubiety, whose characteristics are a geometry denoted by the monovalued attribute `ElementGeo` and an associated membership value indicated by the monovalued attribute `ElementMval`. The domain of `ElementMval` is $]0, 1]$ if it denotes a membership degree, otherwise it is $\{-1, 1\}$ where -1 and 1 mean *maybe* and *true*, respectively. The class `VSAtributeType` defines the vague spatial attribute and holds a multivalued attribute of the class `VSElementType`. Figure 5.8 shows the classes `VSElement` and `VSAtribute`

and its attributes. They are implemented as UDTs in the DBMS. The multivalued attribute in `VSAtributeType` is implemented as an array in the DBMS.

Figure 5.8 also illustrates the table `T` holding the columns `TPK`, `MergedElementsGeo` and `VSAtribute`. The primary key is defined on `TPK`. Certitude elements and dubiety elements are addressed by `VSAtribute` whose type is the UDT `VSAtributeType`, while `MergedElementsGeo` contains the geometric union of all elements of the object. For example, Figure 5.9 highlights a vague region and illustrates it as an array of pairs in the column `VSAtribute` of the table `T`. The array is of the UDT `VSAtributeType` and each entry of the array is a pair of the UDT `VSElementType`.

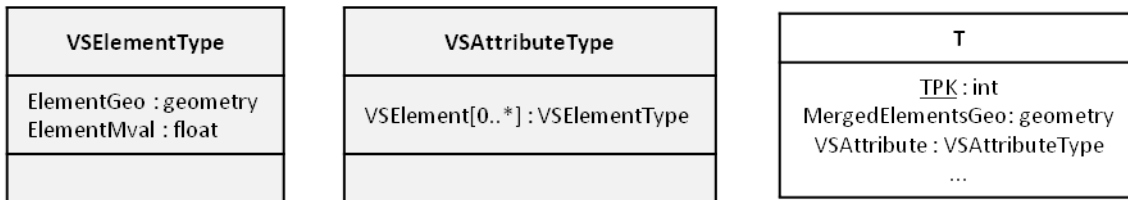
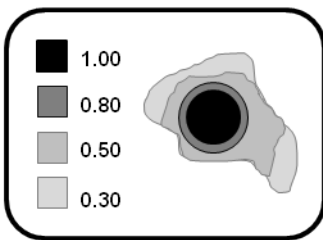


Figure 5.8: The classes `VSElementType` and `VSAtributeType` and the table `T`.



T

<u>TPK</u>	MergedElementsGeo	VSAtribute	...
1		{ 0.30, 0.50, 0.80, 1.00 }	...
...

Figure 5.9: A vague region and its representation as an array.

Embedding the vague spatial attribute in one table avoids joins between tables to process vague spatial aggregation functions and vague spatial predicates, since they can access exclusively the column `VSAtribute` in the table `T` to manipulate objects. Besides, identifiers are not necessary to distinguish elements belonging to the same object. A vague spatial attribute whose both certitude and dubiety are monovalued also complies with this logical design, such that the array in the column `VSAtribute` has at most two entries. These are the main advantages of

the approach described in this section over the approaches previously tackled in Sections 5.2.1 and 5.2.2.

On the other hand, the following drawbacks make the object-relational approach described in this section infeasible. First, columns of a non-geometry type cannot be registered in OGC's metadata and there are shortcomings associated to lack of registration (Section 2.1.2.2) As a result, it is impossible to register the column `VSAtribute` of the UDT `VSAtributeType` because numeric membership values are addressed by the attribute `ElementMval` of the UDT `VSElementType`. Second, it is not possible to create an index on a column whose type is an UDT. Therefore, both the geometries and membership values of the elements cannot be indexed, since the column `VSAtribute` is of the UDT `VSAtributeType`. The impossibility of indexing may impair the performance to process queries. Although PostgreSQL/PostGIS offers the alternative of creating a new operator class for the index (e.g. GiST) to store and sort instances of an UDT, such as those of the composite type `VSAtributeType`, such implementation is platform-dependent. Third, consecutive calls to constructors `ROW` and `ARRAY` are essential to extract both the geometries and the membership values of elements. Although these constructors are intrinsically a SQL standard, they add an overhead to query processing as they are like function calls. The aforementioned drawbacks are tackled in Sections 5.2.4 and 5.2.5.

5.2.4 A Pair of Arrays

In PostgreSQL/PostGIS, the vague spatial attribute can also be designed as a pair of columns of type array as shown in Figure 5.10: one array of geometries called `ElementsGeo` and one array of membership values called `ElementsMval`. A constraint must ensure that these arrays have essentially the same length for each row of the table `T`. The names of the columns `ElementsGeo` and `ElementsMval` have the prefix `Elements` in the plural to emphasize that they are multivalued. The table `T` also holds the column `MergedElementsGeo` to store the geometric union of the certitude and the dubiety of each object. The value of each entry of an array in `ElementsMval` is either in $]0, 1]$ if it denotes a membership degree, or in $\{-1, 1\}$ where -1 and 1 mean *maybe* and *true*, respectively.

Regarding indexing, a GiST can be built on the column `MergedElementsGeo` while a GIN can be built on the column `ElementsMval`. Nevertheless, a GiST cannot be built on columns of type geometry array and, therefore, the column `ElementsGeo` cannot be indexed. An alternative solution is to build one GiST to index all i -th entries of the arrays provided by the column `ElementsGeo`. As a result, the elements of an object are not indexed by the same GiST and each GiST indexes at most one element of each object. For example, Figure 5.11 highlights two

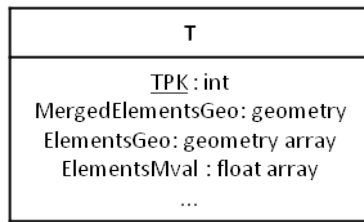
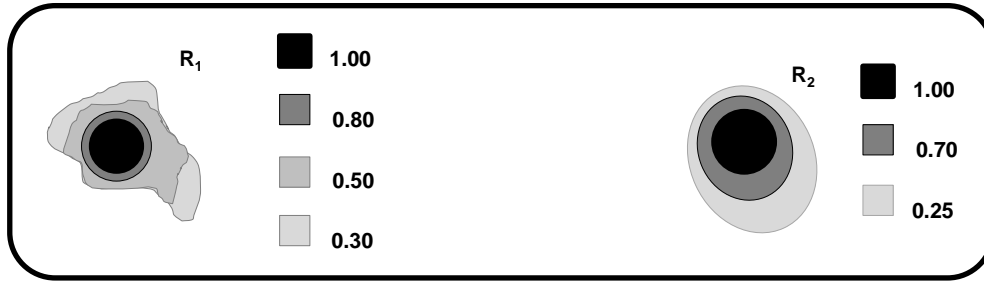


Figure 5.10: The vague spatial attribute represented by a pair of arrays.



T					
<u>TPK</u>	MergedElementsGeo	ElementsGeo		ElementsMval	...
1		{ }		{ 0.30, 0.50, 0.80, 1.00 }	...
2		{ }		{ 0.25, 0.70, 1.00 }	...
...

↓
 $GiST_1$

↓
 $GiST_2$

↓
 $GiST_3$

↓
 $GiST_4$

Figure 5.11: A vague region and its representation as a pair of arrays.

disjoint vague regions and illustrates their storage in the table T as a pair of arrays. In addition, it indicates with dotted arrows which elements of the objects are indexed by each GiST, such that for each ElementsGeo[i] there is an associated $GiST_i$. Conversely, the column ElementsMval can have a single index built.

A GiST whose MBRs refer to different objects might be less impaired by overlapping than a GiST whose MBRs belong to the same object. For example, the elements of the disjoint regions indexed as shown in Figure 5.11 have MBRs that do not overlap each other. Therefore, overlapping does not hamper the indices $GiST_1$ to $GiST_4$. On the other hand, indices that refer to elements of the same object are intrinsically impaired by MBRs that overlap, as discussed in

Section 5.2.2.

The implementation of vague spatial predicates and vague spatial aggregation functions should avoid querying all GiST individually and consecutively, otherwise the performance might be impaired. Besides, scanning simultaneously the array of geometries and the array of membership values to resolve queries might cause an overhead and affect the performance. Regarding insertion in the table T, a tuple whose object has n elements demands one insertion in each one of the n GiSTs and one insertion in the GIN. Therefore, the insertion in the table T may have the performance hampered.

In order to assure that all geometries are indexed, the number of GiSTs that must be built is equal to the length of the longest array maintained in the column ElementsGeo. Since the array length depends on the number of elements a given object has, it is more likely that arrays have different lengths in distinct rows of the table T. For example, in Figure 5.11, $GiST_4$ has only one entry corresponding to the fourth polygon of the first row in T. Therefore, a policy should be developed to avoid the construction of a GiST when the number of geometries to be indexed is very low, as a sequential scan can be more advantageous than an index scan.

Compared to the approaches detailed in Sections 5.2.1 to 5.2.2, the pair of arrays has the advantage of avoiding joins between tables to process queries since the columns are in the table T. The pair of arrays also fit vague spatial attributes whose both the certitude and the dubiety are monovalued, since the arrays stored in the columns ElementsGeo and ElementsMval hold at most two entries. The feasibility of building indices is an advantage of the pair of arrays over the UDT addressed in Section 5.2.3. Another advantage over the UDT is that calls to the constructor ROW are not required. However, like the UDT, the column VSElementGeo is of type geometry array and therefore cannot be registered in OGC's metadata stored in the DBMS. Furthermore, the calls to the ARRAY constructor are essential to extract both the geometries and the membership values of elements from the columns VSElementGeo and VSElementMval. These calls add an overhead to query processing. The aforementioned drawbacks motivated the logical design proposed in Section 5.2.5.

5.2.5 One Multiple Geometry And One Array of Membership Values

In this section, the vague spatial attribute is designed using two columns. The first column has a type that allows multiple geometries, while the second column is an array of membership values. Each element of the object is described both by the n -th geometry in the first column and by the n -th membership value in the second column. For each object, the cardinality of geometries in the first column is equal to the length of the array in the second column. Thus,

a vague spatial attribute whose both the certitude and the dubiety are monovalued represents objects with at most two geometries and an array of at most two entries. The complementary column MergedElementsGeo stores the geometric union of the certitude and the dubiety of each object.

Figure 5.12 depicts the table T containing the columns TPK where the primary key is defined, MergedElementsGeo, ElementsGeo and ElementsMval. The names of the columns ElementsGeo and ElementsMval have the prefix Elements in the plural to emphasize that they support multiple values. The column ElementsGeo has the superclass Geometry as type, but in fact is casted to one of its subclasses that support multiple geometries, i.e. MultiPoint, MultiLinestring, MultiPolygon or GeometryCollection. The columns MergedElementsGeo and ElementsGeo can be both registered in the OGC's metadata and indexed. The column ElementsMval represents the array of membership values and can also be indexed. The value of each entry of an array in ElementsMval is either in]0,1] if it denotes a membership degree, or in {-1,1} where -1 and 1 mean *maybe* and *true*, respectively.

T
<p style="text-align: center;"> <u>TPK</u> : int MergedElementsGeo : geometry ElementsGeo : geometry ElementsMval : float array ... </p>

Figure 5.12: The vague spatial attribute designed as a column for multiple geometries and a column for the array of membership values.

Figure 5.13 depicts one vague region and its storage in the table T as well as highlights separate geometries fetched from the row where TPK=1, using OGC's function GeometryN. The function GeometryN returns the n-th geometry in ElementsGeo. The corresponding n-th membership value in ElementsMval is also highlighted. Note that MergedElementsGeo stores one polygon while ElementsGeo holds a multipolygon.

The spatial index built on the column MergedElementsGeo is identical to the one built on the column ElementsGeo, because in every row of T the MBR of the multiple geometry is equal to the MBR of the merged geometry. Consider a vague spatial predicate that fetches objects, as IRQ_{obj} and CRQ_{obj} . On the one hand, there is redundancy due to the storage of these identical indices and their filter steps to process a predicate using a spatial index would also be identical. On the other hand, the merged geometry is less complex than the multiple geometry because the geometric union removes useless vertices. Therefore, if the predicate requires a refinement step, fetching the merged geometry tends to be less costly than fetching the multiple geometry.

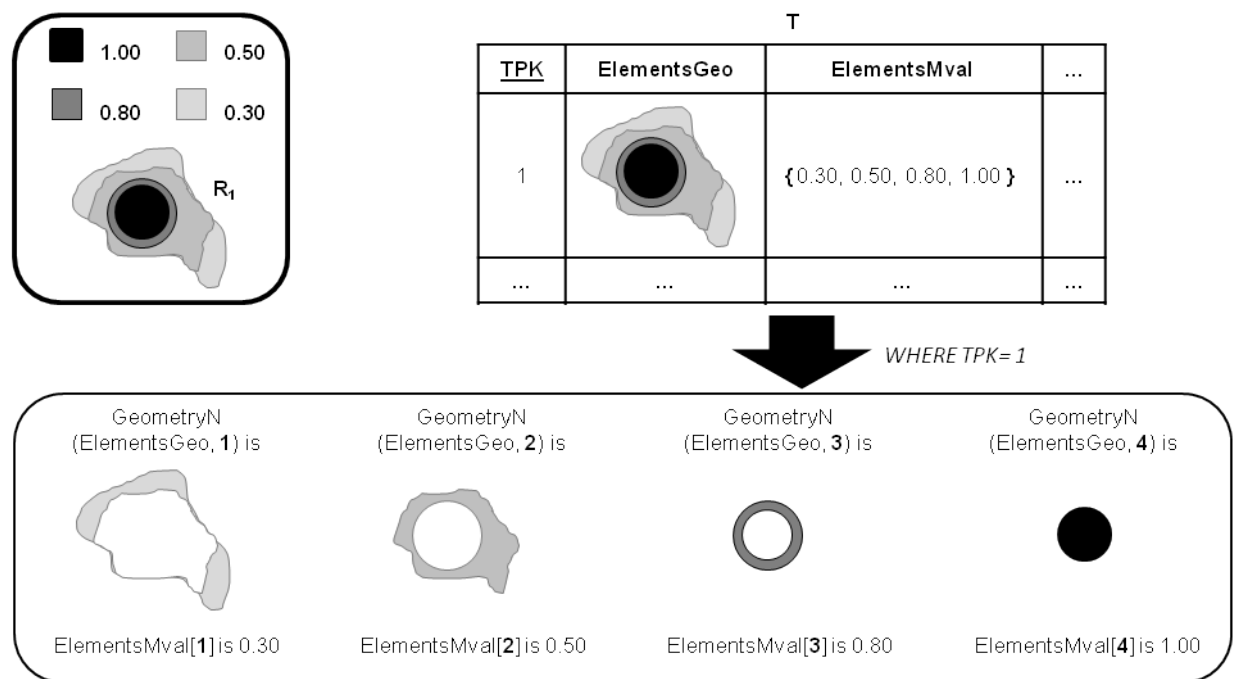


Figure 5.13: A vague region stored as a multiple geometry and an array of membership values.

Therefore, the designer of the vague SDW can opt to keep the column $MergedElementsGeo$ if the merged geometry is significantly less complex than the multiple geometry in $ElementsGeo$, aiming to benefit the performance to process predicates as IRQ_{object} and CRQ_{object} .

The spatial index built on the column $ElementsGeo$ approximates the multiple geometry of each object with one MBR, but the elements of certitude and dubiety of the member do not have individual approximations built. On the one hand, this fact prevents overlapping among several MBRs. On the other hand, elements of the certitude and dubiety cannot be queried with an index scan. As a result, processing vague spatial aggregation functions and vague spatial predicates may demand consecutive calls to the function $GeometryN$. To prevent a low performance, the implementations of vague spatial aggregation functions and vague spatial predicates should avoid fetching the n -th geometry by means of a sequential scan. For example, an $IRQ_{dubiety_elements}$ requires the elements of the dubiety to be retrieved. Thus, its implementation should avoid consecutive n calls to the function $GeometryN$ to test each element of an object against the spatial query window.

The vague spatial attribute designed as one column for multiple geometries and one column for the array of membership values eliminate the undesirable joins between tables introduced by the logical design proposed in Sections 5.2.1 and 5.2.2. It also allows the geometries to be both registered in OGC's metadata and indexed, in contrast with the approaches detailed in Sections 5.2.3 and 5.2.4. Nevertheless, an overhead is expected due to the execution of the func-

tion GeometryN and the constructor ARRAY to access the elements of the object. Section 5.2.6 describes an alternative to design the vague spatial attribute whose both the certitude and the dubiety are monovalued, while Section 5.2.7 details an unfeasible attempt to use 2D geometries with measure and 3D geometries aiming to replace the array of membership values.

5.2.6 Monovalued Certitude and Monovalued Dubiety

In order to represent vague spatial attributes whose both the certitude and the dubiety are monovalued, the alternative logical design shown in Figure 5.14 is proposed. The table T has the primary key defined on the column TPK, the column MergedElementsGeo stores the geometric union of the certitude and the dubiety, the column CertitudeGeo denotes the monovalued certitude, the column DubietyGeo represents the monovalued dubiety and the column DubietyMval indicates the membership value of the dubiety, since the membership value of the certitude is assumed to be 1.0.

The columns CertitudeGeo and DubietyGeo have the superclass Geometry as type, which is in fact casted to one of its subclasses that support a single geometry, i.e. Point, Linestring or Polygon. The columns MergedElementsGeo, CertitudeGeo and DubietyGeo can be registered in OGC's metadata and indexed. The column DubietyMval should not exist if the dubiety of the vague spatial attribute means *maybe*. For example, Figure 5.15 highlights a vague region and illustrates its storage in the table T. The column DubietyMval is not in the table T because the vague spatial attribute's dubiety means *maybe*.

The alternative logical design of the vague spatial attribute proposed in this section to address monovalued certitude and monovalued dubiety eliminate the undesirable joins between tables introduced by the logical design proposed in Sections 5.2.1 and 5.2.2. It allows the geometries to be both registered in OGC's metadata and indexed, in contrast with the approaches detailed in Sections 5.2.3 and 5.2.4. Furthermore, it eliminates additional overheads due to the execution of DBMS internal functions to access the elements of the object, which are intrinsic of the logical designs described in Sections 5.2.3 to 5.2.5. In the following, Section 5.2.7 details an unfeasible attempt to use 2D geometries with measure and 3D geometries aiming to embed geometries and membership values together.

5.2.7 2D Geometry With Measure or 3D Geometry

In this section, an attempt to use 2D geometries with measure and 3D geometries is described aiming to replace the array of membership values used by the logical design addressed

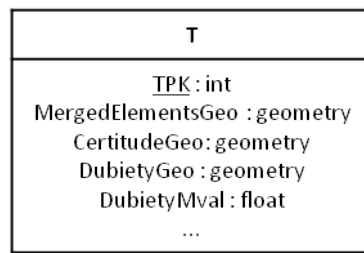


Figure 5.14: The logical design for the vague spatial attribute with monovalued certitude and monovalued dubiety.

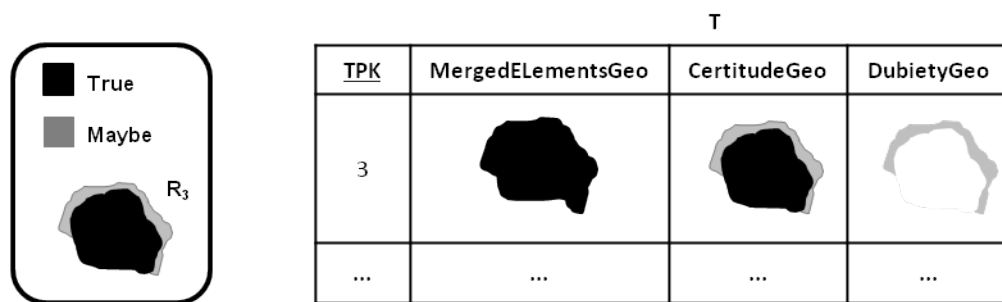


Figure 5.15: The storage of a vague region whose both certitude and dubiety are monovalued.

in Section 5.2.5. Although OGC provides and DBMSs implements the 2D geometry type with measure (X, Y, M) and the 3D geometry type (X, Y, Z), these data types are not adequate to represent vague spatial objects. The following examples illustrate the reasons for such inadequacy. The vague point set shown in 5.16a can be represented both as a 2D multipoint with measure or as a 3D multipoint, since the membership value can be the measure or the ordinate Z of each element, respectively.

On the other hand, a vague line cannot be represented by a 2D linestring with measure because the value of a measure is assigned to a point by default. As a result, it is not possible to assign a membership value to segments of a 2D linestring with measure. Also, by definition, the vertices of a vague line do not have membership values. Figure 5.16b shows a 2D linestring with measure whose vertices do not have membership values to comply with the definition of vague line. Also, the linestring’s segments do not have membership values because these are not allowed. The same problem occurs with a 3D linestring, since the Z ordinate is also assigned to a point (vertex) by default.

Analogously, a 2D multipolygon with measure requires the redundant storage of the membership value for every vertex of each element belonging to a vague region. In Figure 5.16c, all vertices of the hexagon have the same membership value 1.0, while all vertices of the trapezoid have the same membership value 0.6. The same problem occurs with the 3D multipolygon.

Vague spatial aggregation functions and vague spatial predicates may require a frequent use of DBMS internal functions to separately extract the 2D geometry and the membership value both from a 2D geometry with measure and from a 3D geometry. The execution of these functions can impair the performance. In addition, OGC standards state that each (X,Y) coordinate carries an M ordinate value that is part of a linear reference system (as already discussed in Section 2.1.2.2). However, a linear reference system refers to the storage and maintenance of events that occur along a network, which is not the case for the vague spatial attribute.

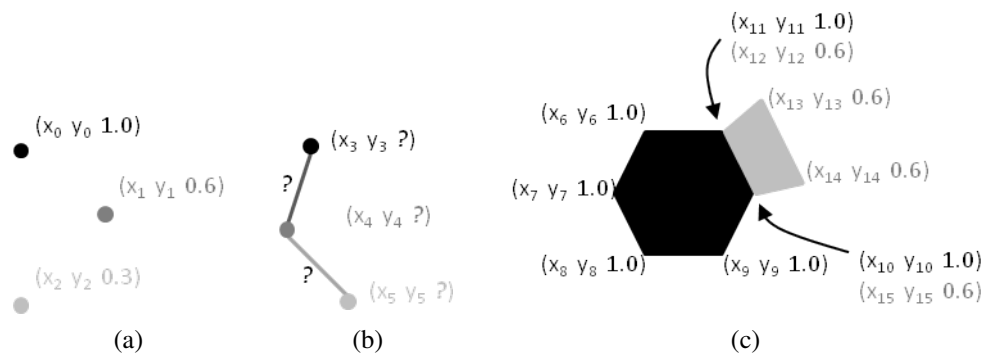


Figure 5.16: Shortcomings of both the 2D geometry type with measure and the 3D geometry type: (a) A vague point set. (b) A vague line. (c) Two elements of a vague region.

5.2.8 Discussion

In Sections 5.2.1 to 5.2.7, different approaches for the logical design of the vague spatial attribute have been described and duly discussed. Table 5.1 lists each approach, the goals the approaches should have achieved and which goals are achieved by each approach. Since the approach detailed in Section 5.2.7 has already been considered not feasible, it is not listed in Table 5.1. In addition, two criteria have also been established. Firstly, joins between tables have been considered more harmful to performance than calls to internal DBMS functions. Secondly, the compliance with OGC has been considered essential to allow compatibility with other applications.

As a result, the logical design of the vague spatial attribute as a multiple geometry and one array of membership values described in Section 5.2.5 has been chosen. Although it depends on calling internal functions of the DBMS as GeometryN and the constructor ARRAY to retrieve elements of a vague spatial object, it is able to represent vague spatial attributes with monovalued or multivalued certitude and dubiety. Alternatively, the approach detailed in Section 5.2.6 can be strictly employed when the vague spatial attribute has both the certitude and the dubiety

Table 5.1: The approaches described in Sections 5.2.1 to 5.2.6 and the goals they achieve.

Logical design	OGC compliance	Mono/Multi-valued elements	Minimization of joins	Indexing	Minimization of function calls	Sorting of elements	Union of elements
Separate tables (Section 5.2.1)	✓	✓		✓	✓	✓	✓
A single table (Section 5.2.2)	✓	✓		✓	✓	✓	✓
User-defined types (Section 5.2.3)		✓	✓			✓	✓
A pair of arrays (Section 5.2.4)		✓	✓			✓	✓
Multiple geometry and array (Section 5.2.5)	✓	✓	✓	✓		✓	✓
Monovalued components (Section 5.2.6)	✓		✓	✓	✓	✓	✓

monovalued.

5.3 Vague Spatial Attribute

The logical design of a vague spatial attribute into a table of the vague SDW complies with Section 5.2.5 and is guided by Rule 1VS, as follows.

Definition 5.3.1. Rule 1VS. A vague spatial attribute is defined into a table and comprises:

- one geometry column to represent multiple geometries of certitude elements and dubiety elements. The column assumes one of the following OGC types, depending on the type used in the conceptual schema designed according to the VSMultiDim model:
 - Point to represent either a simple vague point or a simple fuzzy point;
 - LineString to represent either a simple vague line or a simple fuzzy line;
 - Polygon to represent either a simple vague region or a simple fuzzy region;
 - MultiPoint to denote either a vague point set or a fuzzy point set;
 - MultiLineString to denote either a vague line set or a fuzzy line set; or
 - MultiPolygon to denote either a vague region set or a fuzzy region set;
- one column of type array of real numbers, such as float array, to denote membership values of elements
- an optional geometry column to store the union of the geometries of all elements; and

- constraints on these three columns that are satisfied by all rows of the table.

This implementation of the vague spatial attribute focuses on arbitrary geometries and pre-computed membership values. It mainly supports vague spatial objects modeled according to exact models surveyed in Section 2.3.1 and spatial plateau objects surveyed in Section 2.3.3.1. Therefore, a wide set of vague spatial data types is supported. The type of the column that stores the union of geometries is selected by identifying which type of geometry is expected after such union.

Loading a table holding a vague spatial attribute requires to carefully associate the i -th element of a vague spatial object to the i -th entry in the array of membership values. An example is detailed in Section 5.4. A vague spatial attribute can be defined in a *vague spatial level table* as described in Section 5.4, or assume the role of a vague spatial measure and be defined in a *fact table* as described in Section 5.6. The following constraints are satisfied by all rows of the table and involve the three columns that represent a vague spatial attribute:

- membership values are not null.
- membership values are in the range $]0, 1]$ for denoting degree of membership, or they are in $\{-1, 1\}$ for denoting *maybe* and *true*, respectively;
- membership values are sorted (in ascending or descending order);
- the number of geometries of elements is equal to the length of the array of membership values;
- the interiors of geometries of elements are disjoint;
- the merged geometry is identical to the union of the geometries of elements; and
- the geometry column representing elements, the array column of membership values, and the column for the merged geometry are all of them not null, or they are all of them null.

The aforementioned constraints extend the constraints already specified for the vague spatial attribute of the VSCube model. A routine to check these constraints is essential. The routine has been implemented as an UDF using the procedural language PL/pgSQL from PostgreSQL. The execution of the function is managed by a trigger associated to the table where the vague spatial attribute is. While the function is independent of application and is created only once, the number of triggers is equal to the number of vague spatial attributes, to ensure the integrity. The

function described in Listing 5.1 executes before the DBMS commits the insertion or update for each row inserted or updated in a table containing a vague spatial attribute.

It considers that the vague spatial attribute has been designed according to Rule 1VS. It receives 4 arguments in the array `TG_ARGV`, which are omitted from the function's signature but that are explicit in the trigger. `TG_ARGV[0]` is the name of the geometry column of the elements, `TG_ARGV[1]` is the name of the column with the array of membership values, `TG_ARGV[2]` is the name of the geometry column containing merged geometries of elements, and `TG_ARGV[3]` is a flag for indicating if the vague spatial attribute has been designed with membership values in $]0, 1]$.

The record `NEW` denotes the row being inserted or updated. Thus, it holds values of the columns whose names match those provided as arguments. Whenever a constraint is violated, an exception is raised, the execution is interrupted, the operation is canceled and an error message is issued. Internal comments in the source code benefit the comprehension of the function. In detail, the function `VS_Constraints_VSAttribute` described in Listing 5.1 checks the constraints of a vague spatial attribute in eight steps, as follows.

First, it retrieves the values of arguments and assign them to local variables representing the multiple geometry containing geometries of elements, the array of membership values, and the merged geometry of elements. The query issued with the `EXECUTE` command basically replaces `$1` by `NEW`, fetches the values of each field in `NEW` and copies to the corresponding local variables.

Second, `VS_Constraints_VSAttribute` ensures that the geometry of elements, the array of membership values and the merged geometry are all of them either not or null. Clearly, if all of them are null, there is no reason to continue verifying the constraints, and the execution of the function is terminated with success. However, if some of them are null, the execution is interrupted.

Third, it verifies whether the membership values are in $]0, 1]$ or in $\{-1, 1\}$ before calling a function that checks if the membership values in the array are valid. The function `VS_Constraints_FuzzyMval` described in Appendix A is called if membership values are in $]0, 1]$, while the function `VS_Constraints_ExactMval` described in Appendix A is called if membership values are in $\{-1, 1\}$. To sum up, these functions return the length of the array of membership value passed as argument if all membership values are valid. Otherwise, they return -1. The result of the execution of one of these functions is assigned to a local variable, which is tested to decide whether to interrupt the execution due to an invalid membership value, or to continue.

Listing 5.1: A routine to check the constraints of a vague spatial attribute.

```

CREATE OR REPLACE FUNCTION VS_Constraints_VSAttribute() RETURNS TRIGGER AS $$
DECLARE
    length bigint;
    ElementsGeo geometry;
    ElementsMval float array;
    MergedElementsGeo geometry;
    isFuzzy text;
BEGIN
--1. retrieves values of arguments and assigns to local variables
EXECUTE 'SELECT ($1).' || TG_ARGV[0] || ', ($1).' || TG_ARGV[1] || ', ($1).' || TG_ARGV[2]
USING NEW INTO ElementsGeo, ElementsMval, MergedElementsGeo;

--2. avoids null geometries or null array of membership values
IF (ElementsGeo IS NULL) AND (ElementsMval IS NULL) AND (MergedElementsGeo IS NULL) THEN
    RETURN NEW;
ELSE
    IF (ElementsGeo IS NULL) OR (ElementsMval IS NULL) OR (MergedElementsGeo IS NULL) THEN
        RAISE EXCEPTION 'Null geometries or array of membership values.';
    END IF;
END IF;

--3. checks whether membership values are valid
length:= 0;
isFuzzy=TG_ARGV[3];
IF (isFuzzy='1') THEN
    length:= VS_Constraints_FuzzyMval(ElementsMval);
ELSE
    length:= VS_Constraints_ExactMval(ElementsMval);
END IF;
IF (length=-1) THEN
    RAISE EXCEPTION 'Invalid membership values.';
END IF;

--4. verifies whether the array of membership values is sorted
IF NOT (CheckOrderedArray(ElementsMval)) THEN
    RAISE EXCEPTION 'Membership values must be sorted in ascending order.';
END IF;

--5. verifies if an element has both a geometry and a membership value
IF (length <> ST_NumGeometries(ElementsGeo)) THEN
    RAISE EXCEPTION 'Every geometry must have a corresponding membership value.';
END IF;

--6. verifies if the merged geometry is equal to the union of multiple geometries
IF NOT ST_Equals(ST_UnaryUnion(ElementsGeo), MergedElementsGeo) THEN
    RAISE EXCEPTION 'The union of the geometries from elements does not correspond to '
        || 'the merged geometry provided.';
END IF;

--7. detects if interiors of elements intersect
IF InteriorIntersection(ElementsGeo) THEN
    RAISE EXCEPTION 'The interiors of different elements must be disjoint.';
END IF;

--8. proceeds with insertion if constraints have been satisfied
RETURN NEW;
END;
$$ LANGUAGE 'plpgsql';

```

Fourth, a call to the function `CheckOrderedArray` described in Appendix A verifies whether the array of membership values is sorted. The function `CheckOrderedArray` returns true if the array is sorted in ascending order, or false otherwise. If the array is not sorted, the execution is interrupted.

Fifth, it ensures that every element has both a geometry and a membership value. A sufficient condition for this is: the array of membership values has a number of entries equal to the

quantity of geometries representing each element. The number of entries of the array has been already provided by the function `VS_Constraints_FuzzyMval`, while the quantity of geometries is calculated using PostGIS' function `ST_NumGeometries`¹. If the number of entries is not equal to the quantity of geometries, then the execution is interrupted.

Sixth, it verifies if merged elements are equal to the union of all geometries of elements. The call to PostGIS' function `ST_UnaryUnion`² computes the aforementioned union, while the call to PostGIS' function `ST_Equals`³ compare the result of the union to the merged geometry. If they are not equal, the execution is interrupted.

Seventh, it detects whether the interiors of geometries from different elements intersect, by calling the function `InteriorIntersection` that is described in Appendix A. The function `InteriorIntersection` returns true if it identifies intersection, otherwise it returns false. If there is intersection, the execution is interrupted. Recall that disjoint interiors are essential for aggregation.

Finally, the insertion proceeds if the aforementioned verifications were successful and comply with the constraints. It is noteworthy that the assessments made by function `VS_Constraints_VSAttribute` are in ascending order of complexity, i.e. they start by manipulating membership values and finish by processing geometries. This design has been prioritized to prevent executing unnecessary costly geometry operations when the array of membership values is not valid.

5.4 Vague Spatial Level and Vague Spatial Member

Levels in the `VSMultiDim` conceptual model are comparable to entity types of the E-R conceptual model, as already discussed in Section 4.9. A vague spatial level contains at least one vague spatial attribute and its logical design is described by Rule 2VS, considering the vague spatial attribute as detailed in Section 5.3.

Definition 5.4.1. Rule 2VS. A vague spatial level, which is not related to a fact with a one-to-one relationship, is represented in the relational model by a table called *vague spatial level table* containing:

- one column holding a surrogate key as primary key;
- one column for each conventional attribute of the level;
- one geometry column for each crisp spatial attribute of the level;

¹http://postgis.net/docs/ST_NumGeometries.html

²http://postgis.net/docs/ST_UnaryUnion.html

³http://postgis.net/docs/ST_Equals.html

- the columns and constraints defined by the application of Rule 1VS on each vague spatial attribute of the level.
- vague topological constraints for each pair of vague spatial attributes in the level, which are satisfied by all vague spatial members.

Each row in a vague spatial level table denotes one vague spatial member.

Integrity constraints related to the vague spatial attribute are detailed in Section 5.3. In addition, topological constraints on two or more vague spatial attributes are tackled in Section 5.8.

Example 5.4.1. Figure 5.17 illustrates the vague spatial levels Crop and Infected Region and their corresponding vague spatial level tables created after applying Rule 2VS.

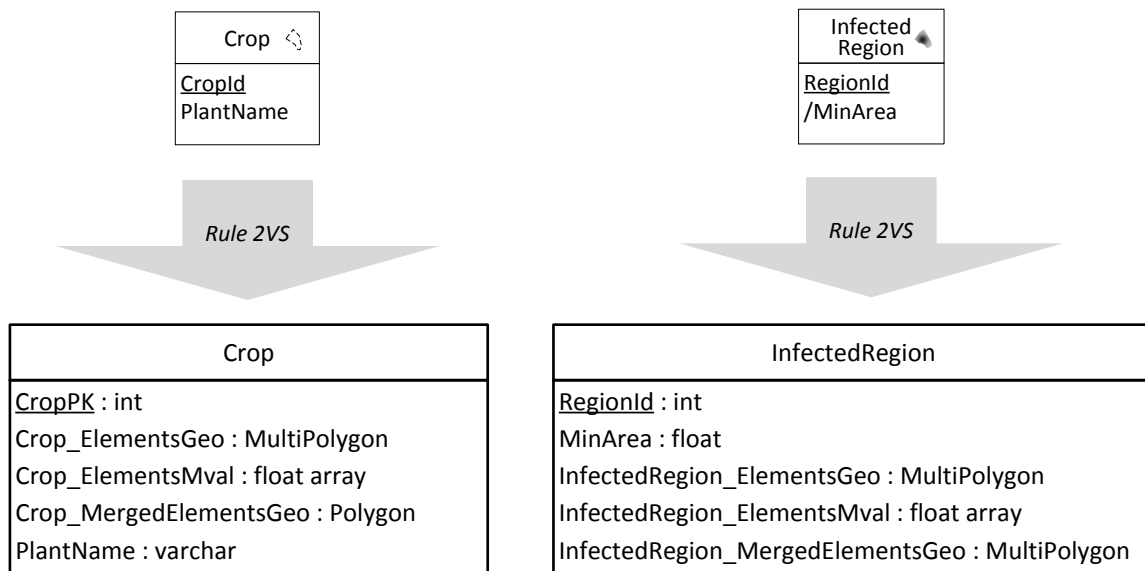


Figure 5.17: Examples of the application of Rule 2VS to obtain the vague spatial level tables Crop and Infection.

Example 5.4.2. The following SQL commands create the vague spatial level table InfectedRegion as shown in Figure 5.17.

```
CREATE TABLE InfectedRegion (
  RegionId int PRIMARY KEY,
  InfectedRegion_ElementsMval float array,
  MinArea float
);
SELECT AddGeometryColumn ('InfectedRegion', 'InfectedRegion_ElementsGeo',
  4326, 'MULTIPOLYGON', 2);
SELECT AddGeometryColumn ('InfectedRegion', 'InfectedRegion_MergedElementsGeo',
  4326, 'POLYGON', 2);
```

The creation of an application-dependent trigger per vague spatial attribute is necessary to benefit from the constraints ensured by the function previously described in Section 5.3 and Listing 5.1.

Example 5.4.3. Consider the vague spatial level tables `Crop` and `InfectedRegion` as shown in Figure 5.17. Triggers are created for their vague spatial attributes, as follows.

```
CREATE TRIGGER Crop_Constraints
BEFORE INSERT OR UPDATE ON Crop FOR EACH ROW EXECUTE PROCEDURE
VS_Constraints_VSAttribute ( 'Crop_ElementsGeo', 'Crop_ElementsMval',
                             'Crop_MergedElementsGeo', '0' );
```

```
CREATE TRIGGER InfectedRegion_Constraints
BEFORE INSERT OR UPDATE ON InfectedRegion FOR EACH ROW EXECUTE PROCEDURE
VS_Constraints_VSAttribute ( 'InfectedRegion_ElementsGeo',
                             'InfectedRegion_ElementsMval',
                             'InfectedRegion_MergedElementsGeo', '1' );
```

After the names of the columns that define the vague spatial attribute, a flag indicates whether membership values are in $]0, 1]$. Crops have membership values in $\{-1, 1\}$, and therefore the last argument passed to `VS_Constraints.VSAttribute` is 0. Conversely, infected regions have membership values in $]0, 1]$ and, therefore, the last argument passed to `VS_Constraints.VSAttribute` is 1.

As commented in Section 5.3, loading a vague spatial level table requires to carefully associate the i -th element of a vague spatial object to the i -th entry in the array of membership values. Rather than providing a routine to load vague spatial level tables, Example 5.4.4 addresses this issue and can be used as a base for developing such routine.

Example 5.4.4. Considering Examples 5.4.2 and 5.4.3, the following `INSERT` command complies with the definition of the vague spatial level table `InfectedRegion` and its constraints.

```
INSERT INTO InfectedRegion (
  RegionId,
  InfectedRegion_ElementsMval,
  InfectedRegion_ElementsGeo,
  InfectedRegion_MergedElementsGeo)
VALUES (
  1,
  ARRAY[0.5, 1.0],
  ST_Collect(
    ARRAY[
      —dubiety [0]
      ST_MakePolygon(
        —external boundary:
        ST_GeomFromText('LINESTRING(0 0, 0 10, 5 10, 5 5, 10 5, 10 0, 0 0)'),
        —hole:
        ARRAY[ST_GeomFromText('LINESTRING(1 1, 0 9, 4 9, 4 4, 9 4, 9 1, 1 1)')]
      )
    ]
  ),
```

```

—certitude [0]:
  ST_Polygon(ST_GeomFromText('LINESTRING(1 1, 0 9, 4 9, 4 4, 9 4, 9 1, 1 1)'),
            4326)
]
),
—merged geometry:
ST_GeomFromText('POLYGON((0 0,0 9,0 10,5 10,5 5,10 5,10 0,0 0))')
);

```

The value provided for `RegionId` is 1. The value provided for `InfectedRegion.ElementsMval` is an array of membership values that is sorted in ascending order. Such array specifies that there is one dubiety element and one certitude element. The value provided for `InfectedRegion.ElementsGeo` is a multiple geometry containing geometries of the elements. PostGIS' function `ST_Collect`⁴ receives an array of geometries as argument and returns a geometry. The first entry of the array is the geometry of the dubiety element, while the second entry is the geometry of the certitude element, to ensure the correspondence with the array of membership values. The geometry of the dubiety element is a polygon with a hole. It is built using the PostGIS' function `ST_MakePolygon`⁵ that receives as arguments a line for the external boundary and an array of lines that determine holes in the resulting polygon. The geometry of the certitude element is a polygon. It is built using the PostGIS's function `ST_Polygon`⁶ that receives as arguments a line and a SRID. Finally, the value provided for `InfectedRegion.MergedElementsGeo` is a polygon that represents the union of the geometries created for elements.

5.5 Hierarchies

A hierarchy is a set of relationships among pairs of levels, as discussed in Section 4.9. In this sense, the cardinality of the relationship is the key aspect addressed by Rule 3VS, as follows.

Definition 5.5.1. Rule 3VS. Let L_{parent} , and L_{child} be a pair of related levels in a hierarchy.

- Rule 3.1VS: if the relationship has the cardinality one-to-one (1:1), then all the attributes of L_{parent} are included in the table of L_{child} .
- Rule 3.2VS: if the relationship has the cardinality one-to-many (1:N), then one column is added in the table of L_{child} with a foreign key referencing the key of the table of L_{parent} .
- Rule 3.3VS: if the relationship has the cardinality many-to-many (M:N), then a bridge table is created and contains one column with a foreign key referencing the key of the

⁴http://postgis.net/docs/ST_Collect.html

⁵http://postgis.net/docs/ST_MakePolygon.html

⁶http://postgis.net/docs/ST_Polygon.html

table of L_{child} , plus one column with a foreign key referencing the key of the table of L_{parent} .

- L_{parent} and L_{child} have vague topological constraints to ensure the hierarchy, if at least one of them is a vague spatial level.

Section 5.8.2 describes the topological constraints to ensure a consistent hierarchy involving at least one vague spatial level.

Example 5.5.1. Figure 5.18 exemplifies a relationship with cardinality 1:N associating the vague spatial levels Infected Region and Infected Group. The logical design is achieved through the application of Rules 2VS, 3VS, 3.2VS, and 1VS. Rule 2VS firstly create one table for each level. Rule 3VS identifies there is a relationship between the levels of the hierarchy. Since the cardinality is 1:N, Rule 3.2VS adds the column RegionFK into the table InfectedGroup with a foreign key referencing the key RegionId of the table InfectedRegion. Finally, Rule 1VS creates columns for the vague spatial attributes.

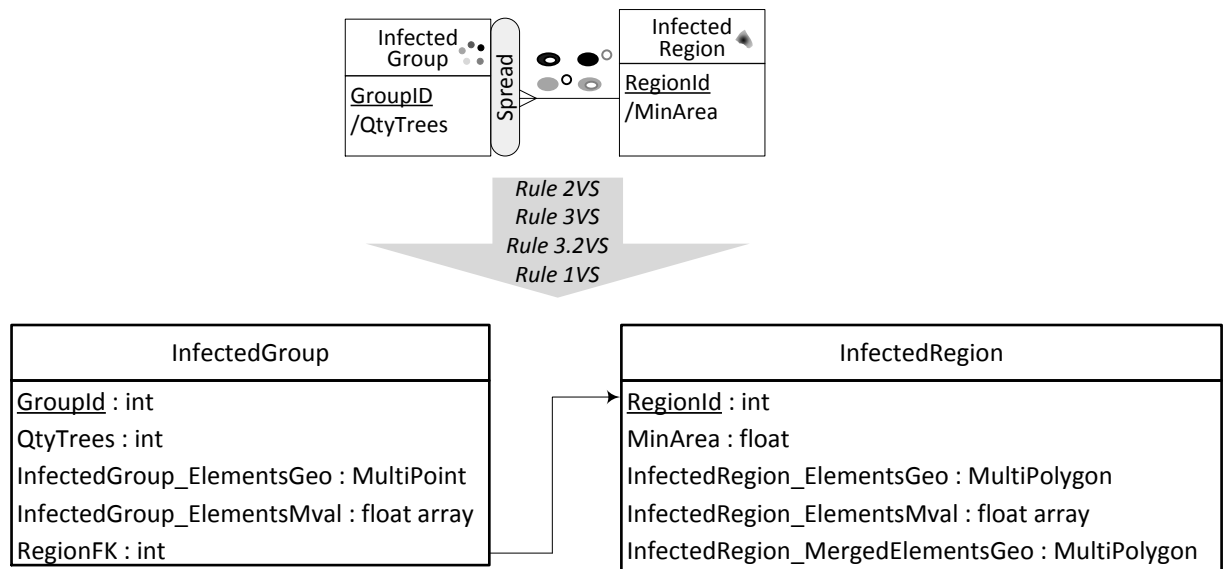


Figure 5.18: Mapping rules applied to a pair of related vague spatial levels.

5.6 Fact and Vague Spatial Measure

A fact in the VSMultiDim conceptual model is a relationship among levels and holds measures, as discussed in Section 4.9.4. The logical design for a fact is guided by Rule 4VS, as follows.

Definition 5.6.1. Rule 4VS. A fact F is mapped to a *fact table* containing:

- one column for each conventional measure;
- one geometry column for each crisp spatial measure;
- the columns and constraints defined by the application of Rule 1VS on each vague spatial measure;
- considering that a level L is related to F , then:
 - Rule 4.1VS: if the relationship between L and F has the cardinality one-to-one (1:1), then all the attributes of L are included in the fact table.
 - Rule 4.2VS: if the relationship between L and F has the cardinality one-to-many (1:N), then one column is added in the fact table with a foreign key referencing the key of the table of L .
 - Rule 4.3VS: if the relationship between L and F has the cardinality many-to-many (M:N), then a bridge table is created and contains one column with a foreign key referencing the key of the fact table, plus one column with a foreign key referencing the key of the table of L .
- vague topological constraints regarding vague spatial measures, which are satisfied by all fact members;
- vague topological constraints concerning spatial levels referenced by the fact table (due to a spatial fact), which are satisfied by all fact members; and
- either a composite key that encompasses the columns that reference the level tables through foreign keys, or a surrogate key.

In addition, each row of a fact table corresponds to a fact member.

A vague spatial measure is a vague spatial attribute and, therefore, has the integrity constraints described in Section 5.3. Vague topological constraints regarding pairs of vague spatial measures are tackled in Section 5.8.4. Vague topological constraints concerning levels referenced by the fact table are addressed in Section 5.8.3.

Example 5.6.1. Figure 5.19 illustrates the logical schema of the vague SDW created for the HLB case study. It has been obtained by applying the aforementioned mapping rules over the conceptual schema shown in Figure 4.27. The relationship between the vague spatial level

Infected Group and the fact HLB Control has cardinality 1:N. Therefore, Rule 4.2VS adds the column GroupFK into the fact table HLBControl with a foreign key referencing the key GroupId of the vague spatial level table InfectedGroup. Rule 1VS creates columns for the vague spatial attributes, but note that a column for the geometric union was not defined for Infected Tree and Infected Group. The rationale is that the geometric union would be identical to the multiple geometry.

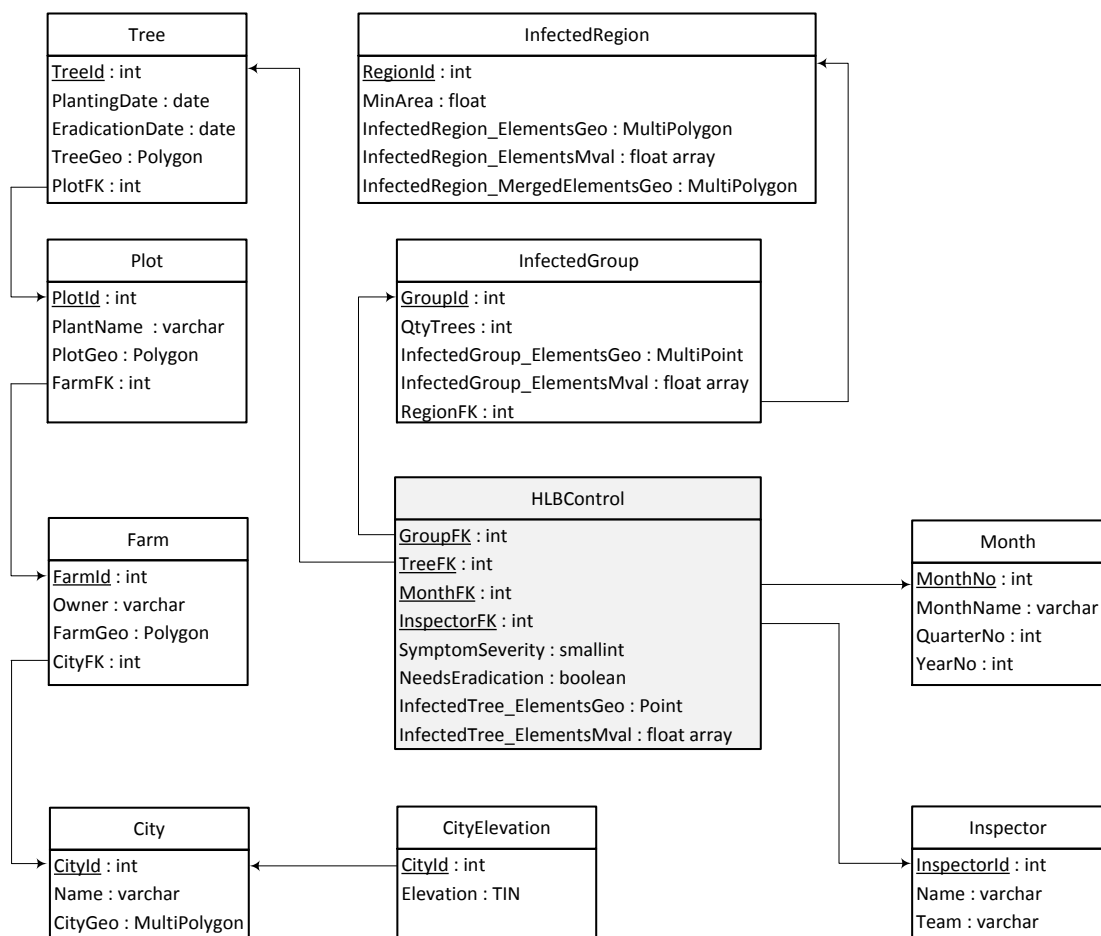


Figure 5.19: The logical schema of the vague SDW regarding the HLB disease.

5.7 Vague Spatial Fact

A vague spatial fact has a finer granularity than a fact and enables the analysis of measures at the grains of certitude elements and dubiety elements of vague spatial objects, as discussed in Section 4.4.4. The logical design of the vague spatial fact produces a *vague spatial fact table* that references at least one vague spatial level table. Then, values of measures in the vague

spatial fact table indicate measure values regarding the referenced vague spatial members as well as partial measure values associated to certitude elements and dubiety elements of the referenced vague spatial members.

Section 5.7.1 addresses the relational representation of a vague spatial fact as a *vague spatial fact table* containing *vague spatial fact sets*, Section 5.7.2 details design issues concerning numeric measures and crisp spatial measures, Section 5.7.3 details design issues concerning vague spatial measures, Section 5.7.4 addresses two methods for loading a vague spatial fact table, and Section 5.7.5 discusses important aspects of the logical design proposed.

5.7.1 Relational Representation

The logical design of a vague spatial fact requires the creation of a *vague spatial fact table* according to Rule 5VS, as follows.

Definition 5.7.1. Rule 5VS. The table *VFact* is a vague spatial fact table whose rows assign partial values of the set of measures $\{M_1, \dots, M_s\}$ to members of the set of vague spatial level tables $\{L_1, \dots, L_t\}$, for $s > 0$ and $t > 0$ such that:

- l_j^v is a vague spatial member of L_j , i.e. a row in L_j , for $0 < j \leq t$ and $0 < v \leq |L_j|$, where $|L_j|$ is the number of rows in L_j ;
- The column holding the primary key of L_j is $L_j.PK$ and a vague spatial attribute A in L_j comprises the columns $L_j.A.ElementsGeo$, $L_j.A.ElementsMval$ and $L_j.A.MergedElementsGeo$;
- in l_j^v , the primary key value is in $l_j^v.PK$, the elements' geometries are in $l_j^v.A.ElementsGeo$, the elements' membership values are in $l_j^v.A.ElementsMval$, and the merged elements' geometries are in $l_j^v.A.MergedElementsGeo$;
- the number of geometries in $l_j^v.A.ElementsGeo$ is $|l_j^v.A.ElementsGeo|$, while the number of entries in the array $l_j^v.A.ElementsMval$ is $|l_j^v.A.ElementsMval|$;
- $|l_j^v.A.ElementsGeo|$ is equal to $|l_j^v.A.ElementsMval|$ and both correspond to the number of elements of l_j^v according to A ;
- the column L_jFK in *VFact* has a foreign key referencing the column $L_j.PK$;
- a column $L_jA.ElementNum$ is created in *VFact* for each vague spatial level L_j that is referenced by *VFact* and that contains a vague spatial attribute A .
- every row in *VFact* assumes one of the following values in the column $L_jA.ElementNum$:

- 0, if it is a fact member;
- n such that $1 \leq n \leq |l_j^v.A_ElementsGeo|$, if it associates a partial measure value to the n -th element described by $l_j^v.A_ElementsGeo$ and $l_j^v.A_ElementsMval$.
- a vague spatial fact table inherits all the constraints of the corresponding fact table, except for the primary key.
- the primary key of *VSFact* is a composite primary key that encompasses the primary key of the corresponding fact table and the column *L_jA_ElementNum*.

A fact member in *VSFact* is a single row with *L_jA_ElementNum* = 0 that relates members from distinct levels and assigns values for each measure M_i in $\{M_1, \dots, M_s\}$. Furthermore, a *vague spatial fact set* in *VSFact* encompasses the fact member and a set of rows with *L_jA_ElementNum* = n that associates values of each measure M_i to the n -th element of the vague spatial member l_j^v described by $l_j^v.A_ElementsGeo$ and $l_j^v.A_ElementsMval$.

The VSCube conceptual model considers, in Section 4.4.4, that a vague spatial fact exists for every vague spatial attribute in a cuboid. Rule 5VS preserves such characteristic because it allows the vague spatial fact table to reference more than one vague spatial level table, i.e. $\{L_1, \dots, L_t\}$, and to store partial measure values for elements of a referenced vague spatial member.

Each measure in $\{M_1, \dots, M_s\}$ of the vague spatial fact table can be represented by a single column if it is a numeric measure or a crisp spatial measure, as described in Section 5.7.2. Conversely, a vague spatial measure requires at most three columns, as detailed in Section 5.7.3.

5.7.2 Numeric Measures and Crisp Spatial Measures

Considering Rule 5VS detailed in Section 5.7.1, let M_i be a numeric measure in $\{M_1, \dots, M_s\}$ represented by the column M_i in *VSFact*, for $1 \leq i \leq s$. Let also agg_i be an aggregation function used to summarize the values of M_i . A fact member in *VSFact* that references a member l_j^v has:

- $L_jFK = l_j^v.PK$;
- $L_jA_ElementNum = 0$; and
- $M_i = m_{i,0}$.

A vague spatial fact set in *VSFact* encompasses the aforementioned fact member and a set of rows where:

- $L_jFK = l_j^v.PK$;
- $L_jA_ElementNum > 0$;
- $M_i = m_{i,n}$; and
- $1 \leq n \leq |l_j^v.A_ElementsGeo|$.

Furthermore, the aggregation of partial measure values produces values identical to the values of the fact member, i.e.:

- $agg_i(m_{i,1}, \dots, m_{i,|l_j^v.A_ElementsGeo|}) = m_{i,0}$.

Example 5.7.1. Figure 5.20a shows the vague spatial fact table PesticideApplicationVSFact whose rows assign partial values of the unitary set of measures {AppliedTons} to members of the unitary set of vague spatial level tables {Crop}. The table PesticideApplicationVSFact has the column CropElementNum, which has been added according to the definition provided in Section 5.7.1. Note that CropElementNum is one of the columns that compose the primary key of PesticideApplicationVSFact. The inclusion of CropElementNum in the primary key enables, for example, CropFK = 1 to appear in several rows, while the pair of values for CropFK and CropElementNum can be repeated only if at least one value for PesticideFK or DateFK is modified.

Considering the crop C_1 and the pesticide application A_1 as exemplified in Figure 1.2a-c, the corresponding vague spatial fact set is shown in Figure 5.20b. Note that crop C_1 has three elements: C_{11} , C_{12} and C_{13} . Then, at most $1+3=4$ rows will compose the vague spatial fact set. The first row in PesticideApplicationVSFact is the fact member and has CropElementNum=0 and the total of 0.30 applied ton. The other two rows complete the vague spatial fact set and assign partial values of AppliedTons to elements of C_1 . The second row refers to the 0.05 ton applied over the single element of the dubiety of the crop C_1 depicted in Figure 1.2c, i.e. C_{13} . The third row refers to 0.25 ton applied over one element of the certitude of the crop C_1 depicted in Figure 1.2b, i.e. C_{11} . A row for the other element of the certitude of C_1 is unnecessary to denote 0.00 ton. AppliedTons is aggregated using SUM, and it is clear that the sum of the partial values regarding AppliedTons in the second and third rows ($0.05+0.25$) is equal to the total value of AppliedTons in the first row (0.30).

A similar approach also applies if M_i is a spatial measure represented by one geometry column M_i in the vague spatial fact table. Each value $m_{i,n}$ assigned to the column M_i is the geometric intersection between $m_{i,0}$ and the geometry in $l_j^v.A_MergedElementsGeo$. Furthermore, $Union(m_{i,1}, \dots, m_{i,|l_j^v.A_ElementsGeo|}) = m_{i,0}$.

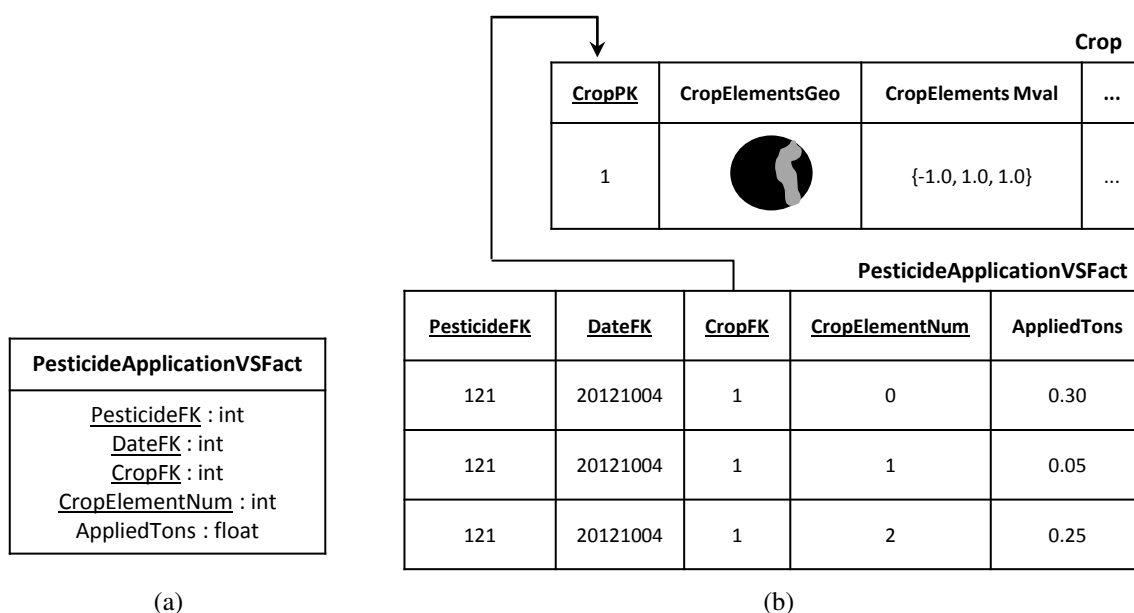


Figure 5.20: A vague spatial fact table. (a) The schema. (b) A vague spatial fact set.

5.7.3 Vague Spatial Measures

Considering Rule 5VS detailed in Section 5.7.1, let M_i be a vague spatial measure in $\{M_1, \dots, M_s\}$, for $1 \leq i \leq s$, represented by the following columns in $VSFact$: $M_ElementsGeo_i$, $M_ElementsMval_i$ and $M_MergedElementsGeo_i$, which correspond to geometries of elements, an array of membership values of elements and the merged geometries of the elements, respectively. Let also $agg_Elements_i$ be an aggregation function used to summarize the values of $M_ElementsGeo_i$ and $M_ElementsMval_i$ (e.g. vague spatial union described in Section 5.9.3) and $agg_MergedElements_i$ be an aggregation function used to summarize the values of $M_MergedElementsGeo_i$ (e.g. union).

A fact member in $VSFact$ that references a member l_j^v has:

- $L_jFK = l_j^v.PK$;
- $L_jA_ElementNum = n$;
- $M_ElementsGeo_i = g_{i,0}$;
- $M_ElementsMval_i = v_{i,0}$; and
- $M_MergedElementsGeo_i = o_{i,0}$.

A vague spatial fact set in $VSFact$ encompasses the aforementioned fact member and a set of rows where:

- $L_jFK = l_j^v.PK$;
- $L_jA_ElementNum > 0$;
- $M_ElementsGeo_i = g_{i,n}$;
- $M_ElementsMval_i = v_{i,n}$;
- $M_MergedElementsGeo_i = o_{i,n}$; and
- $1 \leq n \leq |l_j^v.A_ElementsGeo|$.

Furthermore, the aggregation of partial measure values produces values identical to the values of the fact member, i.e.:

- $agg_Elements_i((g_{i,1}, v_{i,1}), \dots, (g_{i,|l_j^v.A_ElementsGeo|}, v_{i,|l_j^v.A_ElementsGeo|})) = (g_{i,0}, v_{i,0})$; and
- $agg_MergedElements_i(o_{i,1}, \dots, o_{i,|l_j^v.A_ElementsGeo|}) = o_{i,0}$.

The partial values of a vague spatial measure, where $L_jA_ElementNum > 0$, are also given by one multiple geometry, one array of membership values and one merged geometry, as follows.

- $g_{i,n}$ is a multiple geometry composed of p geometries gathered from the p geometric intersections that occur between a geometry of an element in $l_j^v.A_ElementsGeo$ and a geometry of an element in $g_{i,0}$;
- $v_{i,n}$ is an array of p membership values, each one of them obtained from $v_{i,0}$ for each aforementioned intersection;
- $o_{i,n}$ is a geometry obtained by merging the p geometries of $g_{i,n}$.

Example 5.7.2. The logical schema of vague SDW created for the pest control case study is depicted in Figure 5.21. It was created by applying the mapping rules described in Sections 5.3 to 5.5 to transform the conceptual schema illustrated in Figure 4.26. The numeric measure AppliedTons has been also included according to Section 5.7.2. The vague spatial fact table PesticideApplicationVSFact created according to Rule 5VS allows the assignment of partial values of the set of measures $\{AppliedTons, AppliedArea\}$ to members of the unitary set of vague spatial level tables $\{Crop\}$. Besides, the vague spatial measure AppliedArea has been included as the columns AppliedArea.ElementsGeo, AppliedArea.ElementsMval, and AppliedArea.MergedElementsGeo. The column CropElementNum complies with Rule 5VS detailed in Section 5.7.1.

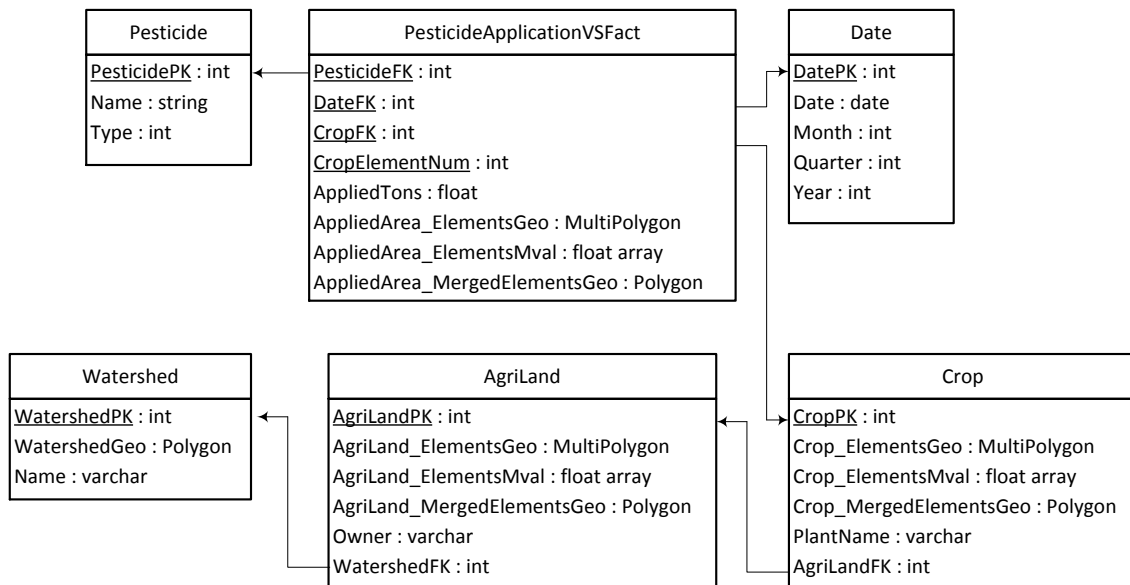


Figure 5.21: A vague spatial fact table with a vague spatial measure for areas where pesticides were applied to.

Example 5.7.3. The crop C_1 and the pesticide application A_1 exemplified in Figure 1.2a are repeated in Figure 5.22a to facilitate the visualization of intersections among the geometries of their elements. Crop C_1 has the certitude elements C_{11} and C_{12} and the dubiety element C_{13} . The applied area A_1 has one certitude element and three dubiety elements. Figure 5.22b illustrates the geometric intersection among elements of A_1 and the element C_{13} , together with the corresponding membership values. Figure 5.22c shows a single geometry obtained by merging the geometries depicted in Figure 5.22b. Figure 5.22d illustrates the geometric intersection among elements of A_1 and the element C_{11} , together with the corresponding membership values. Figure 5.22e shows a single geometry that merges the geometries depicted in Figure 5.22d. Note there is no intersection among elements of A_1 and the element C_{12} .

The corresponding vague spatial fact set is shown in the table `PesticideApplicationVSFact` in Figure 5.23, considering the vague spatial fact table designed as shown in Figure 5.21 and the vague spatial level table `Crop`. The first row of `PesticideApplicationVSFact` is the fact member that has `CropElementNum=0` and describes the total of 0.30 applied ton, multiple geometries of elements of A_1 , the associated array of membership values and a geometry merging the geometries of elements.

In addition, the remaining rows relate values of measures to elements of the crop C_1 , as follows. The second row of `PesticideApplicationVSFact` indicates that 0.05 ton was applied over the first element of C_1 , which is the single element of the dubiety, since `CropElementNum=1`. The ex-

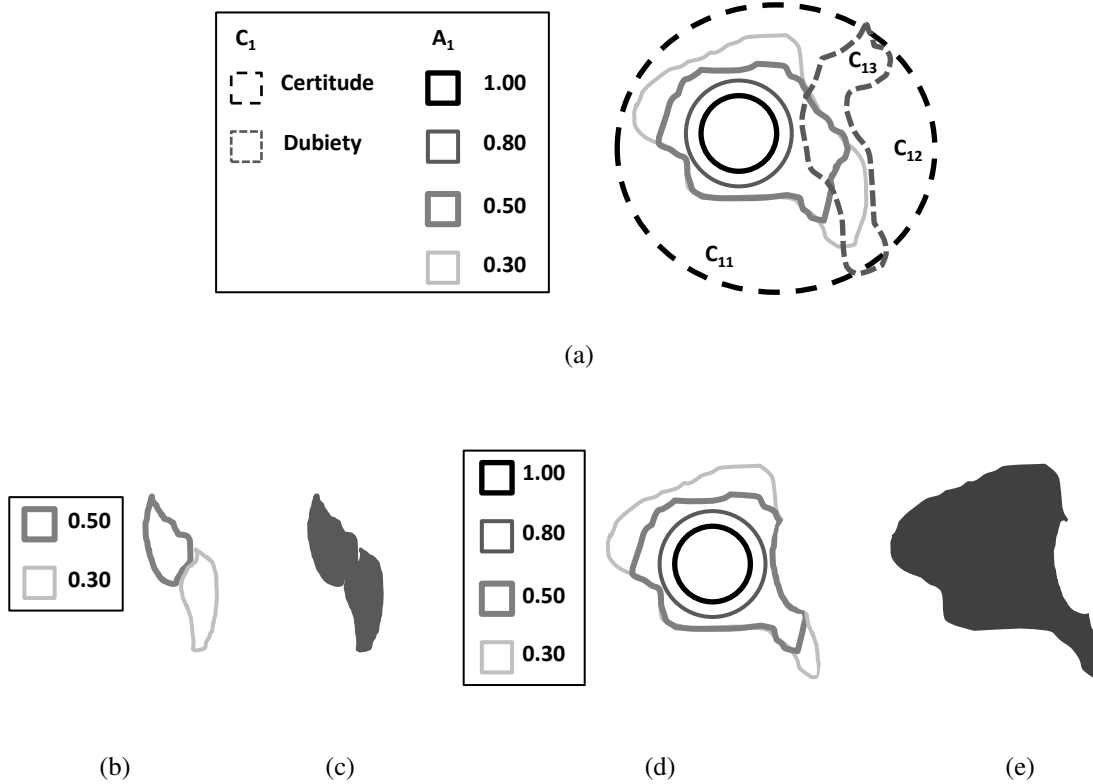
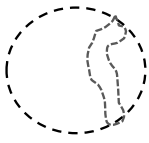


Figure 5.22: Intersections among elements of A_1 and C_1 . (a) A_1 and C_1 . (b) Intersection among elements of A_1 and C_{13} . (c) Merging the geometries from (b). (d) Intersection among elements of A_1 and C_{11} . (e) Merging the geometries from (d).

tent of pesticide application is represented by the geometries in `AppliedArea.ElementsGeo`, whose membership values are $\{0.3, 0.5\}$ stored in `AppliedArea.ElementsGeo`. These geometries and membership values are the same shown in Figure 5.22b. Also, these geometries are merged and stored in `AppliedArea.ElementsGeo`. The merged geometry is identical to the geometry shown in Figure 5.22c.

The third row of `PesticideApplicationVSFact` indicates that 0.05 ton was applied over the second element of C_1 , which is one of the elements of the certitude, since `CropElementNum=1`. The extent of pesticide application is represented by the geometries in `AppliedArea.ElementsGeo`, whose membership values are $\{0.3, 0.5, 0.8, 1.0\}$ stored in `AppliedArea.ElementsMval`. These geometries and membership values are the same shown in Figure 5.22d. Also, these geometries are merged and stored in `AppliedArea.MergedElementsGeo`. The merged geometry is identical to the geometry shown in Figure 5.22e.

The application of the corresponding aggregation functions on the columns `AppliedTons`, `AppliedArea.ElementsGeo`, `AppliedArea.ElementsMval`, and `AppliedArea.MergedElementsGeo` where `CropElementNum > 0` produces the same column values as where `CropElementNum = 0`.

Crop			
<u>CropPK</u>	CropElements Geo	CropElements Mval	...
1		{-1.0, 1.0, 1.0}	...







PesticideApplicationVSFact							
<u>PesticideFK</u>	<u>DateFK</u>	<u>CropFK</u>	<u>Crop ElementNum</u>	Applied Tons	AppliedArea_ Elements Geo	AppliedArea_ Elements Mval	AppliedArea_ MergedElements Geo
121	20121004	1	0	0.30		{0.3, 0.5, 0.8, 1.0}	
121	20121004	1	1	0.05		{0.3, 0.5}	
121	20121004	1	2	0.25		{0.3, 0.5, 0.8, 1.0}	

Figure 5.23: A vague spatial fact set involving one numeric measure, one vague spatial measure and one vague spatial level table.

5.7.4 Loading a Vague Spatial Fact Table

A vague spatial fact table *VSFact* may be loaded according to two different methods outlined as follows. One method considers that the fact member of a vague spatial fact set is known in advance and is stored in *VSFact* before the remaining rows of the vague spatial fact set. The other method considers that the fact member of a vague spatial fact set is not known in advance, but can be calculated using as input all the other rows of the vague spatial fact set.

The first method assumes that the fact member of a vague spatial fact set is firstly stored in the table *VSFact*. Then, the remaining rows of the same vague spatial fact set are inserted in a single transaction. Before committing the transaction, a routine checks the constraints defined by Rule 5VS explained in Section 5.7.1 using the primary key. For instance, consider the vague spatial fact set shown in Figure 5.23. The table *PesticideApplicationVSFact* would be loaded by firstly inserting the first row. Thereafter, the second and third rows are inserted in a single transaction. Finally, assuming that the three rows have the same values for *PesticideFK*, *DateFK* and *CropFK*, partial values of the measures of the second and third rows are aggregated and compared to the total values of the measures of the first row. The aggregation uses adequate functions (i.e. SUM, *VSUnion* and Union). In addition, values provided for *CropElementNum*

must reflect the quantity of geometries in CropElementsGeo where CropFK=CropPK.

The second method assumes that rows associating partial values of measures are inserted in the table *VSFact* in a single transaction. Before committing the transaction, the fact member is calculated by a routine using the aforementioned rows as input. Considering the same primary key, the adequate aggregation functions are applied over the measure values of the input rows to obtain total measure values of the fact member. For instance, consider the vague spatial fact set shown in Figure 5.23. The table PesticideApplicationVSFact would be loaded by inserting the second and third rows in a single transaction. Before committing the transaction, the row of the fact member is created using the same values for PesticideFK, DateFK and CropFK. Besides, the measure values in the row of the fact member are calculated using partial values of the measures of the second and third rows and applying the adequate aggregation function (i.e. SUM, *VSUnion* and Union). In addition, values provided for CropElementNum must reflect the quantity of geometries in CropElementsGeo where CropFK=CropPK.

The first method checks the constraints based on an existing fact member. Conversely, the second method creates a fact member that already complies with the constraints. The use of each method is related to the knowledge, in advance, of the fact member.

5.7.5 Discussion

After explaining and exemplifying the logical design for the vague spatial fact, the five implications of the design of a vague spatial fact are discussed as follows. The first implication is the mixing of granularities in the vague spatial fact table. Each row assumes one of the following granularities: *by vague spatial member* or *by element of vague spatial member*. Queries must clearly specify the grain of the rows to be retrieved, otherwise false aggregation results can be produced. To refer to rows with the grain *by vague spatial member*, the conditional $ElementNum = 0$ must be included in the WHERE clause of the SQL query. Conversely, in order to refer to rows with the grain *by element of vague spatial member*, the conditional $ElementNum \neq 0$ must be included.

The second implication is the impossibility of using partial measure values, i.e. measure values where $ElementNum \neq 0$, for aggregation in a roll-up. Consider l_{parent} and l_{child} as being vague spatial members of the vague spatial levels L_{parent} and L_{child} , respectively, and that $L_{parent} \preceq L_{child}$. Let l_{parent} roll-up to l_{child} . Once a measure value in a vague spatial level table is associated to a certitude element or a dubiety element of l_{child} , it is impossible to state whether the measure value associated to an element of l_{child} must be assigned to a certitude element or to a dubiety element of l_{parent} , even though the hierarchy determine the admissible

topological relationships for $L_{parent} \preceq L_{child}$. For example, suppose that a dubiety element of a crop overlaps both a certitude element and a dubiety element of an agricultural land. Measure values are assigned to the cited dubiety element of a crop in the vague spatial fact table. However, a roll-up cannot aggregate measures values assigned to the cited crop's dubiety element and correctly associate it to the overlapped certitude element of the agricultural land or to the overlapped dubiety element of the agricultural land.

The third implication concerns an increase in the storage requirements due to the creation of the column *ElementNum*, its inclusion in the primary key, and the storage of several rows that compose vague spatial fact sets. The column *ElementNum* is essential to design the vague spatial fact and require a few additional bytes for each row. It also adds a few bytes on each entry of the index created for the primary key. Furthermore, the rows of vague spatial fact sets require several bytes if at least one vague spatial measure exist, due to the storage of two geometries and one array per row. It is essential to avoid storing rows where measure values are equivalent to zero or empty. Nevertheless, as a voluminous fact table is an intrinsic characteristic of a DW, an even more voluminous fact table is expected in a vague SDW due to the complexity of vague spatial data.

The fourth implication is a foreseen overhead to process queries due to the inclusion of conditionals to fetch rows according to different granularities. A bitmap index built on the column *ElementNum* may be a feasible alternative to reduce such overhead although it might also increase the storage requirements. One bit-vector for $ElementNum = 0$ to fetch rows *by vague spatial member* and one binned bit-vector for $ElementNum > 0$ to fetch rows *by element of vague spatial member* comply with the conditionals and might improve the performance to process queries.

The fifth implication concerns an overhead that can be caused by the verification of constraints when loading the vague spatial fact table. One method to load the vague spatial fact table checks most of the constraints after the insertion of rows that belong to a vague spatial fact set, by comparing aggregated partial values of measures to the values of measures of the fact member. Conversely, the other method creates a fact member ensuring that most of the constraints are satisfied. Although these different methods may cause distinct overheads for loading the vague spatial fact table, the decision on which method to use is not only based on performance. Indeed, the choice for the first method assumes the fact member is known in advance, while the choice for the second method considers that the fact member is unknown in advance and must be computed.

5.8 Vague Topological Constraints

In the VSCube conceptual model, vague topological constraints have been addressed mainly in hierarchies relating a pair of vague spatial attributes, or one vague spatial attribute and one crisp spatial attribute. In the VSMultiDim conceptual model, vague topological constraints have been specified in hierarchies and recognized in the spatial fact, which relates a vague spatial level to a crisp or vague spatial level through the fact. The sets of topological relationships $R_{(c,c)}$, $R_{(c,d)}$, $R_{(d,c)}$ and $R_{(d,d)}$ have guided the specification of vague topological constraints in the conceptual design of vague SDWs.

In the logical design of vague SDWs, vague topological constraints are implemented in the DBMS to ensure integrity of vague spatial data. The vague topological constraints provided in the conceptual models are reused, adapted and improved. The following sections focus on the implementation of vague topological constraints for vague SDWs. Section 5.8.1 details the pairwise evaluation of sets of topological relationships, Section 5.8.2 details vague topological constraints in hierarchies, Section 5.8.3 vague topological constraints in a spatial fact, and Section 5.8.4 describes intra-level and intra-fact vague topological constraints.

5.8.1 Pairwise Evaluation of Sets of Topological Relationships

The pairwise evaluation of sets of topological relationships processes both the multiple geometry and the array of membership values as being lists of identical fixed length. The elements of two different vague spatial objects are compared and the satisfiability of one or more topological relationship is assessed. As a result, the evaluation of the sets of topological relationships $R_{(c,c)}$, $R_{(c,d)}$, $R_{(d,c)}$ and $R_{(d,d)}$ becomes feasible.

Let a and b be vague spatial objects whose elements are sorted in ascending order of membership value. The length of the array of membership values of a is l_a , while the length of the array of membership values of b is l_b . The minimum offset of a certitude element of a is the minimum value o_a between 1 and l_a such that $a.ElementsMval[o_a] = 1$. The minimum offset of a certitude element of b is o_b and is analogous.

The pairwise evaluation of the sets of topological relationships $R_{(c,c)}$, $R_{(c,d)}$, $R_{(d,c)}$ and $R_{(d,d)}$ between a and b consists of the following assessments:

- $\forall r \in R_{(c,c)} : \text{is } r \text{ true for } a.ElementsGeo[i] \text{ and } b.ElementsGeo[j], \text{ where } o_a \leq i \leq l_a \text{ and } o_b \leq j \leq l_b?$
- $\forall r \in R_{(c,d)} : \text{is } r \text{ true for } a.ElementsGeo[i] \text{ and } b.ElementsGeo[j], \text{ where } o_a \leq i \leq l_a \text{ and}$

$$1 \leq j < l_b?$$

- $\forall r \in R_{(d,c)} : \text{is } r \text{ true for } a.\text{ElementsGeo}[i] \text{ and } b.\text{ElementsGeo}[j], \text{ where } 1 \leq i < l_a \text{ and } o_b \leq j \leq l_b?$
- $\forall r \in R_{(d,d)} : \text{is } r \text{ true for } a.\text{ElementsGeo}[i] \text{ and } b.\text{ElementsGeo}[j], \text{ where } 1 \leq i < l_a \text{ and } 1 \leq j < l_b?$

Clearly, the assessment of each set of topological relationships requires providing bounds of intervals that constrain the offsets for certitude elements and dubiety elements from both *a* and *b*. The implementation of the pairwise evaluation of sets of topological relationships has been performed using two UDFs: VS_Relate (by analogy with OGC's Relate (HERRING, 2011) and PostGIS' ST_Relate⁷) and VS_Constraints_Topological, which are explained as follows.

Listing 5.2 describes the function VS_Relate that receives as arguments two multiple geometries, bounds of two intervals, and one array of masks. It returns whether at least one topological relationship in a mask is true for a pair of geometries extracted from the input multiple geometries. A mask is a 9×1 representation of the 3×3 matrix that describe a valid topological relationship, as outlined in Sections 2.1.1.3 and 2.1.2.2. The *i*-th geometry in *a.ElementsGeo* is tested against the *j*-th geometry in *b.ElementsGeo* to assess the topological relationship specified by the *r*-th mask in *masks*, for $a_lbound \leq i \leq a_ubound$ and $b_lbound \leq j \leq b_ubound$. If a test yields true, then the counter of hits is incremented and it becomes irrelevant to verify the (*r*+1)-th mask, since one topological relationship holds. In the end, if the counter of hits is equal to the number of geometries evaluated from *a.ElementsGeo*, then the function returns true. Otherwise, it returns false.

Listing 5.2: A routine to evaluate topological relationships between several pairs of element geometries.

```
CREATE OR REPLACE FUNCTION VS_Relate(
  a.ElementsGeo geometry, a_lbound int, a_ubound int,
  b.ElementsGeo geometry, b_lbound int, b_ubound int,
  masks text array)
RETURNS boolean AS $$
DECLARE
  hits int;
BEGIN
  hits:=0;
  FOR i IN a_lbound .. a_ubound LOOP
    FOR j IN b_lbound .. b_ubound LOOP
      FOR r IN 1 .. array_length(masks,1) LOOP
        IF (ST_Relate(ST_GeometryN(a.ElementsGeo, i),
                     ST_GeometryN(b.ElementsGeo, j),
                     masks[r]))
        THEN
          hits := hits + 1;
        EXIT;
      END LOOP;
    END LOOP;
  END LOOP;
END;
```

⁷http://postgis.net/docs/ST_Relate.html

```

        END IF ;
    END LOOP;
END LOOP;
IF (hits = 1 + a_ubound - a_lbound) THEN
    RETURN TRUE;
END IF ;
RETURN FALSE;
END; $$
LANGUAGE 'plpgsql';

```

Another routine is essential to call `VS_Relate` and provide the bounds of the intervals, since there is one pair of intervals for each one of the sets of topological relationships $R_{(c,c)}$, $R_{(c,d)}$, $R_{(d,c)}$ and $R_{(d,d)}$. Listing 5.3 describes the function `VS_TopologicalConstraints`, which complies with these purposes. The input arguments of `VS_TopologicalConstraints` are the multiple geometries representing the elements of two vague spatial objects, as well as their arrays of membership values. In addition, each one of the aforementioned sets are denoted by one array of masks that are also provided as arguments. The function `VS_TopologicalConstraints` executes as follows.

First, it calculates the length of each array of membership values. Second, it calls the function `VS_MinOffsetCertitudeElement` for each array of membership values. Then, dubiety elements assume offsets between 1 and 1 minus the offset returned by `VS_MinOffsetCertitudeElement`, while certitude elements assume offsets between the value returned by `VS_MinOffsetCertitudeElement` and the array length. The function `VS_MinOffsetCertitudeElement` is detailed in Appendix A. Third, at most one call is made to the function `VS_Relate` for the set of topological relationships $R_{(c,c)}$, considering the bounds of certitude elements and bounds of dubiety elements from each vague spatial object. The function `VS_Relate` is not called if the set of topological relationships is empty or has a null array of masks. Fourth, fifth, and sixth concern $R_{(c,d)}$, $R_{(d,c)}$, and $R_{(d,d)}$, respectively. If one execution of `VS_Relate` yields false for a set of topological relationships, then a message is prompted specifying the corresponding set and the execution is terminated by returning false. Seventh, if all the sets of topological relationships $R_{(c,c)}$, $R_{(c,d)}$, $R_{(d,c)}$, and $R_{(d,d)}$ are satisfied, it returns true.

Listing 5.3: A routine to evaluate the sets of topological relationships for a pair of vague spatial objects.

```

CREATE OR REPLACE FUNCTION VS_TopologicalConstraints (
    a_ElementsGeo geometry, a_ElementsMval float array,
    b_ElementsGeo geometry, b_ElementsMval float array,
    rcc text array,
    rcd text array,
    rdc text array,
    rdd text array)
RETURNS boolean AS $$
DECLARE
a_length int;
b_length int;
a_offset int;
b_offset int;

```



```

BEGIN
--1. calculates lengths of arrays
a_length:= array_length(a_ElementsMval,1);
b_length:= array_length(b_ElementsMval,1);

--2. obtains the minimum offsets of a certitude element
a_offset:= VS_MinOffsetCertitudeElement(a_ElementsMval);
b_offset:= VS_MinOffsetCertitudeElement(b_ElementsMval);

--3. evaluates R(c,c) if both objects have certitude
IF (rcc IS NOT NULL) THEN
  IF NOT VS_Relate(a_ElementsGeo, a_offset, a_length,
                  b_ElementsGeo, b_offset, b_length, rcc) THEN
    RAISE NOTICE 'Invalid topological relationship between certitudes: Rcc.';
    RETURN FALSE;
  END IF;
END IF;

--4. evaluates R(c,d) if one object has certitude and the other has dubiety
IF (rcd IS NOT NULL) THEN
  IF NOT VS_Relate(a_ElementsGeo, a_offset, a_length,
                  b_ElementsGeo, 1, b_offset-1, rcd) THEN
    RAISE NOTICE 'Invalid topological relationship between certitude and
                  dubiety: Rcd.';
    RETURN FALSE;
  END IF;
END IF;

--5. evaluates R(d,c) if one object has dubiety and the other has certitude
IF (rdc IS NOT NULL) THEN
  IF NOT VS_Relate(a_ElementsGeo, 1, a_offset-1,
                  b_ElementsGeo, b_offset, b_length, rdc) THEN
    RAISE NOTICE 'Invalid topological relationship between dubiety and
                  certitude: Rdc.';
    RETURN FALSE;
  END IF;
END IF;

--6. evaluates R(d,d) if both objects have dubiety
IF (rdd IS NOT NULL) THEN
  IF NOT VS_Relate(a_ElementsGeo, 1, a_offset-1,
                  b_ElementsGeo, 1, b_offset-1, rdd) THEN
    RAISE NOTICE 'Invalid topological relationship between dubieties: Rdd.';
    RETURN FALSE;
  END IF;
END IF;

--7. the relationships have been satisfied, then return
RETURN TRUE;
END; $$
LANGUAGE 'plpgsql';

```

This section addressed the verification of topological constraints among elements of a pair of vague spatial objects, independently from the tables these objects belong to. In the following sections, specific cases are tackled regarding the tables these objects belong to.

5.8.2 Hierarchy

This section details the vague topological constraints on a hierarchy that was designed as explained in Section 5.5 and focuses on the relationship held by a pair of spatial levels, such that at least one is a vague spatial level. The specific case addressed in this section concerns the

insertion or the update in the child level. Then, the topological constraint verifies if the member being inserted or updated is correctly related to a member in the parent level. Furthermore, the parent and child levels have a relationship with cardinality 1:N.

The UDF `VS_Constraints_Topological_Hierarchy` described in Listing 5.4 complies with these purposes. Whenever a row is inserted or updated in the child level table, a trigger calls that UDF. Then, the sets of topological relationships $R_{(c,c)}$, $R_{(c,d)}$, $R_{(d,c)}$, and $R_{(d,d)}$ are assessed by comparing the vague spatial object being inserted or updated to the vague spatial object that is referenced by the row, which belongs to the parent level.

The array `TG_ARGV` is an array of arguments passed by the trigger to a function. When a trigger calls the function `VS_Constraints_Topological_Hierarchy`, the following arguments concern the child level table: the name of the geometry column is `TG_ARGV[0]`, the name of the column with the array of membership values is `TG_ARGV[1]`, and the name of the column that references the parent level is `TG_ARGV[2]`. Also, the following arguments concern the parent level table: the name of the table is `TG_ARGV[3]`, the name of the column with the primary key is `TG_ARGV[4]`, the name of the geometry column is `TG_ARGV[5]`, and the name of the column with the array of membership values is `TG_ARGV[6]`. Besides, the following arguments concern the sets of topological relationships: $R_{(c,c)}$ is in `TG_ARGV[7]`, $R_{(c,d)}$ is in `TG_ARGV[8]`, $R_{(d,c)}$ is in `TG_ARGV[9]`, and $R_{(d,d)}$ is in `TG_ARGV[10]`.

Listing 5.4: A routine to ensure the satisfaction of vague topological constraints on a hierarchy.

```
CREATE FUNCTION VS_Constraints_Topological_Hierarchy ()
RETURNS TRIGGER AS $$
DECLARE
gChild geometry; mvChild float array; gParent geometry; mvParent float array;
fkParent bigint;
rcc text array; rcd text array; rdc text array; rdd text array;
tableParent text; pkParentCol text; gParentCol text; mvParentCol text;
target record;

BEGIN
--1. retrieves geometries and arrays of membership values from the child level table
EXECUTE 'SELECT ($1).' || TG_ARGV[0] || ', ($1).' || TG_ARGV[1] || ', ($1).' || TG_ARGV[2]
USING NEW INTO gChild, mvChild, fkParent;

--2. copies the name of the parent table and the names of its columns
tableParent:= TG_ARGV[3];
pkParentCol:= TG_ARGV[4];
gParentCol:= TG_ARGV[5];
mvParentCol:= TG_ARGV[6];

--3. retrieves the vague spatial object referenced in the parent level table
IF (fkParent IS NOT NULL) THEN
EXECUTE 'SELECT ' || gParentCol || ', ' || mvParentCol || ' FROM ' || tableParent
|| ' WHERE ' || pkParentCol || '=' || fkParent
INTO STRICT gParent, mvParent;
ELSE
RAISE NOTICE 'There is not an associated vague spatial object in the parent level.';
RETURN NEW;
END IF;

--4. issues a warning if the foreign key has been violated and return
```

```

IF (gParent IS NULL) THEN
RAISE NOTICE 'There is not a vague spatial object in % where % = %.',
tableParent, pkParentCol, fkparent;
RETURN NEW;
END IF;

--5. transforms comma-separated masks into array of masks
rcc:= string_to_array(TG.ARGV[7],',');
rcd:= string_to_array(TG.ARGV[8],',');
rdc:= string_to_array(TG.ARGV[9],',');
rdd:= string_to_array(TG.ARGV[10],',');

--6. evaluates the sets of topological relationships
EXECUTE 'SELECT VS_TopologicalConstraints($1,$2,$3,$4,$5,$6,$7,$8) as result'
INTO STRICT target
USING gParent, mvParent, gChild, mvChild, rcc, rcd, rdc, rdd;

--7. verifies whether a violation occurred
IF (NOT target.result) THEN
RAISE EXCEPTION 'Violation of vague topological constraints.';
END IF;

--8. proceeds with the insert or update
RETURN NEW;
END; $$
LANGUAGE 'plpgsql';

```

First, the geometry, the array of membership values and the parent key value are retrieved from the child level table, according to the names of columns provided by the arguments. Second, the names of the parent table and of its columns are copied to local variables. Third, a query retrieves the referenced vague spatial object in the parent level. Note that the key value provided for searching the parent level table can be null, which means that there is no correspondence with a vague spatial object in the parent level and, consequently, it is unnecessary to evaluate topological relationships. In such case, a warning message is issued and the execution is finished with success. On the other hand, if the value is not null and the query retrieved an empty result set, then it indicates a violation of foreign key that must be handled by the DBMS. Then, fourth, a warning message is issued and the control is given back to the DBMS.

However, if a vague spatial object is retrieved from the parent level table, the execution `VS_Constraints_Topological_Hierarchy` continues. Fifth, the masks of topological relationships are copied to local variables. Sixth, consecutive calls to the function `VS_TopologicalConstraints` are made to test the pair of vague spatial objects against the sets of topological relationships represented by their masks. Seventh, if a violation occurred, an exception is raised. Eighth, if the evaluation of the topological constraints was successful, the row can be inserted or updated in the child level table.

The function `VS_Constraints_Topological_Hierarchy` allows a vague spatial object to be defined in a child level without being related to any vague spatial object in the parent level. If this is not the case for a given application, adaptations should be performed. The creation of one application-dependent trigger for each pair of vague spatial attributes in a hierarchy is neces-

sary to benefit from the vague topological constraint on a hierarchy. A trigger is described in Example 5.8.1.

Example 5.8.1. Consider the vague SDW designed for the pest control case study whose logical schema is shown in Figure 5.21. A hierarchy which relates vague regions of crops and vague regions of agricultural lands. Then, the following trigger executes the function `VS_Constraints_Topological_Hierarchy` for each row inserted into or updated in the table `Crop`.

```
CREATE TRIGGER TopologicalConstraints_AgriLand_Crop
BEFORE INSERT OR UPDATE ON Crop FOR EACH ROW EXECUTE PROCEDURE
VS_Constraints_Topological_Hierarchy(
  'Crop_ElementsGeo', 'Crop_ElementsMval', 'AgriLandFK',
  'AgriLand', 'AgriLandPK', 'AgriLand_ElementsGeo', 'AgriLand_ElementsMval',
  —R(c,c) comma-separated masks for Contains and Covers:
  'T*****FF*,*T*****FF*,***T**FF*,*****T*FF*',
  —R(c,d) comma-separated masks for Contains:
  'T*****FF*',
  —R(d,c) comma-separated masks for for Meets and Disjoint:
  'FT***** ,F**T***** ,F***T**** ,FF*FF*****',
  —R(d,d) comma-separated masks for for Meets and Disjoint:
  'FT***** ,F**T***** ,F***T**** ,FF*FF*****'
)
```

The task of the vague SDW designer consists of specifying only the trigger according to the application, since the function `VS_Constraints_Topological_Hierarchy` and the functions described in Section 5.8.1 do not depend on the application. Another important remark is that, for each pair of vague spatial attributes in distinct levels of a hierarchy, one trigger should be specified.

5.8.3 Spatial Fact

In Section 4.9.4, the concept of spatial fact has been described. This section details the logical design of vague topological constraint imposed by a spatial fact, which occurs when a fact table relates at least two vague spatial level tables, or one crisp spatial level table and one vague spatial level table. Not only a fact table designed according to Rule 4VS is supported, but also a vague spatial fact table designed according to Rule 5VS.

An UDF has been defined and is called by a trigger whenever a row is inserted in the fact table. Updates in the fact table are not addressed. The insertion provides key values referencing the level tables. These are used to retrieve the referenced vague spatial members for assessing the sets of topological relationships $R_{(c,c)}$, $R_{(c,d)}$, $R_{(d,c)}$ and $R_{(d,d)}$. Function `VS_Constraints_SpatialFact` described in Listing 5.5 complies with these purposes.

The function `VS_Constraints_SpatialFact` receives the following arguments passed by a trigger. Concerning one of the referenced level tables, the arguments are: the name of the column

in the fact table referencing a level table in TG_ARGV[0], the name of the referenced table in TG_ARGV[2], the name of the column with a primary key in TG_ARGV[3], the name of the geometry column in TG_ARGV[4], and the name of the column with the array of membership value in TG_ARGV[5]. Arguments are analogously provided for the other referenced level table, in TG_ARGV[1], TG_ARGV[6], TG_ARGV[7], TG_ARGV[8], and TG_ARGV[9], respectively. Besides, the following arguments concern the sets of topological relationships: $R_{(c,c)}$ is in TG_ARGV[10], $R_{(c,d)}$ is in TG_ARGV[11], $R_{(d,c)}$ is in TG_ARGV[12], and $R_{(d,d)}$ is in TG_ARGV[13].

First, the UDF retrieves the key values of the referenced tuples, using the names of the columns that reference level tables. Second, if one or both of these key values is null, it means that the row being inserted does not relate vague spatial objects. Thus, a warning message is issued and the control is given back to the DBMS, since it is unnecessary to check topological constraints. If both key values are not null, the execution continues. Third, the names of referenced tables and their columns are copied into local variables. Fourth, these names are used in a query that fetches one referenced vague spatial object by using the key values previously obtained. Fifth, the other vague spatial object referenced is fetched analogously. Sixth, if the geometries are null, it also means that the row being inserted does not relate vague spatial objects, and the control is given back to the DBMS. Otherwise, the execution continues. Seventh, the masks of topological relationships are copied to local variables. Eighth, the function VS_TopologicalConstraints is called to test the retrieved vague spatial objects against the sets of topological relationships represented by their masks. Ninth, if the evaluation of the topological constraints was not successful, then an exception is raised. Otherwise, tenth, the row can be inserted or updated in the fact table.

Listing 5.5: A routine to ensure the satisfaction of vague topological constraints on a spatial fact.

```
CREATE FUNCTION VS_Constraints_SpatialFact() RETURNS TRIGGER AS $$
DECLARE
  g1 geometry; mv1 float array;
  g2 geometry; mv2 float array;
  fk1 bigint; fk2 bigint;
  rcc text array; rcd text array; rdc text array; rdd text array;
  table1 text; table2 text; pk1Col text; pk2Col text;
  g1Col text; mv1Col text; g2Col text; mv2Col text;
  target record;

BEGIN
--1. retrieves the key values of the referenced vague spatial objects
EXECUTE 'SELECT ($1).' || TG_ARGV[0] || ', ($1).' || TG_ARGV[1]
USING NEW INTO fk1, fk2;

--2. issues a warning message if at least one vague spatial object is not referenced
IF (fk1 IS NULL) or (fk2 IS NULL) THEN
RAISE NOTICE 'The inserted row does not reference level tables.';
RETURN NEW;
END IF;

--3. copies the name of the parent table and the names of its columns
table1:= TG_ARGV[2]; table2:= TG_ARGV[6];
```

```

pk1Col:= TG.ARGV[3];   pk2Col:= TG.ARGV[7];
g1Col:=  TG.ARGV[4];   g2Col:=  TG.ARGV[8];
mv1Col:= TG.ARGV[5];   mv2Col:= TG.ARGV[9];

--4. retrieves one of the referenced vague spatial objects
EXECUTE 'SELECT ' || g1Col || ', ' || mv1Col || ' FROM ' || table1
        || ' WHERE ' || pk1Col || '=' || fk1
INTO STRICT g1, mv1;

--5. retrieves the other referenced vague spatial object
EXECUTE 'SELECT ' || g2Col || ', ' || mv2Col || ' FROM ' || table2
        || ' WHERE ' || pk2Col || '=' || fk2
INTO STRICT g2, mv2;

--6. issues a warning message if at least one vague spatial object is not referenced
IF (g1 IS NULL) OR (g2 IS NULL) THEN
    RAISE NOTICE '';
RETURN NEW;
END IF;

--7. transforms comma-separated masks into array of masks
rcc:= string_to_array(TG.ARGV[10],',');
rcd:= string_to_array(TG.ARGV[11],',');
rdc:= string_to_array(TG.ARGV[12],',');
rdd:= string_to_array(TG.ARGV[13],',');

--8. evaluates the sets of topological relationships
EXECUTE 'SELECT VS_TopologicalConstraints($1,$2,$3,$4,$5,$6,$7,$8) as result '
INTO STRICT target
USING g1, mv1, g2, mv2, rcc, rcd, rdc, rdd;

--9. checks whether there was a violation
IF (NOT target.result) THEN
    RAISE EXCEPTION 'Violation of vague topological constraints.';
END IF;

10. proceeds with the insertion
RETURN NEW;
END; $$
LANGUAGE 'plpgsql';

```

Example 5.8.2. Consider the vague SDW designed for the HLB case study whose logical schema is shown in Figure 5.19 and Example 4.9.9 that describes the topological constraint on the spatial fact involving trees and infected groups. The following trigger executes the function detailed in Listing 5.5 for each row inserted into the table HLBControl. Note that trees do not have dubieties. To denote that a tree is composed of a single certitude element whose degree of membership is 1.0, the argument “ARRAY[1.0]” has been passed, while the masks for testing the dubiety of trees are null.

```

CREATE TRIGGER TopologicalConstraints_Tree_InfectedGroup
BEFORE INSERT ON HLBControl FOR EACH ROW EXECUTE PROCEDURE
VS_Constraints_SpatialFact(
    'TreeFK', 'GroupFK', 'Tree', 'Treeld', 'TreeGeo', 'ARRAY[1.0]',
    'InfectedGroup', 'GroupId', 'InfectedGroup_ElementsGeo',
    'InfectedGroup_ElementsMval',
    --R(c,c) comma-separated masks for intersects:
    'T*****', '*T*****', '**T*****', '***T*****',
    --R(c,d) comma-separated masks for intersects:
    'T*****', '*T*****', '**T*****', '***T*****',
    --R(d,c) is null since a tree is crisp:
    null,

```

```

—R(d,d) is null since a tree is crisp:
null
);

```

The task of the vague SDW designer consists of specifying only the trigger according to the application, since the function `VS_Constraints_SpatialFact` and the functions described in Section 5.8.1 do not depend on the application. Another important remark is that, for each pair of vague spatial attributes related by a spatial fact, one trigger should be specified.

5.8.4 Intra Level and Intra Fact

This section details vague topological constraints related to a pair of vague spatial attributes defined in the same vague spatial level table, i.e. intra level vague topological constraints. It also addresses vague topological constraints related to a pair of vague spatial measures defined in the same fact table, i.e. intra fact vague topological constraints. It assumes that vague spatial attributes and vague spatial measures have been designed as explained in Section 5.3.

Let T be a vague spatial level table or a fact table and contain the vague spatial attributes A_1 and A_2 designed according to Rule 1VS. Let also *new* be the row which is being inserted or updated in T . The insertion or update must not violate the topological constraint imposed by the sets of topological relationships $R_{(c,c)}$, $R_{(c,d)}$, $R_{(d,c)}$ and $R_{(d,d)}$, which are held between A_1 and A_2 . Therefore, a routine must ensure that every row in T satisfies the vague topological constraints.

The function `VS.TopologicalConstraintsInTable` shown in Listing 5.6 receives 8 arguments in the array `TG_ARGV`. The arguments are provided by the trigger that calls the function. `TG_ARGV[0]` is the name of the geometry column of A_1 , `TG_ARGV[1]` is the name of the column with membership values of A_1 , `TG_ARGV[2]` is the name of the geometry column of A_2 , `TG_ARGV[3]` is the name of the column with membership values of A_2 , `TG_ARGV[4]`, `TG_ARGV[5]`, `TG_ARGV[6]`, and `TG_ARGV[7]` are strings with comma-separated masks representing the sets $R_{(c,c)}$, $R_{(c,d)}$, $R_{(d,c)}$, and $R_{(d,d)}$, respectively. The function allows the modification in table T if the sets of topological relationships $R_{(c,c)}$, $R_{(c,d)}$, $R_{(d,c)}$ and $R_{(d,d)}$ are satisfied by the row being modified, which means that the geometries and arrays of membership values provided for the attributes A_1 and A_2 comply with the cited sets of topological relationships. The execution of the function `VS_Constraints_Topological_IntraLevel_IntraFact` is as follows.

First, it copies the geometries and arrays of membership values into local variables. The record `NEW`, which refers to the row being modified, maintains these geometries and arrays into data fields whose names are those passed as the first to fourth arguments. Second, it transforms

each comma-separated mask into an array of masks and assigns to a specific local variable. Third, it calls the function `VS_TopologicalConstraints` using the aforementioned local variables as arguments. The result is assigned to the local variable `target`. Fourth, if the result in `target` yields that topological constraints were not satisfied, then an exception is thrown and the modification of the row is canceled. Fifth, it concludes that the topological constraint was satisfied and the modification is allowed.

Listing 5.6: A routine to ensure the satisfaction of intra level and intra fact vague topological constraints.

```
CREATE FUNCTION VS_Constraints_Topological_IntraLevel_IntraFact ()
RETURNS TRIGGER AS $$
DECLARE
  g1 geometry; mv1 float array;
  g2 geometry; mv2 float array;
  rcc text array; rcd text array; rdc text array; rdd text array;
  target record;
BEGIN

--1. retrieves geometries and arrays of membership values
EXECUTE 'SELECT ($1).' || TG.ARGV[0] || ', ($1).' || TG.ARGV[1]
      || ', ($1).' || TG.ARGV[2] || ', ($1).' || TG.ARGV[3]
USING NEW INTO g1, mv1, g2, mv2;

--2. transforms comma-separated masks into array of masks
rcc:= string_to_array(TG.ARGV[4],',');
rcd:= string_to_array(TG.ARGV[5],',');
rdc:= string_to_array(TG.ARGV[6],',');
rdd:= string_to_array(TG.ARGV[7],',');

--3. evaluates the sets of topological relationships
EXECUTE 'SELECT VS_TopologicalConstraints($1,$2,$3,$4,$5,$6,$7,$8) as result '
INTO STRICT target
USING g1, mv1, g2, mv2, rcc, rcd, rdc, rdd;

--4. notifies whether the topological constraint was violated
IF (NOT target.result) THEN
  RAISE EXCEPTION 'Violation of the vague topological constraint.';
END IF;

--5. allows the insert or update
RETURN NEW;
END; $$
LANGUAGE 'plpgsql';
```

The creation of one application-dependent trigger per pair of vague spatial attributes in a table is necessary to benefit from the intra level or intra fact vague topological constraint. It is noteworthy that the task of the vague SDW designer consists of specifying only the trigger according to the application, since the function `VS_Constraints_Topological_IntraLevel_IntraFact` and the functions described in Section 5.8.1 do not depend on the application.

5.9 Vague Spatial Online Analytical Processing

Considering relational schemata of vague SDWs produced by the logical design, vague SOLAP provides multidimensional queries extended with vague spatial predicates and vague spatial data aggregation. Accessors, vague spatial aggregation functions and vague spatial predicates introduced by the VSCube conceptual Model in Chapter 4 are, therefore, reused, extended, improved and then implemented in the DBMS to allow their use in SQL queries. They constitute different classes of operations, as listed in Table 5.2. In addition, slice-and-dice, drill-down and roll-up may require one or more SQL queries using these operations. The following sections describe the implemented operations listed in Table 5.2 and exemplify their use in SQL queries issued over vague SDWs. Section 5.9.1 addresses accessors, Section 5.9.2 tackles vague spatial predicates, Section 5.9.3 stands for vague spatial aggregation, and Section 5.9.4 addresses slice-and-dice, drill-down and roll-up.

Table 5.2: Operations on vague spatial attributes.

Class	Operations
Accessors	Elements, ElementsExprMval, ElementsGeoExprMval, CertitudeGeo, DubietyGeo, ElementsOffsetsExprMval
Vague Spatial Predicates	IntersectsCertitude, IntersectsDubiety, IntersectsDubietyExprMval, IntersectingCertitudeElements, IntersectingDubietyElements, IntersectingElementsExprMval, WithinInnerUIntersectsOuterAndNotWithinInner
Vague Spatial Aggregation Functions	Union, Intersection, Difference

5.9.1 Accessors

Accessors provide access to certitude elements and dubiety elements of a given vague spatial object. They have been implemented as UDFs using the procedural language PL/pgSQL from PostgreSQL. Their method signatures are listed in Table 5.3, considering that certitude elements and dubiety elements have their geometries in ElementsGeo and their membership values in the array ElementsMval. Examples 5.9.1 to 5.9.3 demonstrate the use of accessors to query vague SDWs. Each query is briefly commented regarding the utilized accessor.

Example 5.9.1. VS_ElementsExprMval.

Consider the HLB case study and the vague SDW logical schema shown in Figure 5.19. The following SQL query implements the query HLB7 listed in Table 1.2.

```
SELECT VS_ElementsExprMval ( InfectedRegion_ElementsGeo ,
```

Table 5.3: Accessors: methods' signatures and descriptions.

Method's signature and description
record VS.Elements(geometry ElementsGeo, float array ElementsMval) Returns a record with all elements of the vague spatial object.
record VS.ElementsExprMval(geometry ElementsGeo, float array ElementsMval, varchar ElementsMvalColumnName, text expression) Returns a record with elements of the vague spatial object that comply with the expression.
geometry VS.ElementsGeoExprMval(geometry ElementsGeo, float array ElementsMval, text expression) Returns a multiple geometry containing geometries of elements whose membership values comply with the expression.
geometry VS.CertitudeGeo(geometry ElementsGeo, float array ElementsMval) Returns a geometry obtained by merging the geometries of all certitude elements of the vague spatial object.
geometry VS.DubietyGeo(geometry ElementsGeo, float array ElementsMval) Returns a geometry obtained by merging the geometries of all dubiety elements of the vague spatial object.
setof bigint VS.ElementsOffsetsExprMval(float array ElementsMval) Returns the offsets of membership values that comply with the expression.

```

                                InfectedRegion_ElementsMval ,
                                'InfectedRegion_ElementsMval ' ,
                                'InfectedRegion_ElementsMval > 0.80 ')
FROM      AS GeoMval
FROM      InfectedRegion R, InfectedGroup G, HLBCControl H,
          Month M, Tree T, Plot P, Farm F, City C
WHERE     G.RegionFK = R.RegionId AND
          H.GroupFK = G.GroupId AND
          H.MonthFK = M.MonthNo AND
          H.TreeFK = T.TreeId AND
          T.PlotFK = P.PlotId AND
          P.FarmFK = F.FarmId AND
          F.CityFK = C.CityId AND
          C.Name = 'Sao Carlos' AND
          (M.MonthNo = 201401 OR
           M.MonthNo = 201402 OR
           M.MonthNo = 201403)
GROUP BY R.RegionId

```

Given an infected region that satisfies the conditions of the WHERE clause, every element whose membership value is greater than 0.80 is fetched and returned in a record. The GROUP BY clause avoids repeated rows in the result set, since an infected region is referenced by several rows of the fact table HLBCControl. In order retrieve the multiple geometry and the array of membership values in separate columns rather than a record into a single column, the previous query can be

surrounded by:

```
SELECT (GeoMval).ElementsGeo , (GeoMval).ElementsMval
FROM(
  —the previous query
) AS HLB7
```

Example 5.9.2. VS_CertitudeGeo and VS_DubietyGeo.

Consider the HLB case study and the vague SDW logical schema shown in Figure 5.19. The following SQL query implements the query HLB6 listed in Table 1.2.

```
SELECT P.PlotId , M.MonthNo, COUNT(*) AS NoInfectedTrees
FROM Month M, HLBControl H, Tree T, Plot P, Farm F, City C,
CityElevation E
WHERE H.MonthFK = M.MonthNo AND
H.TreeFK = T.TreeId AND
T.PlotFK = P.PlotId AND
P.FarmFK = F.FarmId AND
F.CityFK = C.CityId AND
E.CityId = C.CityId AND
C.Name = 'Sao Carlos' AND
(M.QuarterNo = 20141 OR
M.QuarterNo = 20142)
Val (AtGeometry (E.Elevation ,
VS_CertitudeGeo (H. InfectedTree_ElementsGeo ,
H. InfectedTree_ElementsMval)
)
) BETWEEN 500 AND 600
GROUP BY P.PlotId , M.MonthNo
```

The accessor VS_CertitudeGeo retrieves, from the vague spatial measure, only that points where certainly infected trees are located. AtGeometry and Val are operations described in for manipulating fields (VAISMAN; ZIMÁNYI, 2014b). AtGeometry returns the value of a continuous field (first argument) at a geometry (second argument). Val returns the value of the elevation field at a particular geometry, which is a point where a certainly infected tree is located. Supposing that the query required only points where possibly infected trees are located, replacing the certitude accessor by VS_DubietyGeo would suffice.

Example 5.9.2 highlights a query involving vague spatial data and continuous fields. This is an intrinsic feature enabled by the logical design of vague SDWs described in this thesis. It became feasible because the VSMultiDim model inherits and extends the MultiDim model (VAISMAN; ZIMÁNYI, 2014b) that supports continuous fields.

Example 5.9.3. VS_ElementsOffsetsExprMval and vague spatial fact.

Consider the vague SDW regarding pest control whose schema is depicted in Figure 5.21 with a vague spatial fact table. The following SQL query implements the query PC4 listed in Table 1.1.

```
SELECT SUM(AppliedTons) AS Loss , CropPK, Year
```

```

FROM PesticideApplicationVSFact , Crop , Date , Pesticide
WHERE CropFK = CropPK
      AND DateFK = DatePK
      AND PesticideFK = PesticidePK
      AND Type = 'Herbicide '
      AND CropElementNum <> 0
      AND CropElementsNum IN (ElementsOffsetsExprMval(
                              CropElementsMval , 'CropElementsMval<>1.0') )
GROUP BY CropPK , Year

```

The vague spatial fact table is required for the aggregation of values of applied tons of pesticides that refer to dubiety elements of crops. The conditional `CropElementNum <> 0` determine that only measure values assigned to certitude or dubiety elements of crops must be fetched. Besides, certitude elements of crops must not be considered and are ignored due to the expression provided as argument for the function `VS.ElementsOffsetsExprMval`.

5.9.2 Vague Spatial Predicates

Vague spatial predicates select vague spatial objects or their elements according to a set of criteria. They have been based on vague spatial predicates of the VSCube conceptual model (Section 4.5) and implemented as UDFs using the procedural language PL/pgSQL from PostgreSQL. Table 5.4 lists method's signatures of vague spatial predicates involving the topological relationship *intersects*. Analogous method's signatures for vague spatial predicates involving other topological relationships can be obtained, such as for containment range queries. Examples 5.9.5 to 5.9.7 demonstrate the use of vague spatial predicates issued against vague SDWs. Each exemplified query is briefly commented regarding the utilized vague spatial predicate.

Example 5.9.4. *IRQ_{object}*.

Consider the HLB case study and the vague SDW logical schema shown in Figure 5.19. The following SQL query corresponds to query HLB5 listed in Table 1.2, such that *w* is the rectangular spatial query window provided by the user.

```

SELECT COUNT(*) , M.YearNo , I.Team
FROM Tree T , HLBControl H , Month M , Inspector I ,
      InfectedGroup G , InfectedRegion R
WHERE H.TreeFK = T.TreeId AND
      H.MonthFK = M.MonthNo AND
      H.InspectorFK = I.InspectorId AND
      H.GroupFK = G.GroupId AND
      G.RegionFK = R.RegionId AND
      T.EradicationDate IS NOT NULL AND
      ST_Intersects ( InfectedRegion_MergedElementsGeo , w )
GROUP BY M.YearNo , I.Team

```

Table 5.4: Vague spatial predicates: methods' signatures and descriptions.

Method's signature and description
boolean VS.IntersectsCertitude(geometry ElementsGeo, float array ElementsMval, geometry w) Returns TRUE if the certitude of the vague spatial object intersects w. Otherwise, returns FALSE.
boolean VS.IntersectsDubiety(geometry ElementsGeo, float array ElementsMval, geometry w) Returns TRUE if the dubiety of the vague spatial object intersects w. Otherwise, returns FALSE.
boolean VS.IntersectsElementsExprMval(geometry ElementsGeo, float array ElementsMval, geometry w, text expression) Returns TRUE if elements of the vague spatial object that intersect w also comply with the expression.
geometry VS.IntersectingCertitudeElements(geometry ElementsGeo, float array ElementsMval, geometry w) Returns a multiple geometry containing certitude elements' geometries that intersect w. If there are not certitude elements intersecting w, then returns an empty geometry.
geometry VS.IntersectingDubietyElements(geometry ElementsGeo, float array ElementsMval, geometry w) Returns a multiple geometry containing dubiety elements' geometries that intersect w. If there are not dubiety elements intersecting w, then returns an empty geometry.
geometry VS.IntersectingElementsExprMval(geometry ElementsGeo, float array ElementsMval, geometry w, text expression) Returns a multiple geometry containing elements' geometries that both intersect w and comply with the expression. If there are not elements satisfying the conditions, returns an empty geometry.
setof record VS.WithinInnerUIntersectsOuterAndNotWithinInner(text Query, text Column, geometry Inner, geometry Outer) Returns the result set of a query that is created using Query as template and that assesses a vague spatial range query on Column using the spatial query windows Inner and Outer.

OGC's function Intersects implemented by PostGIS' function ST_Intersects is reused to demonstrate that developing an UDF is not necessary for the vague spatial predicate IRQ_{object} . Also, instead of testing a multipolygon in InfectedRegion.ElementsGeo, the query prioritizes querying the merged geometry in InfectedRegion.MergedElementsGeo that is a polygon.

Example 5.9.5. $IRQ_{certitude}$ and VS.IntersectsCertitude.

Consider the HLB case study, the vague SDW logical schema shown in Figure 5.19, and the query HLB5 listed in Table 1.2 such that w is the rectangular spatial query window provided by the user. Suppose that only infected regions that certainly intersect w are required. Then, only infected regions whose certitude intersect w must be considered:

```
SELECT COUNT(*), M.YearNo, I.Team
FROM Tree T, HLBControl H, Month M, Inspector I,
```

```

WHERE      InfectedGroup G, InfectedRegion R
           H.TreeFK = T.TreeId AND
           H.MonthFK = M.MonthNo AND
           H.InspectorFK = I.InspectorId AND
           H.GroupFK = G.GroupId AND
           G.RegionFK = R.RegionId AND
           T.EradicationDate IS NOT NULL AND
           VS_IntersectsCertitude ( InfectedRegion_ElementsGeo ,
                                   InfectedRegion_ElementsMval , w)
GROUP BY M.YearNo , I.Team

```

Example 5.9.6. *IRQ_{dubiety}* and *VS_IntersectsDubiety*.

Consider the HLB case study, the vague SDW logical schema shown in Figure 5.19, and the query HLB5 listed in Table 1.2 such that w is the rectangular spatial query window provided by the user. Suppose that only infected regions that possibly intersect w are required. Then, only infected regions whose dubiety intersect w must be considered:

```

SELECT     COUNT(*) , M.YearNo , I.Team
FROM       Tree T, HLBControl H, Month M, Inspector I ,
           InfectedGroup G, InfectedRegion R
WHERE      H.TreeFK = T.TreeId AND
           H.MonthFK = M.MonthNo AND
           H.InspectorFK = I.InspectorId AND
           H.GroupFK = G.GroupId AND
           G.RegionFK = R.RegionId AND
           T.EradicationDate IS NOT NULL AND
           VS_IntersectsDubiety ( InfectedRegion_ElementsGeo ,
                                   InfectedRegion_ElementsMval , w)
GROUP BY M.YearNo , I.Team

```

Example 5.9.7. *VSRQ_{object}* and *WithinInnerUIntersectsOuterAndNotWithinInner*.

Consider the HLB case study, the vague SDW logical schema shown in Figure 5.19, and the query HLB5 listed in Table 1.2. Let *inner* and *outer* be concentric rectangular spatial query windows provided by the user, rather than a single spatial query window. Also, suppose that infected regions within *inner* and infected regions that intersect *outer* are required, such that results concerning *inner* are more relevant than results regarding *outer*. The following SQL query performs a *VSRQ_{object}*.

```

SELECT * FROM VS_WithinInnerUIntersectsOuterAndNotWithinInner (
  'SELECT COUNT(*) , M.YearNo , I.Team
  FROM       Tree T, HLBControl H, Month M, Inspector I ,
             InfectedGroup G, InfectedRegion R
  WHERE      H.TreeFK = T.TreeId AND
             H.MonthFK = M.MonthNo AND
             H.InspectorFK = I.InspectorId AND
             H.GroupFK = G.GroupId AND
             G.RegionFK = R.RegionId AND
             T.EradicationDate IS NOT NULL AND
             VSRQobject

```

```

GROUP BY M.YearNo, I.Team
ORDER BY M.YearNo, I.Team',
'InfectedRegion_MergedElementsGeo',
inner,
outer
)

```

In detail, four arguments are provided to call `VS_WithinInnerUIntersectsOuterAndNotWithinInner` in Example 5.9.7. The query supplied as the first argument is used to create two sub-queries. These sub-queries assess the column provided as the second argument against the spatial query windows provided as the third and fourth arguments. The first sub-query adds an additional column with the constant “More relevant” to the SELECT clause. The string “VSRQobject” in the WHERE clause is replaced by `ST_Within(InfectedRegion_MergedElementsGeo, inner)`, which is a *CRQ_{object}* using *inner*. The second sub-query adds an additional column with the constant “Less relevant” to the SELECT clause. The string “VSRQobject” in the WHERE clause is replaced by the conjunction: `ST_Intersects(InfectedRegion_MergedElementsGeo, outer) AND NOT ST_Within(InfectedRegion_MergedElementsGeo, inner)`.

The conjunction determines both an *IRQ_{object}* using *outer* and a *CRQ_{object}* using *inner*. Note that the differences between the sub-queries are the columns added to the SELECT clause and the topological relationships they assess in the WHERE clause. Finally, a single query is written by placing the UNION operator between the sub-queries. The execution of the query produces a single result set comprising the result sets of both sub-queries.

Example 5.9.8. `VS.IntersectsElementsExprMval`.

Consider the HLB case study, the vague SDW logical schema shown in Figure 5.19, and the query HLB5 listed in Table 1.2 such that w is the rectangular spatial query window provided by the user. Suppose that infected regions must be retrieved if, and only if their elements that intersect w have possibility of infection greater than 75%. As a result, the query ignores vague regions whose elements intersect w but have lower possibility of infection.

```

SELECT COUNT(*), P.PlotId, M.YearNo, I.Team
FROM Plot P, Tree T, HLBControl H, Month M, Inspector I,
InfectedGroup G, InfectedRegion R
WHERE T.PlotFK = P.PlotId AND
H.TreeFK = T.TreeId AND
H.MonthFK = M.MonthNo AND
H.InspectorFK = I.InspectorId AND
H.GroupFK = G.GroupId AND
G.RegionFK = R.RegionId AND
T.EradicationDate IS NOT NULL AND
VS_IntersectsElementsExprMval(InfectedRegion_ElementsGeo,

```

```

                                InfectedRegion_ElementsMval ,
                                'InfectedRegion_ElementsMval > 0.75', w)
GROUP BY P.PlotId , M.YearNo , I.Team;

```

Example 5.9.8 illustrates that the operator `VS_IntersectsElementsExprMval` extends the vague spatial predicate $IRQ_{dubiety-mval}$. Different from the predicate, the operator is not constrained to dubiety elements. Also, the operator takes into account only the elements that do satisfy *intersects* to provide the result. As already mentioned in Section 4.5, the VSCube model's vague spatial predicates can be reused and extended to provide new predicates, such as the one implemented by the operator `VS_IntersectsElementsExprMval`.

Example 5.9.9. $IRQ_{certitude-elements}$ and `VS.IntersectingCertitude`.

Consider the HLB case study, the vague SDW logical schema shown in Figure 5.19, and the query HLB7 listed in Table 1.2. Suppose that only parts of an infected region that have possibility of infection 1 must be retrieved. In addition, these parts must intersect a spatial query window w , which is a rectangle of interest drawn by the user. Then:

```

SELECT    VS_IntersectingCertitude ( InfectedRegion_ElementsGeo ,
                                InfectedRegion_ElementsMval , w)
FROM      InfectedRegion R, InfectedGroup G, HLBControl H,
          Month M, Tree T, Plot P, Farm F, City C
WHERE     G.RegionFK = R.RegionId AND
          H.GroupFK = G.GroupId AND
          H.MonthFK = M.MonthNo AND
          H.TreeFK = T.TreeId AND
          T.PlotFK = P.PlotId AND
          P.FarmFK = F.FarmId AND
          F.CityFK = C.CityId AND
          C.Name = 'Sao Carlos' AND
          (M.MonthNo = 201401 OR
           M.MonthNo = 201402 OR
           M.MonthNo = 201403)
GROUP BY R.RegionId

```

The operator `VS.IntersectingCertitude` retrieves geometries of certitude elements that intersect w . The GROUP BY clause avoids repeated rows in the result set, since an infected region is referenced by several rows of the fact table HLBControl.

Example 5.9.10. $IRQ_{dubiety-elements}$ and `IntersectingDubiety`.

In contrast with Example 5.9.9, suppose that only parts of an infected region that have possibility of infection less than 1 must be retrieved. These parts must intersect a spatial query window w , which is a rectangle of interest drawn by the user. Then:

```

SELECT    VS_IntersectingDubiety ( InfectedRegion_ElementsGeo ,
                                InfectedRegion_ElementsMval , w)
FROM      InfectedRegion R, InfectedGroup G, HLBControl H,
          Month M, Tree T, Plot P, Farm F, City C

```



```

WHERE   G.RegionFK = R.RegionId AND
        H.GroupFK = G.GroupId AND
        H.MonthFK = M.MonthNo AND
        H.TreeFK = T.TreeId AND
        T.PlotFK = P.PlotId AND
        P.FarmFK = F.FarmId AND
        F.CityFK = C.CityId AND
        C.Name = 'Sao Carlos' AND
        (M.MonthNo = 201401 OR
         M.MonthNo = 201402 OR
         M.MonthNo = 201403)
GROUP BY R.RegionId

```

The operator `VS.IntersectingDubiety` retrieves geometries of dubiety elements that intersect w . The `GROUP BY` clause avoids repeated rows in the result set, since an infected region is referenced by several rows of the fact table `HLBControl`.

Example 5.9.11. *IntersectingElementsExprMval.*

In contrast with Examples 5.9.9 and 5.9.10, suppose that only parts of an infected region that have possibility of infection higher than 80% must be retrieved. Then:

```

SELECT  VS.IntersectingElementsExprMval( InfectedRegion.ElementsGeo ,
        InfectedRegion.ElementsMval , w)
FROM    InfectedRegion R, InfectedGroup G, HLBControl H,
        Month M, Tree T, Plot P, Farm F, City C
WHERE   G.RegionFK = R.RegionId AND
        H.GroupFK = G.GroupId AND
        H.MonthFK = M.MonthNo AND
        H.TreeFK = T.TreeId AND
        T.PlotFK = P.PlotId AND
        P.FarmFK = F.FarmId AND
        F.CityFK = C.CityId AND
        C.Name = 'Sao Carlos' AND
        (M.MonthNo = 201401 OR
         M.MonthNo = 201402 OR
         M.MonthNo = 201403)
GROUP BY R.RegionId

```

The operator `VS.IntersectingElementsExprMval` retrieves geometries of an infected region's elements that both intersect w and have membership value greater than 0.80. The `GROUP BY` clause avoids repeated rows in the result set, since an infected region is referenced by several rows of the fact table `HLBControl`.

Example 5.9.11 illustrates that the operator `VS.IntersectingElementsExprMval` extends the vague spatial predicate $IRQ_{dubiety-elements-mval}$. Different from the predicate, the operator is not constrained to dubiety elements.

5.9.3 Vague Spatial Aggregation Functions

Vague spatial aggregation functions summarize vague spatial data by processing geometries and membership values of vague spatial objects. These operations have been on vague spatial aggregation functions of the VSCube conceptual model described in Section 4.6. This section addresses the vague spatial union.

PostgreSQL demands the creation of an UDT to allow the implementation of an aggregation function. Therefore, the UDT called `VS_AttributeType` has been defined to refer to records containing a geometry and an array of membership values:

```
CREATE TYPE VS_AttributeType AS(
  geom geometry ,
  mval float array
)
```

The union of a pair of vague spatial objects has been implemented as an UDF called `VS_UnionXY` using the procedural language PL/pgSQL from PostgreSQL. The corresponding method's signature is shown in Table 5.5. Furthermore, the union of a set of vague spatial objects has been implemented as an *aggregate*⁸, as also demanded by PostgreSQL:

```
CREATE AGGREGATE VS_Union (VS_AttributeType)(
  sfunc = VS_UnionXY,
  stype = VS_AttributeType
)
```

In detail, the aggregate `VS_Union` processes instances of the UDT `VS_AttributeType` by calling the function `VS_UnionXY` consecutively. In the first call to `VS_UnionXY`, the argument `x` is null, while the argument `y` is a value of type `VS_AttributeType` to be processed. The result of the first execution of `VS_UnionXY` is kept as a partial value and is provided as the argument `x` to call `VS_UnionXY` again, such that the argument `y` is the subsequent value of type `VS_AttributeType` to be processed. Partial values are updated after each call to `VS_UnionXY` and a final value is returned after processing the last call. The corresponding method signature is shown in Table 5.5

Table 5.5: Vague spatial aggregation functions: methods' signatures and descriptions.

Method's signature and description
<code>VS_AttributeType VS_UnionXY(VS_AttributeType x, VS_AttributeType y)</code> Returns the vague spatial union of <code>x</code> and <code>y</code> .
<code>VS_AttributeType VS_Union(VS_AttributeType set)</code> Returns the vague spatial union of a set of objects.

⁸<http://www.postgresql.org/docs/9.3/static/sql-createaggregate.html>

Although the UDT and the aggregate have been created, a vague spatial attribute remains implemented as a pair of columns. The following example demonstrates how the vague spatial union is used to query a vague SDW.

Example 5.9.12. *VSUnion* and *VS_Union*.

Consider the HLB case study and the vague SDW logical schema shown in Figure 5.19. The following SQL query implements the query HLB2 listed in Table 1.2.

```
SELECT  T.TreeId , P.PlotId , M.QuarterNo ,
        VS_Union(ROW(H. InfectedTree_ElementsGeo ,
                    H. InfectedTree_ElementsMval)
                :: VS_AttributeType) AS HLBTrees
FROM    HLBControl H, Tree T, Plot P, Month M
WHERE   H.TreeFK = T.TreeId AND
        T.PlotFK = P.PlotId AND
        H.MonthFK = M.MonthNo AND
        M.YearNo = 2014
GROUP BY T.TreeId , P.PlotId , M.QuarterNo
```

The pair (H.InfectedTree_ElementsGeo, H.InfectedTree_ElementsMval) is transformed into a record using the operator ROW. The record is casted to the UDT VS_AttributeType and passed as argument of VS_Union. This query requires keeping track of the key value of each tree to distinguish trees that were planted in the same place. This happens when a tree is eradicated at a given time due to HLB infection and is thereafter replaced by another tree. Consequently, T.TreeId is mentioned in the GROUP BY clause. In order to facilitate reading the results, the previous query can be surrounded by:

```
SELECT  T.TreeId , P.PlotId , M.QuarterNo ,
        ST_AsText((HLBTrees).geom) , (HLBTrees).mval
FROM(
    —the previous query
) AS HLB2
```

5.9.4 Slice-and-Dice, Roll-Up, and Drill-Down

Vague SOLAP provides a class of operations over a vague SDW logical schema. These operations take as arguments a set of tables and yield as result another table. Therefore, the operations can be nested for expressing complex queries. This section tackles the following operations: *slice*, *dice*, *roll-up*, and *drill-down*. There are not specific UDFs implemented for the vague SOLAP operations. On the other hand, queries can be written using the UDFs described in Sections 5.9.1 to 5.9.3.

Slice and dice belong to vague SOLAP if at least one of the following conditions apply: (i) a crisp spatial attribute is tested against a vague spatial predicate; or (ii) a vague spatial attribute

is involved. Table 5.6 differentiates slice and dice operations in OLAP, SOLAP, and vague SOLAP, according to the queried attribute type and the predicate type.

Table 5.6: Classes of slice and dice operations.

Attribute	Predicate	Class
Conventional	Conventional	OLAP
Crisp Spatial	Crisp Spatial	SOLAP
Crisp Spatial	Vague Spatial	vague SOLAP
Vague Spatial	Crisp Spatial	vague SOLAP
Vague Spatial	Vague Spatial	vague SOLAP

Example 5.9.13. Slice and dice.

Consider the HLB case study and the vague SDW logical schema shown in Figure 5.19. The query HLB7 listed in Table 1.2 and the corresponding SQL query described in Example 5.9.9 perform slice and dice operations. The slice provides the value “São Carlos” to the conventional attribute Name defined in the conventional level table City. A dice provides a set of values to the conventional attribute MonthNo defined in the conventional level table Month. Another dice provides a set of values to the vague spatial attribute implemented as three columns in the vague spatial level table InfectedRegion. Therefore, the query HLB7 and the corresponding SQL query perform an OLAP slice, an OLAP dice, and a vague SOLAP dice.

The operations roll-up and drill-down involve a source level and a target level, according to the traversal of a hierarchy. The target level in a roll-up is a parent level, while the target level in a drill-down is a child level. Roll-up and drill-down operations in vague SOLAP differ from the corresponding operations from OLAP and SOLAP, according to the measures being aggregated and to the target level table. Table 5.7 differentiates roll-up and drill-down operations in OLAP, SOLAP, and vague SOLAP. To sum up, roll-up and drill-down are vague SOLAP operations if at least one of the following conditions apply: (i) aggregation is performed on a vague spatial measure; or (ii) the target level table is a vague spatial level table.

Table 5.7: Classes of roll-up and drill-down operations.

Measure	Target level	Class
Conventional	Conventional	OLAP
Conventional	Crisp Spatial	SOLAP
Conventional	Vague Spatial	vague SOLAP
Crisp Spatial	Conventional	SOLAP
Crisp Spatial	Crisp Spatial	SOLAP
Crisp Spatial	Vague Spatial	vague SOLAP
Vague Spatial	Conventional	vague SOLAP
Vague Spatial	Crisp Spatial	vague SOLAP
Vague Spatial	Vague Spatial	vague SOLAP

Example 5.9.14. Roll-up.

Consider the HLB case study and the vague SDW logical schema shown in Figure 5.19. The query HLB2 listed in Table 1.2 and the corresponding SQL query described in Example 5.9.12 perform a vague SOLAP roll-up operation, since the vague spatial measure `InfectedTree` is aggregated. Furthermore, the query HLB3 listed in Table 1.2 performs a vague SOLAP roll-up operation, as it aggregates the numeric measure `SymptomSeverity` and targets the vague spatial level table `InfectedGroup`.

Example 5.9.15. Drill-down.

Consider the HLB case study and the vague SDW logical schema shown in Figure 5.19. Suppose the execution of the following query: “retrieve the average symptom severity by infected region by month in 2014”. A subsequent execution of the query HLB3 listed in Table 1.2 determines a vague SOLAP drill-down operation, since the hierarchy relating infected regions and infected groups is traversed and the target is the vague spatial level table `InfectedGroup`, which has a finer granularity than the vague spatial level table `InfectedRegion`.

5.10 Summary

This chapter has addressed the logical design of vague SDWs according to the relational model with extensions provided by current DBMSs, such as spatial data types, spatial operators, spatial functions, UDTs, UDFs and multivalued attributes implemented as arrays. Table 5.8 lists the topics that have been addressed regarding the logical design of vague SDWs and the corresponding guidelines or implementations that have been described. Guidelines and implementations have been applied on both the pest control and the HLB case studies.

The logical design of the vague spatial attribute comprises one column to store multiple geometries and one column to store the array of membership values, such that the i -th geometry refers to the i -th entry in the array and both describe the i -th (certitude or dubiety) element of the vague spatial object. Optionally, another geometry column can store the geometric union of the elements. Such approach, described by Rule 1VS, ensures the compliance with OGC standards, enables vague spatial attributes whose both certitude and dubiety are monovalued or multivalued, avoids joins among tables to process vague spatial predicates and vague spatial aggregation functions, allows indexing of the aforementioned columns and has geometries sorted according to membership values. A shortcoming concerns the execution of internal DBMS functions, e.g. to retrieve the geometry and the membership value of the i -th element in a vague spatial object. Although such approach has been chosen, several alternative approaches have also been compared and discussed. A vague spatial attribute requires a set of constraints to allow valid

Table 5.8: The logical design of vague SDWs and the corresponding guidelines or implementations.

Logical Design	Guidelines / Implementations
Vague Spatial Attribute	Rule 1VS
Constraints on the Vague Spatial Attribute	UDF: VS_Constraints_VSAttribute
Vague Spatial Level	Rule 2VS
Intra-level Vague Topological Constraints	UDF: VS_Constraints_Topological_IntraLevel_IntraFact
Vague Spatial Hierarchy	Rule 3VS
Hierarchy Vague Topological Constraints	UDF: VS_Constraints_Topological_Hierarchy
Fact and Vague Spatial Measure	Rule 4VS
Intra-fact Vague Topological Constraints	UDF: VS_Constraints_Topological_IntraLevel_IntraFact
Spatial Fact Vague Topological Constraints	UDF: VS_Constraints_SpatialFact
Vague Spatial Fact	Rule 5VS
Accessors	UDFs: VS_Elements, VS_ElementsExprMval, ...
Vague Spatial Predicates	UDFs: VS_Intersects_Certitude, ...
Vague Spatial Aggregation Functions	UDT: VS_AttributeType; UDF: VS_Union
Vague SOLAP	SQL queries

geometries and arrays of membership values for elements of vague spatial objects, which have been implemented as an UDF.

Once the logical design of the vague spatial attribute became feasible, a set of guidelines has been proposed to map a conceptual schema of vague SDW into a logical schema of vague SDW under the relational model. Vague spatial levels, hierarchies, and fact and vague spatial measures have been designed according to Rules 2VS, 3VS, and 4VS, respectively. Sets of topological relationships are ensured for pairs of vague spatial objects that belong to the same level or to the same fact, or that are associated through a hierarchy, or that are related by a spatial fact. All constraints have been implemented as UDFs and triggers, such that UDFs are independent of application and triggers are created according to the requirements of the application for which the vague SDW is built. Furthermore, Rule 5VS has been proposed for the logical design of the vague spatial fact, enabling values of measures to be assigned to elements of vague spatial members.

The aforementioned guidelines and implementations enable the design of the logical schema for a vague SDW. The logical schema of a vague SDW is the collection of its tables created accordingly. The tables of the logical schema are obtained by applying Rules 1VS-5VS. Furthermore, the vague topological constraints are ensured in all the tables of the schema.

Once the logical design of the vague SDW became feasible, vague SOLAP has been tackled with the elaboration of several operations implemented as UDFs in the DBMS. Accessors that provide access to elements of vague spatial objects. Vague spatial predicates select vague spatial objects according to criteria involving topological relationships and conditionals regard-

ing membership values. Aggregation functions that aggregate vague spatial data aiming at data summarization. These UDFs extend SQL and are essential for querying the vague SDW schema using vague SOLAP operations, such as slice-and-dice, roll-up, and drill-down.

Although efforts have been made in the logical design to provide an implementation to achieve a reasonable performance on query processing, only an experimental evaluation can provide enough details about the cost to manipulate vague spatial data in vague SDWs and motivate the creation of indices to speedup query processing. The evaluation of the performance to process queries over a vague SDW and the creation of indices for vague SDW are tasks related to the physical design of vague SDWs, which is addressed in Chapter 6.

Chapter 6

PHYSICAL DESIGN OF VAGUE SPATIAL DATA WAREHOUSES

The physical design of a relational database aims at improving the performance at runtime for query processing. After investigating the workload and mainly the columns used in query predicates, a technique such as indexing or view materialization is applied. The physical design of a data warehouse implemented as a relational database additionally considers the huge volume of data, the predominance of denormalized schemata and a massive computation of joins between tables, aggregations, conventional predicates and sorting. The physical design of a spatial data warehouse is even more challenging due to the storage of spatial data, to the resolution of spatial predicates, and to the computation of spatial data aggregation, since spatial data are more complex than conventional data

This chapter addresses the physical design of vague spatial data warehouses and introduces an index for processing queries against vague regions, called Vague Spatial Bitmap index (VSB-index). An overview of indexing vague spatial data warehouses is outlined in Section 6.1. Section 6.2 describes an experimental evaluation that assessed the performance of an existing database management system (DBMS) and of existing indices that were designed for spatial data warehouses, as well as discusses and compares the results. Section 6.3 details the VSB-index. Section 6.4 describes an experimental evaluation that assessed the performance of the VSB-index and discusses the results. Finally, Section 6.5 summarizes the chapter.

6.1 Indexing Vague Spatial Data Warehouses

The physical design of a vague spatial data warehouse (vague SDW) implemented as a relational database aims at improving the performance to process queries. The workload of a vague

SDW comprises queries that require processing joins between tables, computing aggregation of conventional data, crisp spatial data, and vague spatial data, resolving conventional predicates, spatial predicates, and vague spatial predicates, and sorting the results. Furthermore, these queries fetch both crisp and vague spatial data characterized by different types and complexity.

The physical design of a vague SDW starts by examining the extent to which existing indices can benefit the performance to process queries of the workload. Bitmap indices avoid joins between tables, compute aggregation of conventional data and resolve conventional predicates in data warehouses. Spatial indices offered by DBMSs aid to compute the aggregation of spatial data and to resolve spatial predicates in spatial data warehouses. Spatial indices are also reused or combined to bitmap indices for developing indices for spatial data warehouses. Indices for vague regions focuses on the resolution of particular predicates and have not yet been applied to vague SDWs.

Bottlenecks and limitations of existing indices can be identified by carrying out an experimental evaluation of existing solutions using a workload of a vague SDW. The analysis of the results can indicate necessary adaptations on their data structures or query processing algorithms to improve the performance of one or more operations. Moreover, such analysis can provide substantial information for motivating the design of a new index to overcome the drawbacks faced by existing indices.

The design of an index for vague SDW starts by selecting a subset of the operations required by queries in the workload. In addition, the diversity of vague spatial data types and their shapes may impair the development of an index able to address all vague spatial data types. In particular, if vague spatial objects are implemented using geometries, a multistep resolution of the spatial predicate can be adopted. Then, the filter step should reduce as most as possible the set of candidates using approximations of vague spatial objects. Such reduction tends to decrease the number of geometries stored in secondary memory to be fetched in the refinement step. An experimental evaluation using a workload of a vague SDW is mandatory for the proposed index to corroborate its benefits for the performance to process queries.

This chapter describes an experimental evaluation of both a DBMS and existing indices for spatial data warehouses. It also describes the VSB-index for indexing vague regions in vague SDW and processing range queries. An experimental evaluation corroborates that the VSB-index improves the performance to process queries in vague SDW.

6.2 Evaluation of a DBMS and of Indices for Spatial Data Warehouses

The following sections describe an experimental evaluation of a DBMS and existing indices for SDW. The evaluation focuses on the storage requirements of vague SDWs and the performance to process queries over vague SDWs. The DBMS chosen for the experiments was PostgreSQL¹ with the spatial extension PostGIS (OBE; HSU, 2015), as they offer spatial data types and operators that can be reused to design logical schemata of vague SDW, as discussed in Chapter 5. Also, the SB-index and the aR-tree were selected as the existing indices for SDW, since they efficiently process spatial range queries over SDW, as already discussed in Section 3.3.

Regarding storage requirements, the influence of the following factors are analyzed:

- the use of vague spatial data types whose complexities vary, such as vague point set and vague region; and
- the adoption of different logical designs for the vague SDW, as those addressed in Chapter 5.

Concerning the performance to process queries, both the aforementioned factors and the following factors are investigated:

- the resolution of different vague spatial predicates that might or not require the refinement step; and
- the resolution of vague spatial predicates with increasing selectivities.

Section 6.2.1 addresses containment range queries issued over a vague SDW containing vague point sets. Section 6.2.2 tackles intersection range queries executed in a vague SDW containing vague regions. Section 6.2.3 focuses on vague spatial range queries issued over a vague SDW containing vague regions.

6.2.1 Containment Range Queries against Vague Point Sets

The following sections describe an evaluation of the DBMS. The vague SDW used in the experiments stored vague point sets. The queries issued over the SDW encompassed the vague

¹<http://www.postgresql.org>

spatial predicate CRQ_{Object} , which does not require a refinement step. The effects of increasing the complexity of vague point sets are also evaluated. Section 6.2.1.1 details the workbench and platforms, Section 6.2.1.2 describes the workload and Section 6.2.1.3 addresses the results.

6.2.1.1 Workbench and Platforms

The workbench was based on the Star Schema Benchmark (O'NEIL et al., 2009), whose DW addresses a retail application. The data generator of the Star Schema Benchmark was executed adjusting the scale factor to 10. The generated data was loaded into the DBMS. Then, the original DW schema of the cited benchmark was transformed into a vague SDW schema concerning the HLB study, as shown in Figure 6.1. Modifications that were made to the schema and details about the data volume are described in the following.

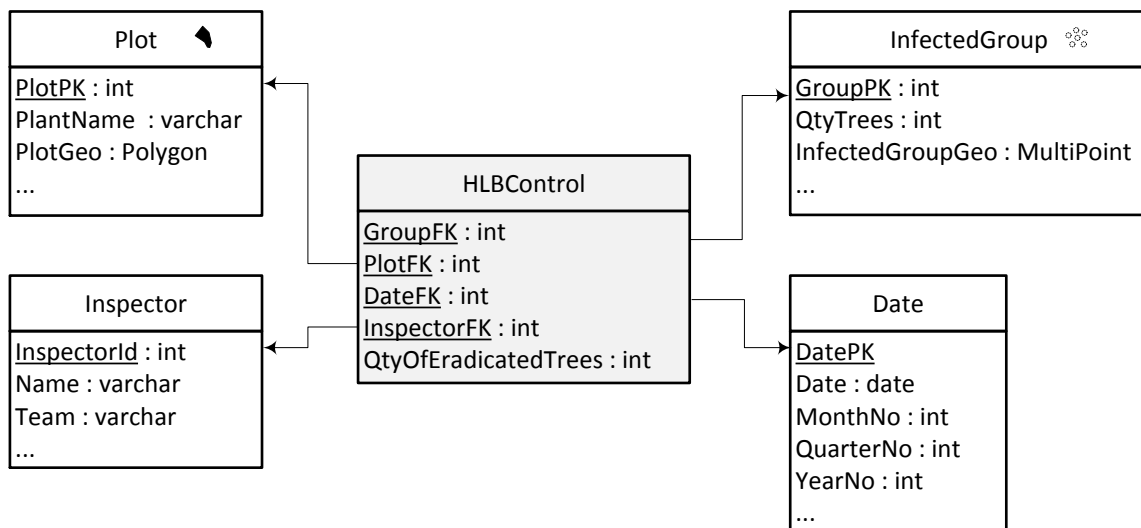


Figure 6.1: The vague SDW storing infected groups as vague point sets.

The original fact table *LineOrder* was renamed to *HLBControl*, its original column *Quantity* was renamed to *QtyOfEradicatedTrees*, and it stored approximately 60,000,000 rows. The original table *Date* was kept unmodified and stored 2,556 rows. The original table *Part* was renamed to *Inspector*, its original column *Brand* was renamed to *Team* and it stored approximately 800,000 rows. The original table *Supplier* was renamed to *Plot* and had the crisp spatial attribute *PlotGeo* added and implemented as a *Polygon* as shown in Figure 6.1. However, crisp spatial data were not loaded in the table *Plot* because the workload described in Section 6.2.1.2 focus on querying vague spatial data. The remaining original columns were also renamed accordingly to comply with the HLB case study.

Furthermore, the original table *Customer* was renamed to *InfectedGroup*. The vague spatial attribute *InfectedGroup* denoted vague point sets whose certitude located certainly infected trees and dubiety indicated possibly infected trees. Such attribute was implemented as a column of type *MultiPoint* and added to the table *InfectedGroup*. A GiST was created to index the column *InfectedGroupGeo*. Differently from the logical design proposed in Section 5.2.5, a column for the array of membership values was not added to the table *InfectedGroup*, since the certitude and the dubiety of the vague point set were not queried separately, as detailed in Section 6.2.1.2.

Furthermore, the vague point sets of the column *InfectedGroup* were generated as follows. A dataset containing 302,357 real polygons was gathered from the rural census of the Brazilian Institute of Geography and Statistics (IBGE)². Each real polygon was used to create one vague point set. Firstly, the centroid of the real polygon was obtained. Then, a square was temporarily built around the centroid. Finally, a set of n points was placed on the edges of the square and then added to the multipoint. For the sake of simplicity, the first point added to the multipoint was the single element of the certitude, while the remaining points were considered elements of the dubiety. Figure 6.2a illustrates a real polygon, its centroid and the temporary square used to place the points of the vague point, while Figure 6.2b depicts the generated vague point set considering with four points, i.e. $n = 4$.

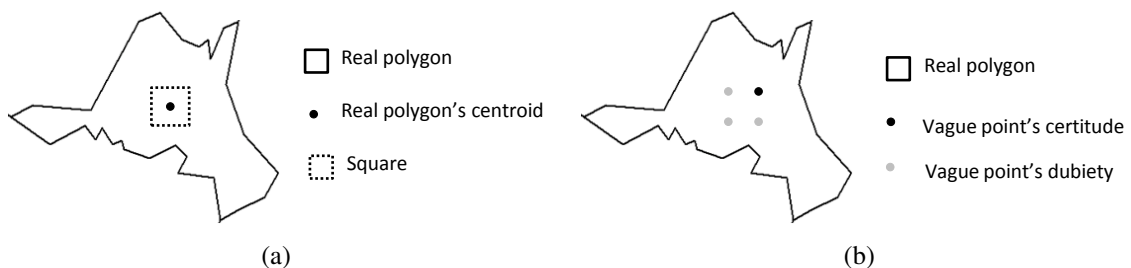


Figure 6.2: Creating vague point sets: (a) Processing the real polygon. (b) The vague point set.

Three different versions of the table *InfectedGroup* were created: *InfectedGroup4*, *InfectedGroup12*, and *InfectedGroup24*. The first version had $n = 4$ in the procedure that created vague point sets. As a result, its multipoints had 4 points. Analogously, the second and the third versions had $n = 12$ and $n = 24$, respectively.

The hardware platform was a computer with a 3.2 GHz Pentium D processor, 8 GB of main memory, a 7200 RPM SATA 750 GB hard disk with 32 MB of cache. The software platform had Linux CentOS 5.2 PostgreSQL 8.2.5 and PostGIS 1.3.3.

²<http://www.ibge.gov.br>

6.2.1.2 Workload

The workload of the experimental evaluation was also based on the Star Schema Benchmark, as follows. Query Q2.3 of the aforementioned benchmark, which encompasses typical characteristics of an OLAP query, such as joins among huge tables, conventional predicates, aggregation and sorting, was modified to comply with vague SDW and vague SOLAP. One of the original conventional predicates was replaced by a CRQ_{object} to select vague point sets. The template query for the workload is detailed in Listing 6.1. The conditional $ST_Within(InfectedGroup, w)$ played the role of the CRQ_{object} , where w was a rectangular spatial query window whose shape was not stored in the vague SDW. Figure 6.3 illustrates one spatial query window q containing two vague point sets.

Listing 6.1: The template query for the SDW with vague point sets.

```

SELECT Team, Year, SUM(QtyOfEradicatedTrees)
FROM Inspector, Date, InfectedGroup, HLBControl
WHERE InspectorPK = InspectorFK AND
      DatePK = DateFK AND
      GroupPK = GroupFK AND
      Team = 'XV' AND
      ST_Within(InfectedGroupGeo, w)
GROUP BY Team, Year
ORDER BY Team, Year;

```

Five disjoint spatial query windows were created and assigned to one query. The area of each spatial query window corresponded to 0.10% of the extent where the vague point sets were distributed. The five queries were executed once using the table `InfectedGroup4`, and thereafter cache and buffers were flushed. Then, the five queries were executed once using the table `InfectedGroup12`, and thereafter cache and buffers were flushed. Finally, the five queries were executed once using the table `InfectedGroup24`.

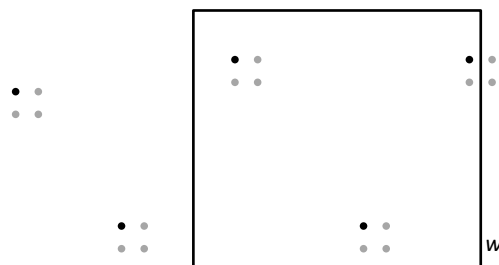


Figure 6.3: The spatial query window w containing two vague point sets.

6.2.1.3 Results

Figure 6.4a reports the storage requirements demanded by each version of the table `InfectedGroup` described in Section 6.2.1.1. They stored conventional attributes and multipoints

encompassing 4, 12 or 24 points. Clearly, more storage space was required as the number of points of the vague point set increased.

Furthermore, the workload described in Section 6.2.1.2 was executed using the workbench detailed in Section 6.2.1.1. The average elapsed times were gathered and reported in Figure 6.4b. The increase on the number of points of the vague point set also added an overhead to the query response time. In general, the average elapsed times were of approximately 1 minute and, therefore, prohibitive.

It is noteworthy that each version of the table `InfectedGroup` had one GiST built on the column `InfectedGroupGeo`. Every GiST built a MBR for each vague point set, such that the shape of the MBR was identical to the square used to create the vague point set, as shown in Figure 6.2a. However, the squares used to create vague point sets containing 4 points differed from the squares used to create vague point sets encompassing 12 or 24 points. Consequently, the three GiSTs were distinct as they stored different MBRs. The performance to execute the workload was influenced by the performance achieved by each GiST to process the CRQ_{object} and by the computation of the other components of the query, i.e. conventional predicate, joins, aggregation and sorting.

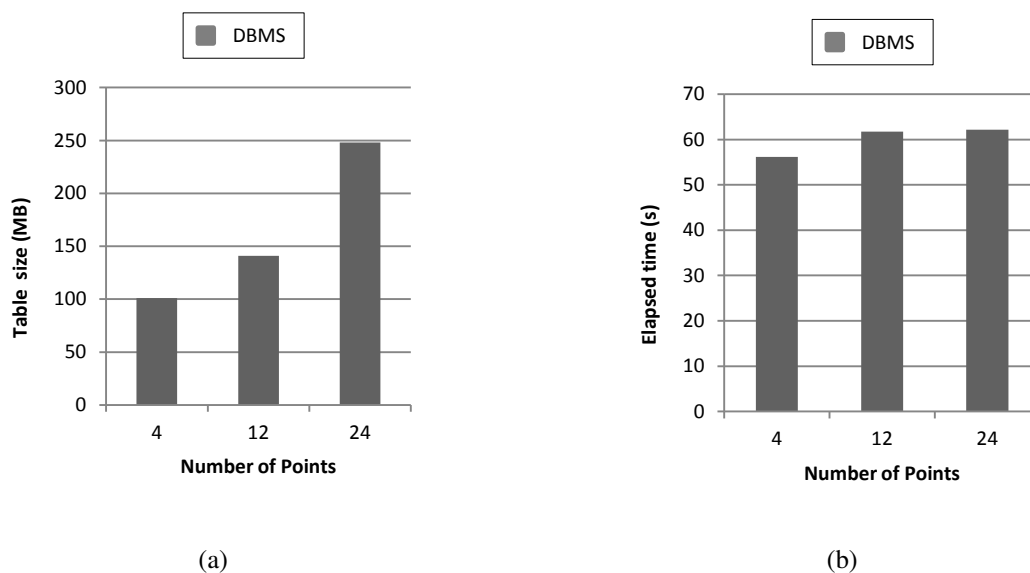


Figure 6.4: Performance results of the DBMS for the vague SDW containing vague point sets: (a) Storage requirements. (b) Query processing.

6.2.2 Intersection Range Queries against Vague Regions

The following sections describe an evaluation of the DBMS and existing indices for SDW. In contrast with Section 6.2.1, which addressed vague point sets and CRQ_{object} , the follow-

ing sections tackle vague regions as a more complex type of vague spatial data and IRQ_{object} as a more costly vague spatial predicate whose resolution requires a refinement step. Furthermore, the effects of increasing selectivities for IRQ_{object} are investigated, and the costs to resolve the vague spatial predicate using each index are also analyzed. Section 6.2.2.1 details the workbench and platforms used in the experiments, Section 6.2.2.2 describes the workload and Section 6.2.2.3 reports and discusses the results.

6.2.2.1 Workbench and Platforms

The workbench was based on the vague SDW described in Section 6.2.1.1 as the tables Inspector, Date, Plot and HLBControl were maintained unaltered and with the same data volume. On the other hand, two distinct tables were created to store vague regions representing the infected region. They allowed to analyze whether different logical designs of the vague SDW can reduce the query response time.

The incomplete schema of vague SDW shown in Figure 6.5a was completed with one of the tables depicted in Figure 6.5b, i.e. InfectedRegion1 or InfectedRegion2. The table InfectedRegion1 implemented vague regions as multipolygons, similarly to the logical design described in Section 5.2.5. Conversely, the table InfectedRegion2 implemented vague regions as a pair of polygons, i.e. one for the certitude and one for the dubiety, analogously to the logical design addressed in Section 5.2.6. Columns for the arrays of membership values were not created because the workload described in Section 6.2.2.2 focused only on geometries. GiSTs were built on the columns InfectedRegionGeo, InfectedRegionCertitudeGeo and InfectedRegionDubietyGeo.

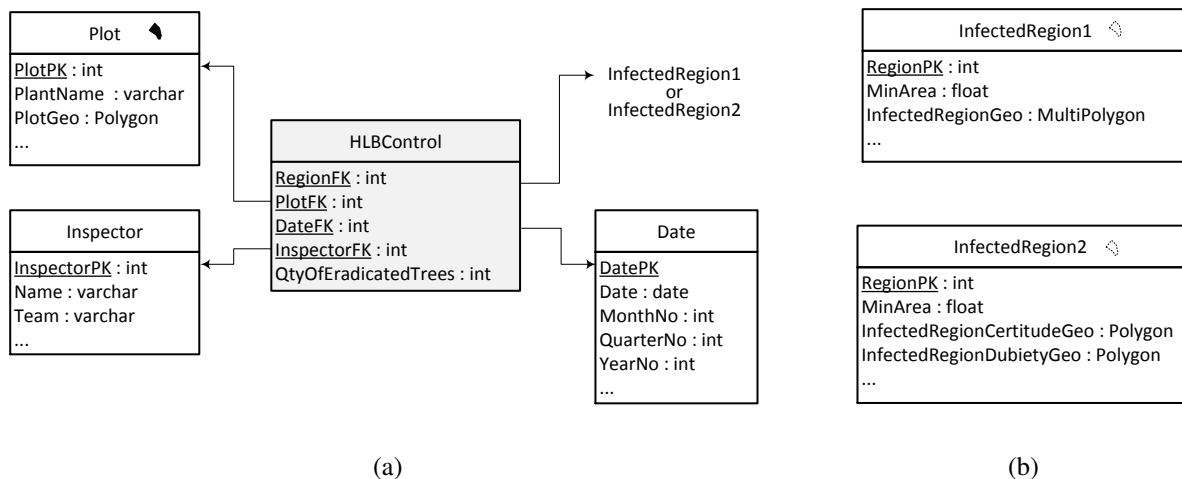


Figure 6.5: The schema of the vague SDW storing vague regions: (a) Incomplete schema. (b) Tables to complete the schema.

The same vague regions were created to load the tables `InfectedRegion1` and `InfectedRegion2` by processing 302,357 polygons gathered from the rural census of IBGE. Each real polygon was processed to create one vague region. Firstly, the certitude was obtained by applying a negative buffer on the real polygon. Secondly, the convex hull of the real polygon was computed. Finally, the dubiety was obtained by computing the difference between the cited convex hull and the certitude. All the created vague regions were regions with broad boundaries, such that the dubiety surrounded the certitude. Figure 6.6a illustrates a real polygon, its convex hull and a negative buffer applied to it, while Figure 6.6b depicts the obtained vague region.

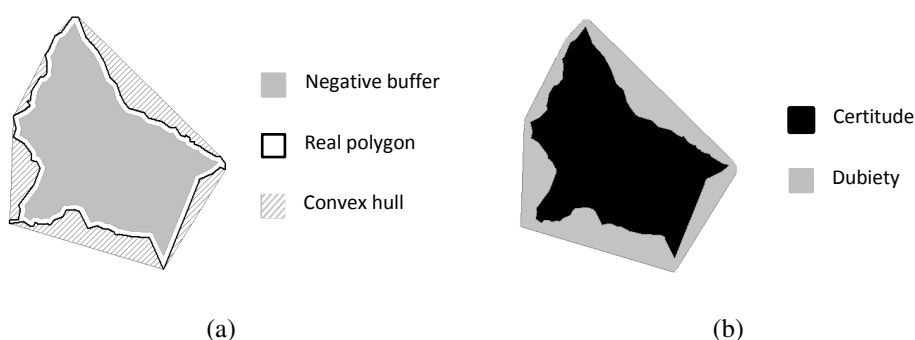


Figure 6.6: From real polygons to vague regions: (a) Processing the real polygon. (b) The obtained vague region.

The SB-index and the aR-tree were implemented in C/C++ and their disk page size was set to 8 KB. Both were built on the column `InfectedRegionGeo` and, therefore, processed their refinement steps accessing the multipolygons. The bitmap join indices of the SB-index were implemented using FastBit (WU et al., 2009) and built on the columns `RegionFK`, `Team`, `Year` and `QtyOfEradicatedTrees` to comply with the workload described in Section 6.2.2.2. Analogously, the entries of the aR-tree referenced one multidimensional array containing values of the measure `QtyOfEradicatedTrees` assigned to each pair of values for `Team` and `Year`.

The hardware 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. The software platform encompassed Linux CentOS 6, PostgreSQL 9.2, PostGIS 2.0.1 and FastBit 1.3.5.

6.2.2.2 Workload

IRQ_{object} was chosen as the vague spatial predicate for the workload as it requires a refinement step after the filter step to be resolved. The template queries issued over the schema shown in Figure 6.6a took into account the particularities the tables `InfectedRegion1` and `InfectedRegion2` described in Figure 6.6b, as follows. The template query detailed in Listing 6.2

fetches vague regions implemented as multipolygons in the table `InfectedRegion1`. The conditional `ST.Intersects(InfectedRegionGeo, w)` played the role of the *IRQ_{object}*, where `w` was a rectangular spatial query window whose shape was not stored in the vague SDW. Analogously, the template query detailed in Listing 6.3 fetched vague regions implemented as a pair of polygons in the table `InfectedRegion2`. The conditional `ST.Intersects(InfectedRegionDubietyGeo, w)` played the role of the *IRQ_{object}* since the outer boundary of the dubiety was equivalent to the outer boundary of the vague region. This characteristic can be visualized in Figure 6.6b.

Listing 6.2: The template query for the vague SDW with the table `Infection1`.

```
SELECT Team, Year, SUM(QtyOfEradicatedTrees)
FROM Inspector, Date, InfectedRegion1, HLBCControl
WHERE InspectorPK = InspectorFK AND
      DatePK = DateFK AND
      RegionPK = RegionFK AND
      Team = 'XV' AND
      ST.Intersects(InfectedRegionGeo, w)
GROUP BY Team, Year
ORDER BY Team, Year;
```

Listing 6.3: The template query for the vague SDW with the table `Infection2`.

```
SELECT Team, Year, SUM(QtyOfEradicatedTrees)
FROM Inspector, Date, InfectedRegion2, HLBCControl
WHERE InspectorPK = InspectorFK AND
      DatePK = DateFK AND
      RegionPK = RegionFK AND
      Team = 'XV' AND
      ST.Intersects(InfectedRegionDubietyGeo, w)
GROUP BY Team, Year
ORDER BY Team, Year;
```

Figure 6.7 exemplifies one rectangular spatial query window `w` intersecting several vague regions of the vague SDW. The generation of spatial query windows was based on the selectivity of the vague spatial predicate *IRQ_{object}*. Ten rectangular spatial query windows were created for each one of the following selectivities for *IRQ_{object}*: 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. Then, a total of 130 disjoint spatial query windows were obtained. The aforementioned selectivities ensured that 300 up to 12,000 vague regions of the synthetic vague SDW were queried. Greater selectivities were not utilized since a user of SOLAP tool would hardly select more than 12,000 vague regions to be retrieved and displayed. The workload comprised 130 queries, each one using a distinct spatial query window. The execution of queries complied with the ascending order of the selectivity. Cache and buffers were flushed each ten queries, i.e., before starting the execution of a query with greater selectivity.

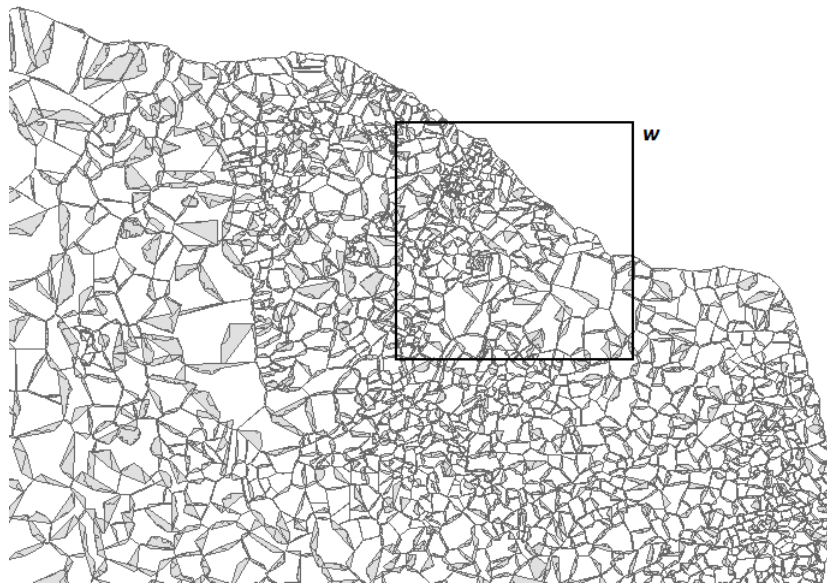


Figure 6.7: The spatial query window w intersecting several vague regions of the vague SDW.

6.2.2.3 Results

Concerning the storage requirements, the table `InfectedRegion1` occupied 2.27 GB while the table `InfectedRegion2` occupied 1.36 GB. Comparing these results with those discussed in Section 6.2.1.3, geometries with a higher complexity such as vague regions demanded more storage requirements. In addition, vague regions designed as a pair of polygons in table `InfectedRegion2` demanded less storage space than if designed as a single multipolygon in table `InfectedRegion1`. As for the indices, the SB-index required 3.14 GB while the aR-tree demanded 16.61 GB. The main storage cost of the indices was not related to the vague regions, since the latter were approximated by MBRs. Rather, bitmap join indices of the SB-index and multidimensional arrays of the aR-tree represented more than 99% of their storage requirements.

Figure 6.8a reports the average elapsed times spent by the DBMS, the SB-index and the aR-tree to process the workload described in Section 6.2.2.2 over the vague SDW detailed in Section 6.2.2.1. Configuration `DBMS1` refers to the DBMS executing the template query described in Listing 6.2, while configuration `DBMS2` refers to the DBMS processing the template query shown in Listing 6.3. Clearly, as higher the selectivity was, longer was the time spent to process queries using the DBMS or the indices. Furthermore, the separation of the vague spatial attribute of type multipolygon in two attributes of type polygon benefited the performance, since `DBMS2` spent less time than `DBMS1` to process queries. Such finding corroborated the adoption of the logical design described in Section 5.2.6 for regions with broad boundaries whose both the certitude and the dubiety are monovalued.

However, both `DBMS1` and `DBMS2` spent prohibitive query response times and indicated

the necessity of using indices to provide a better performance. As for the indices, the aR-tree greatly overcame the DBMS and the SB-index for selectivities below 0.01, while the SB-index overcame the DBMS and the aR-tree for selectivities of 0.01 and above. Even though, both indices spent prohibitive times to process queries with increasing values of selectivity.

The time spent by both indices on each phase of their query processing algorithms was measured to identify the bottleneck in the performance. The fraction to resolve the spatial predicate is reported in Figure 6.8b. Clearly, the resolution of the vague spatial predicate IRQ_{Object} against vague regions was a costly step to process queries over the vague SDW. As 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. Conversely, the cost to process IRQ_{Object} decreased in the aR-tree for increasing values of selectivity since the manipulation of the multidimensional arrays imposed an even larger overhead.

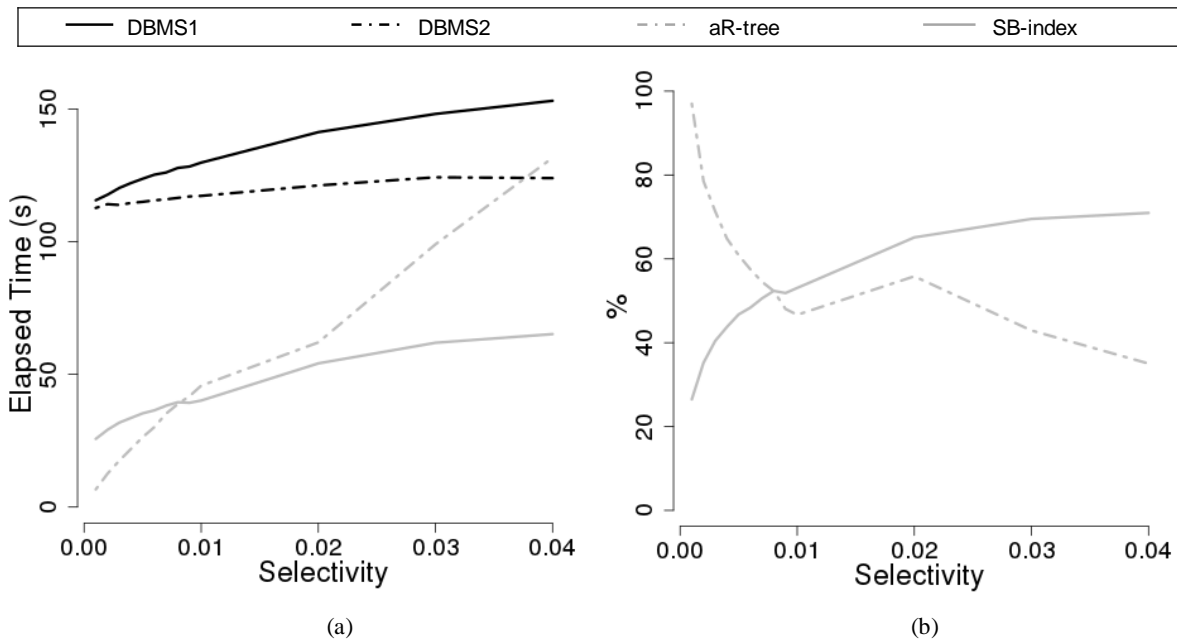


Figure 6.8: Results for the DBMS and indices for SDWs: (a) Average elapsed time. (b) Fraction of the vague spatial predicate.

6.2.3 Vague Spatial Range Queries against Vague Regions

The following sections describe an evaluation of the DBMS and the SB-index regarding vague regions queried by the vague spatial range query $VSRQ_{Object}$, which classifies results according to their relevance using two spatial query windows. Since the original query processing algorithm of the SB-index does not support $VSRQ_{Object}$, an adaption is proposed. The costs to resolve the vague spatial predicate using the SB-index are also analyzed. Section 6.2.3.1 details

the workbench and platforms used in the experiments, Section 6.2.3.2 describes the workload, Section 6.2.3.3 tackles the adaption of the SB-index, and Section 6.2.3.4 reports the results.

6.2.3.1 Workbench and Platforms

The workbench comprised the vague SDW whose schema is depicted in Figure 6.9. Table *Infection* stored exclusively conventional attributes, while the table *InfectedRegion1* stored vague regions implemented as multipolygons and a surrogate key referenced by the table *Infection*. Although there was an 1:1 association among the values of *InfectionPK* and *InfectedRegionGeo*, the objective of proposing such schema was to analyze whether an alternative logical design for the vague SDW could reduce the query response time.

The tables *Inspector*, *Date*, *Plot* and *HLBControl* were identical to those described in Section 6.2.2.1. The data generation was also similar to those described in Section 6.2.2.1. Additionally, the original table *Customer* of the Star Schema Benchmark (O'NEIL et al., 2009) was renamed to *Infection*. The table *InfectedRegion1* was created separately and was loaded with vague regions according to the same procedure already detailed in Section 6.2.2.1.

The DBMS stored the vague SDW. The SB-index was implemented in C/C++ and its disk page size was set to 4 KB. It was built on the column *InfectedRegionGeo* and its bitmap join indices were built on the columns *RegionFK*, *Team*, *Year* and *QtyOfEradicatedTrees* to comply with the workload described in Section 6.2.3.2. Therefore, it processed the refinement step accessing the column *InfectedRegionGeo* of type *MultiPolygon*. The bitmap join indices of the SB-index were implemented using *FastBit* (WU et al., 2009). A slight modification on SB-index' query processing algorithm was developed to process *VSRQ_{object}*, as described in Section 6.2.3.3.

The hardware 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. The software platform encompassed Linux CentOS 5.2, PostgreSQL 8.2.5, PostGIS 1.3.3, and *FastBit* 0.9.2b.

6.2.3.2 Workload

The *VSRQ_{object}* was chosen as the vague spatial predicate for the workload. The template query detailed in Listing 6.4 fetches vague regions of the vague SDW whose schema is shown in Figure 6.9. One sub-query classifies the results as more relevant by selecting vague regions inside the inner spatial query window *i* with the conditional *ST_Within(InfectedRegion, i)*. The other sub-query classifies the results as less relevant by selecting vague regions that intersect the outer spatial query window *o* but that are not inside the inner spatial query window *i* with the

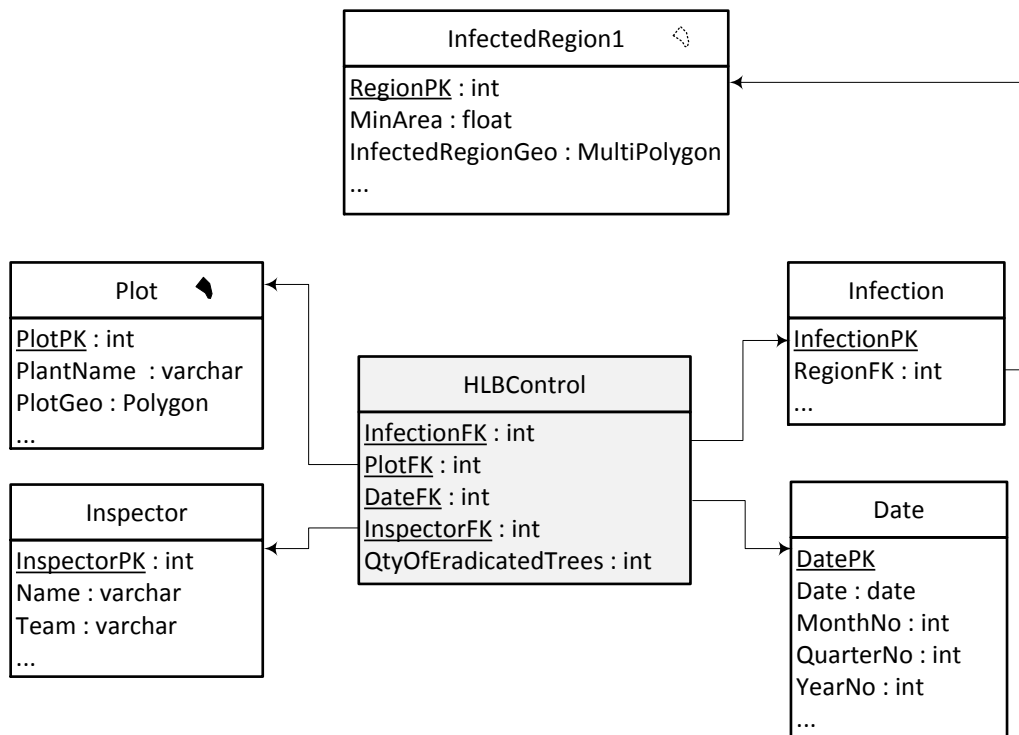


Figure 6.9: The vague SDW storing vague regions in a separate table.

conditional $ST_Intersects(InfectedRegion, o)$ AND NOT $ST_Within(InfectedRegion, i)$. The operator UNION combines the result sets of the sub-queries. This source code in SQL can be generated using the UDF `VS_RangeQueryObject` described in Section 5.9.2 (Example 5.9.7).

The spatial query windows i and o were rectangular and generated such that i covered 0.25% of the extent and o covered 0.5% of the extent. Five disjoint pairs of these spatial query windows were created. The template query was executed 5 times, i.e. once for each pair of spatial query windows. Cache and buffers were not flushed between different queries.

Listing 6.4: The template query for the VSRQobject.

```

SELECT Team, Year, SUM(QtyOfEradicatedTrees), 'More relevant'
FROM Inspector, Date, Infection, InfectedRegion1, HLBControl
WHERE InspectorPK = InspectorFK AND
      DatePK = DateFK AND
      InfectionPK = InfectionFK AND
      RegionPK = RegionFK AND
      Team = 'XV' AND
      ST_Within(InfectedRegionGeo, i)
GROUP BY Team, Year
ORDER BY Team, Year

UNION

SELECT Team, Year, SUM(QtyOfEradicatedTrees), 'Less relevant'
FROM Inspector, Date, Infection, InfectedRegion1, HLBControl
WHERE InspectorPK = InspectorFK AND
      DatePK = DateFK AND

```

```

        InfectionPK = InfectionFK AND
        InfectedRegionPK = InfectedRegionFK AND
        Team = 'XV' AND
        ST_Intersects(InfectedRegionGeo, o) AND
        NOT ST_Within(InfectedRegionGeo, i)
GROUP BY Team, Year
ORDER BY Team, Year

```

6.2.3.3 Extending the SB-index

Since the SB-index was not originally proposed to support vague spatial range queries, a slight modification on the SB-index' query processing algorithm was performed to allow it to process $VSRQ_{object}$. The principle adopted was to execute each sub-query separately and merge their result sets. The original algorithm was kept unaltered to execute the first sub-query, whose filter step checked whether spatial objects were within the inner spatial query window. On the other hand, the filter step for the second sub-query checked whether the MBRs of the spatial objects intersected the outer spatial query window. Besides, the refinement step of the second sub-query was changed to check whether the candidates produced by the filter step both intersected the outer spatial query window and were not within the inner spatial query window. Finally, the access to bitmap join indices for the second sub-query was not altered.

For example, considering the template query described in Listing 6.4, the query executed in the DBMS for the refinement step of the second sub-query is equivalent to the following SQL query. Note that *candidates* is a string of comma-separated key values of the corresponding candidates previously identified in the filter step.

```

SELECT   RegionPK
FROM     InfectedRegion1
WHERE    RegionPK IN (candidates) AND
         ST_Intersects(InfectedRegionGeo, o) AND
         NOT ST_Within(InfectedRegionGeo, i)
ORDER BY RegionPK

```

6.2.3.4 Results

Figure 6.10 compares the average elapsed times to process the workload described in Section 6.2.3.2 over the vague SDW detailed in Section 6.2.3.1 using the SB-index and the DBMS. The alternative logical design of the vague SDW did not benefit the performance to process queries using the DBMS. Although the SB-index outperformed the DBMS, both spent prohibitive times to process the queries, i.e. more than 90 seconds. These results corroborated the necessity of a novel index for vague SDW to process queries encompassing vague spatial range queries such as $VSRQ_{object}$.

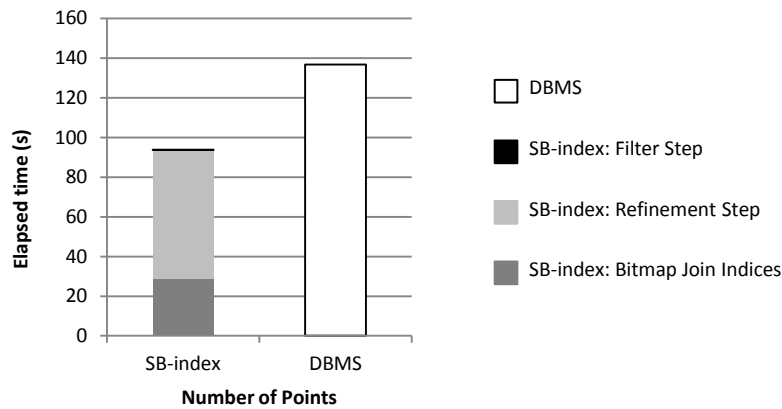


Figure 6.10: Elapsed time to process $VSRQ_{object}$ over the vague SDW containing vague regions.

Furthermore, each step of the query processing algorithm of the SB-index was analyzed to identify bottlenecks. The SB-index' filter step was performed twice: one for each spatial query window. Together, they represented the undermost fraction of 0.10% of the total elapsed time to process the queries. On the other hand, the refinement step to check which spatial objects are intersected by the spatial query window o but are not within the spatial query window i consumed 69.04% of the total elapsed time. Finally, the access to the bitmap join indices represented a fraction of around 30.86%. These results demonstrated that the refinement step imposed the greater overhead to process queries with the vague spatial predicate $VSRQ_{object}$ using the SB-index.

6.2.4 Discussion

Sections 6.2.1 to 6.2.3 have described experimental evaluations of a DBMS and existing indices for SDW and discussed the results concerning storage requirements for vague SDWs and performance to process queries over vague SDWs.

Section 6.2.1 has focused a simpler type of vague spatial data, i.e. the vague point set, and the theoretically costless vague spatial predicate CRQ_{object} that does not require a refinement step. The inclusion of vague spatial data into the SDW impaired both the storage requirements and the performance to process queries using the DBMS. Furthermore, the increase on the complexity of vague point sets also increased both the storage requirements and the elapsed time to process queries.

Section 6.2.2 has addressed vague regions and the vague spatial predicate IRQ_{object} that requires a refinement step, aiming to increase the complexity of both the vague spatial data type and vague spatial predicate. The results revealed that the inclusion of vague regions into

the SDW increased the storage requirements and that both the DBMS and indices designed for SDW spent prohibitive times to process queries with IRQ_{object} . Furthermore, as the selectivity of IRQ_{object} increased, longer became the query response time and more costly became the resolution of such vague spatial predicate.

Finally, 6.2.3 has tackled vague regions and the vague spatial predicate $VSRQ_{object}$. The latter demanded an adaption on the query processing algorithm of the SB-index, since it does not originally supports the $VSRQ_{object}$. Similarly to the previous sections, the results revealed that both the DBMS and the SB-index spent prohibitive times to process queries over vague SDWs. Besides, regarding the adapted SB-index, the refinement step was identified as being the more costly step to process queries with $VSRQ_{object}$.

The aforementioned results identified drawbacks that impair the performance to process queries over vague SDW using the DBMS and existing indices for SDW. They have also motivated the development of an index for vague SDW to efficiently process vague spatial predicates. In Section 6.3, the Vague Spatial Bitmap Index is proposed to comply with this purpose.

6.3 The Vague Spatial Bitmap Index

To propose the Vague Spatial Bitmap Index (VSB-index), some design choices were made as follows, mainly based on the results of the experimental evaluation described in Section 6.2, conducted for the DBMS and indices for SDWs. First, the support for regions with broad boundaries is tackled because this type of vague region impaired both storage requirements and the performance to process queries in vague SDW. Second, a multi-step resolution of the vague spatial predicate is prioritized, since such resolution caused increasing overheads on query response time. To comply with this purpose, a progressive approximation called MIP is introduced. Third, spatial range queries were chosen as the vague spatial predicates to be supported by the VSB-index, initially, to satisfy the requirements of the HLB case study. Fourth, since queries issued over a vague SDW has also conventional predicates, aggregation and sorting, these are processed by bitmap join indices that are commonly used in DWs.

For the sake of simplicity, the terms regions with broad boundaries and vague regions are used interchangeably in the following sections, which detail the VSB-index. Section 6.3.1 introduces the progressive approximation MIP. Section 6.3.2 describes the data structure of the VSB-index. Section 6.3.3 addresses the building operation of the VSB-index. Sections 6.3.4 and 6.3.5 focus on processing queries with the VSB-index according to different vague spatial predicates.

6.3.1 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 $x = 5$. The MIP was designed to be applied specially on regions with broad boundaries and to improve the resolution of vague spatial predicates. Figure 6.11a shows a vague region composed of the dark grey certitude and the light grey dubiety, and a MIP5 built on the certitude with a black contour. Note that the MIP5 built on the certitude is also a subset of the vague region. Figure 6.11b illustrates the MIP5 on the certitude and a MBR built on the vague region. The latter is identical to the MBR built on the outer boundary of the dubiety.

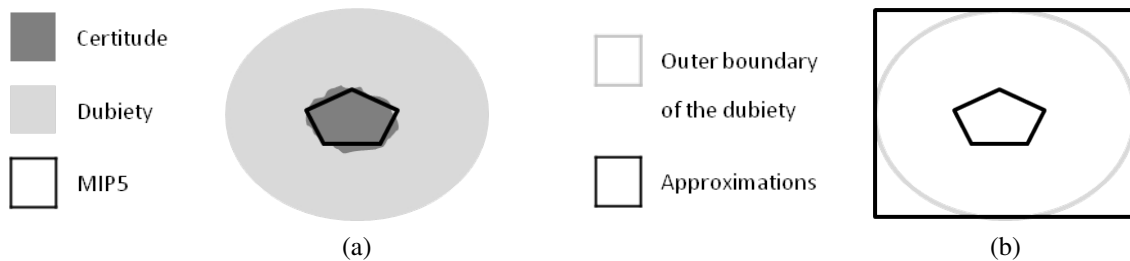


Figure 6.11: A vague region and its approximations: (a) The MIP5 on the certitude. (b) The MBR on the vague region and the MIP5 on the certitude.

6.3.2 Data Structure

The data structure of the VSB-index encompasses an array and a bitmap join index, as follows. The array has entries of the type *vrbitvector* (vague region bit-vector) comprising:

- one key value;
- one mandatory conservative approximation O_{\supseteq} of the vague region;
- one optional progressive approximation O_{\subseteq} of the vague region;
- one optional conservative approximation C_{\supseteq} of the certitude; and
- one optional progressive approximation C_{\subseteq} of the certitude.

The conservative approximation O_{\supseteq} is mandatory and enables processing queries. The other approximations are optional and allow flexible data structures and query processing algorithms, as described in Section 6.3.4. The aforementioned notation considers the conservative approximation \supseteq as a superset and the progressive approximation \subseteq as a subset. Also, the vague

region O is the vague spatial object, while C is its certitude. The feasible configurations for the VSB-index are, therefore: $O_{\supseteq}O_{\subseteq}C_{\supseteq}C_{\subseteq}$, $O_{\supseteq}O_{\subseteq}C_{\supseteq}$, $O_{\supseteq}O_{\subseteq}C_{\subseteq}$, $O_{\supseteq}O_{\subseteq}$, $O_{\supseteq}C_{\supseteq}C_{\subseteq}$, $O_{\supseteq}C_{\supseteq}$, $O_{\supseteq}C_{\subseteq}$ and O_{\supseteq} . When approximations are selected and assigned, they replace the corresponding symbols. For instance, the configuration $O_{MBR}C_{MIP5}$ holds an MBR on the vague region as conservative approximation and a MIP5 on the certitude as progressive approximation.

In addition to the array, the VSB-index also comprises a bitmap join index created for the key values. Therefore, each entry of the array has a corresponding bit-vector in the bitmap join index. Every bit-vector indicates the rows in the fact table where the key value of an array entry occurs.

6.3.3 Building Operation

The building operation of the VSB-index implements the array described in Section 6.3.2 as a sequential file stored in secondary memory. The size in bytes of an entry is $s = \text{sizeof}(int) + \text{sizeof}(O_{\supseteq}) + \text{sizeof}(O_{\subseteq}) + \text{sizeof}(C_{\supseteq}) + \text{sizeof}(C_{\subseteq})$. Then, each disk page with l bytes maintains $L = l \text{ DIV } s$ entries. Some unused bytes are left at the end of each disk page to avoid the fragmentation of an entry in two disk pages, and thus to prevent two disk accesses to obtain a single entry. These unused bytes are $U = c \text{ MOD } L$ bytes, where c is the cardinality of the indexed column. A header disk page stores metadata, such as the cardinality of the indexed column. A total of $A = 1 + c \text{ DIV } L + y$ disk pages are required to store the array of the VSB-index implemented as a sequential file. Furthermore, A disk accesses are required to build the sequential file. If $c \text{ MOD } L = 0$ then $y = 0$, otherwise $y = 1$.

Example 6.3.1. Table 6.1 lists all configurations of the VSB-index and the corresponding values of s , L and A considering the MBR as conservative approximation, MIP5 as progressive approximation, $l = 8192$ bytes, for both $c = 129$ and $c = 302,357$.

Table 6.1: Entry size in bytes (s), number of entries per disk page (L) and number of disk pages to store the index file (A) considering different cardinalities (c).

Configuration	s	L	$A (c = 129)$	$A (c = 302,357)$
$O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$	228	35	5	8,640
$O_{MBR}O_{MIP5}C_{MBR}$	148	55	4	5,499
$O_{MBR}O_{MIP5}C_{MIP5}$	196	41	5	7,376
$O_{MBR}O_{MIP5}$	116	70	3	4,321
$O_{MBR}C_{MBR}C_{MIP5}$	148	55	4	5,499
$O_{MBR}C_{MBR}$	68	120	3	2,521
$O_{MBR}C_{MIP5}$	116	70	3	4,321
O_{MBR}	36	227	2	1,333

Algorithm 1 details the building operation of the VSB-index. First, considering there is a connection with the DBMS, a SQL query selects the primary key and the geometry columns of the vague spatial attribute, sorting the results in ascending order based on the primary key (line 1). Then, the sequential file of the VSB-index is opened (line 3). A loop iterates through every row of the result set (lines 4-15). For each row, the key value and the approximations of the corresponding vague region are copied to one entry of an array in the main memory, whose type is *vrbivector* (lines 5-12). The functions *build_O_⊇* and *build_O_⊆* build the conservative approximation and the progressive approximation of the vague region (lines 7-8). The functions *build_C_⊇* and *build_C_⊆* build the conservative approximation and the progressive approximation of the certitude of the vague region (lines 9-10). Further, when the array becomes full (line 13), it is written to a disk page of the sequential file (line 14). Some unused bytes are left in the disk page (line 15). The sequential file is then closed (line 16). Finally, and a bitmap join index is built using the key values sorted in ascending order as input (lines 17-18). As the entries of the VSB-index remain sorted by the primary key values, *idx[i]* refers to the *i*-th bit-vector of the bitmap join index.

Functions whose implementation are not detailed in Algorithm 1 depend on the chosen approximations. An overview of them is provided as follows. The implementation of the functions *build_O_⊇* and *build_C_⊇*, which build conservative approximations, can reuse existing functions available in the DBMS. To build a MBR, *ST_Envelope* and *ST_XMin*, *ST_YMin*, *ST_XMax* and *ST_YMax* (OBE; HSU, 2015) can be used. Conversely, the implementation of the functions *build_O_⊆* and *build_C_⊆*, which build progressive approximations, can reuse functions of libraries for computational geometry, e.g. CGAL, Computational Geometry Algorithms Library³ and the method *CGAL::maximum_area_inscribed_k_gon_2* to build the MIP. Obviously, if the configuration of the VSB-index being built does not hold a given approximation, the corresponding function is skipped. For instance, functions *build_O_⊆* and *build_C_⊇* are skipped if the configuration is *O_⊇C_⊆*.

Finally, the implementation of the function *buildBitmapJoinIndex* can reuse the statement *CREATE BITMAP INDEX* and specify a join in the *FROM* and *WHERE* clauses (LANE; POTINENI, 2014). Alternatively, an specific software that implements the bitmap index can be reused (WU et al., 2009; CARNIEL; SIQUEIRA, 2012).

Example 6.3.2. The following call to the procedure *buildBitmapJoinIndex* builds a VSB-index for infected regions of the table *InfectedRegion* depicted in Figure 5.19, considering that 302,357 vague regions are stored. The first argument is the index file name. Regarding the second

³<http://www.cgal.org>

Algorithm 1: BuildVSBIndex(idx, L, U, queryVSAttribute, queryPK)

Input: idx is the sequential file of the VSB-index.

L is the maximum number of index entries that a disk page can hold.

U is the unused amount of bytes in a disk page.

queryVSAttribute is a SQL query that retrieves columns of the vague spatial attribute.

queryPK is a SQL query that retrieves primary key values.

Data: record, i, array

Result: A VSB-index.

```

1 record ← executeDBMS(queryVSAttribute)
2 i ← 0
3 open (idx)
4 while record ≠ eof do
5     while i ≤ L and record ≠ eof do
6         array[i].pk ← record.pk
7         array[i].O⊇ ← buildO⊇(record.o)
8         array[i].O⊆ ← buildO⊆(record.o)
9         array[i].C⊇ ← buildC⊇(record.c)
10        array[i].C⊆ ← buildC⊆(record.c)
11        i ← i + 1
12        record.next()
13    i ← 0
14    write (array, idx)
15    forward (U, idx)
16 close (idx)
17 record ← executeDBMS(queryPK)
18 buildBitmapJoinIndex(record)

```

argument, Table 6.1 indicates that configuration $O_{MBR}O_{MIP5}$ of the VSB-index has entries that require 116 bytes of storage space. Then, each disk page with 8192 bytes maintains $L = 8192 \text{ DIV } 116 = 70$ entries. Concerning the third argument, $U = 302,357 \text{ MOD } 70 = 72$ bytes. The fourth argument is a SQL query that selects key values, the merged geometries and geometries of certitude elements from the infections. The fifth argument is a SQL query that selects key values of infections.

```

BuildVSBIndex('vsbindex_infection.bin',
70,
72,
'SELECT RegionId as pk,
        InfectedRegion_MergedElementsGeo AS o,
        VS_CertitudeGeo(InfectedRegion_ElementsGeo,
                        InfectedRegion_ElementsMval) AS c
FROM InfectedRegion
ORDER BY RegionId',
'SELECT RegionId as pk FROM InfectedRegion ORDER BY RegionId'
)

```

over the vague SDW whose schema is depicted in .

6.3.4 Processing Queries containing Spatial Range Queries

The VSB-index has algorithms to process queries issued to vague SDWs that have spatial range queries as the vague spatial predicate. It resolves IRQ_{object} , CRQ_{object} , $IRQ_{certitude}$, $CRQ_{certitude}$, $IRQ_{dubiety}$, and $CRQ_{dubiety}$ spatial range queries through a multi-step resolution of the vague spatial predicate, a key matching and the subsequent access to efficient bitmap join indices. The multi-step resolution of the vague spatial predicate comprises a filter step and a refinement step. The filter step is implemented as a sequential scan on the sequential file, which requires A disk accesses, as shown in Table 6.1, depending on the available conservative and progressive approximations. The filter step produces a set of key values of the candidates that satisfy the vague spatial predicate. Such set is used in the refinement step to fetch the corresponding geometries and verify which of them are answers of the vague spatial predicate. The key values of the answers compose another set. Once every key value matches one vague spatial object, the set of answers is used to create a conventional predicate that replaces the corresponding vague spatial predicate of the query. Finally, the rewritten query is processed by bitmap join indices.

Algorithm 2 details how the VSB-index processes a query. First, the sequential file is opened and read to fetch the number of index entries per disk page and which configuration of the VSB-index is implemented (lines 1-5). Then, the filter step is performed (line 6). If there are candidates produced by the filter step, the refinement step is executed accessing the DBMS and providing the key values of the candidates (lines 7-8). Note that answers previously fetched in the filter step are not overwritten. If answers were retrieved by the filter or the refinement steps, then a string is composed with such answers (lines 9-10). The string replaces the original vague spatial predicate of the query (line 11). The rewritten query is, then, processed by accessing bitmap join indices (line 12). Finally, the result set of the query is returned (line 13).

Example 6.3.3. The following call to the procedure `ProcessSpatialRangeQuery` processes the query described in Example 5.9.4 over the vague SDW whose schema is depicted in Figure 5.19. Note that `vsbindex.bin` is the index file, `query594` is the SQL query described in Example 5.9.4, `IRQobject` indicates the vague spatial predicate, `w` is the spatial query window, the `SELECT` statement describes the SQL query to perform the refinement step, and `InfectedRegionId` is the column with the primary key to enable key matching.

```
ProcessSpatialRangeQuery (
    'vsbindex.bin',
```

Algorithm 2: ProcessSpatialRangeQuery(idx, query, vsPredicate, window, queryRefinement, pkColumn)

Input: idx is the sequential file of the VSB-index

query is the query issued over the vague SDW

vsPredicate is the vague spatial predicate

window is the spatial query window

queryRefinement is the SQL query for the refinement step

pkColumn is the column that has a primary key and is used for keymatching

Data: header, L, vsbIndexConfig, setAnswers, setCandidates, conventionalPredicate, resultSet

Result: The result set of the query.

```

1 open (idx)
2 read(idx, header)
3 close(idx)
4 L ← getL(header)
5 vsbIndexConfig ← getConfig(header)
6 executeFilterStep(vsbIndexConfig, L, vsPredicate, window, setCandidates, setAnswers,
  idx)
7 if setCandidates is not empty then
8   setAnswers ← setAnswers + executeRefinementStep(queryRefinement,
  setCandidates)
9 if setAnswers is not empty then
10  conventionalPredicate ← 'AND' + pkColumn + 'IN (' + toString(setAnswers) + ')'
11  replaceVSPredicate (query, conventionalPredicate)
12  resultSet ← executeBitmapJoinIndex(query)
13 return resultSet

```

```

query594 ,
'IRQobject' ,
w,
'SELECT InfectionPK FROM Infection WHERE InfectionPK IN (%)
AND ST_Intersects (InfectedRegion_MergedElementsGeo , w) ',
'InfectedRegionId '
)

```

Algorithm 2 does not specify how the filter step is executed (line 6). In fact, the routine *executeFilterStep* has a single decision structure (e.g. IF) that, based on the VSB-index configuration *vsbindexConfig* and the type of vague spatial predicate *vsPredicate*, calls one procedure that processes the filter step. Only one call is necessary independently from configuration and vague spatial predicate. These procedures are f1, f2, f3 and f4. The procedures f1 and f2 are described in Sections 6.3.4.1 and 6.3.4.2, respectively, while the procedures f3 and f4 are described in Appendix B. Particularities of querying the dubiety (*IRQ_{dubiety}* and *CRQ_{dubiety}*) are described in Section 6.3.4.3. There are 48 possible calls to procedures to resolve the filter step using one of the 8 configurations of the VSB-index. Calls to procedures f1 and f2 comprise 44

of them, as detailed in Section 6.3.4.4.

After the filter step, the refinement step is performed by the function *executeRefinementStep* in Algorithm 2 (line 8) if there are candidates. Basically, *executeRefinementStep* requires a connection to the DBMS to issue a query that checks which of the candidates, i.e. vague regions represented by their key values in *setCandidates*, are answers of the vague spatial predicate. After the refinement step, the query is rewritten and submitted for execution using bitmap join indices with the function *executeBitmapJoinIndex*, whose implementation can reuse bitmap join indices of the DBMS (LANE; POTINENI, 2014) or bitmap join indices implemented by specific software (WU et al., 2009).

6.3.4.1 Filtering with a Conservative Approximation

Algorithm 3 describes the procedure *f1*, which performs the filter step using strictly one conservative approximation. In detail, the procedure *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 C_{\supseteq} , depending on the parameter passed, and is tested against the spatial query window w (line 6). If the topological 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.

Algorithm 3: *f1*(R , window, conservative, set, idx, L)

Input: R is a topological relationship
 window is a spatial query window
 conservative is a conservative approximation
 set is a set of answers or a set of candidates
 idx is the sequential file of the VSB-index
 L the number of VSB-index' entries that a disk page can hold.
Data: page, array
Result: A set of answers or candidates of the vague spatial predicate.

```

1 open (idx)
2 while not eof(idx) do
3   read (idx, page)
4   copy (page, array)
5   for  $i \leftarrow 0$  to  $L$  do
6     if  $R(\text{get}(\text{array}[i], \text{conservative}), \text{window})$  then
7       append(set, array[i].pk)
8 close(idx)

```

Example 6.3.4. The configuration O_{\supseteq} implemented as O_{MBR} processes a CRQ_{object} by calling the procedure f1 with:

f1(within, w, O_{\supseteq} , setAnswers, idxFile, L).

Note that the MBR of the vague region and the set of answers *setAnswers* are passed as parameters. Consider that the MBR of the vague region is within the spatial query window w, as shown in Figure 6.12a. Then, the corresponding key value is added to the set of answers. It is noteworthy that the configuration O_{\supseteq} identifies answers of CRQ_{object} already in the filter step and produces a set of answers, but does not produce a set of candidates. As a result, the costly refinement step is skipped.

Example 6.3.5. The configuration $O_{\supseteq}C_{\supseteq}$ implemented as $O_{MBR}C_{MBR}$ processes a $CRQ_{certitude}$ by calling the procedure f1 with:

f1 (within, w, C_{\supseteq} , setAnswers, idxFile, L).

The MBR of the certitude and the set of answers *setAnswers* are passed as parameters. Consider that the MBR of the certitude is within the spatial query window w, as shown in Figure 6.12b. Then, the corresponding key value is added to the set of answers. The configuration $O_{\supseteq}C_{\supseteq}$, therefore, skips the refinement step to process a $CRQ_{certitude}$.

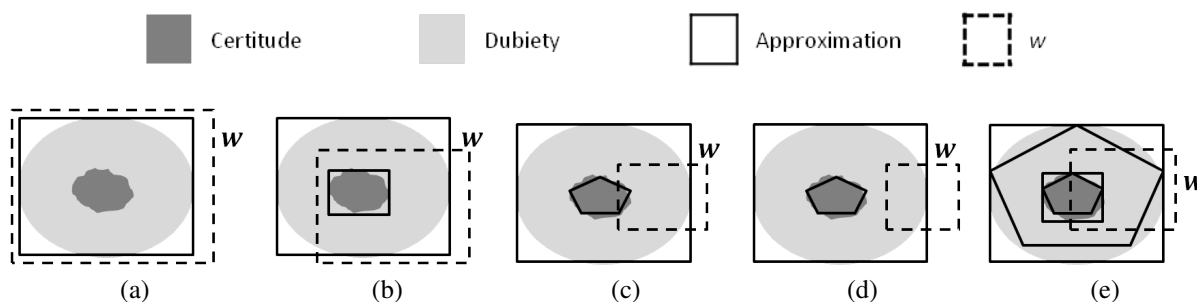


Figure 6.12: A vague region in tones of gray, its approximations with black continuous contour and spatial range queries with dashed rectangles: (a) A CRQ_{object} against O_{MBR} . (b) A $CRQ_{certitude}$ against $O_{MBR}C_{MBR}$. (c) An IRQ_{object} against $O_{MBR}C_{MIP5}$. (d) Another IRQ_{object} against $O_{MBR}C_{MIP5}$. (e) An $IRQ_{certitude}$ against $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$.

6.3.4.2 Filtering with a Conservative Approximation and a Progressive Approximation

Algorithm 4 describes the procedure f2, which performs the filter step using both a conservative approximation and a progressive approximation. To identify answers already in the filter step, the procedure f2 performs a sequential scan that tests the conservative approximation and subsequently tests the progressive approximation. For each entry (lines 5-10), if the topological relationship is satisfied for both the conservative and progressive approximations, the entry is considered as 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 topological relationship, the entry is considered as a candidate and its primary key value is stored in the set of candidates (line 10).

Algorithm 4: $f2(R, \text{window}, \text{conservative}, \text{progressive}, \text{setCandidates}, \text{setAnswers}, \text{idx}, L)$

Input: R is a topological relationship
 window is a spatial query window
 conservative is a conservative approximation
 progressive is a progressive approximation
 setCandidates is a set of candidates
 setAnswers is a set of answers
 idx is the sequential file of the VSB-index
 L the number of VSB-index' entries that a disk page can hold.

Data: page, array

Result: The set of answers and the set of candidates of the vague 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(get(array[i], conservative), window) then
7       if R(get(array[i], progressive), window) then
8         append(setAnswers, array[i].pk)
9       else
10        append(setCandidates, array[i].pk)
11 close(idx)

```

Example 6.3.6. The configuration $O_{\supseteq}C_{\subseteq}$ implemented as $O_{MBRC_{MIP5}}$ processes an IRQ_{object} by calling the procedure $f2$ as:

$f2$ (intersects, window, O_{\supseteq} , C_{\subseteq} , setCandidates, setAnswers, idxFile, L).

The MBR of the vague region and the MIP5 of the certitude are passed as parameters. Consider that the MBR of the vague region and the MIP5 of the certitude intersect the spatial query window w , as shown in Figure 6.12c. Then, the corresponding key value is added to the set of answers. Conversely, consider that the MBR of the vague region intersects the spatial query window w and the MIP5 of the certitude does not intersect w , as shown in Figure 6.12d. Then, the corresponding key value is added to the set of candidates. A subsequent refinement step to check whether the candidates satisfy IRQ_{object} is mandatory.

In Example 6.3.6, the cost of the refinement step might be decreased since answers of the vague spatial predicate are already identified in the filter step. Although the progressive approximation of the certitude can aid to identify answers of IRQ_{object} , the benefit might not be

significant if the area of the certitude is relatively small, as shown in Figure 6.12d.

Example 6.3.7. The configuration $O_{\supseteq}O_{\subseteq}C_{\supseteq}C_{\subseteq}$ implemented as $O_{MBR}O_{MIP5}C_{MBR}C_{MIP}$ processes an $IRQ_{certitude}$ by calling the procedure f2 as:

f2 (intersects, window, C_{\supseteq} , C_{\subseteq} , setCandidates, setAnswers, idxFile, L).

The MBR and the MIP5 of the certitude are passed as parameters. Consider that both the MBR and the MIP5 of the certitude intersect the spatial query window w , as shown in Figure 6.12e. Then, the corresponding key value is added to the set of answers. A subsequent refinement step is still mandatory.

In Example 6.3.7, the cost of the refinement step might be decreased due to the identification of answers of the vague spatial predicate already in the filter step.

6.3.4.3 Particularities of Querying the Dubiety

The resolutions of CRQ_{object} and $CRQ_{dubiety}$ are the same due to the following reasons. It is well-known that a conservative approximation is a superset built on the boundary of an object. If the object is a vague region, then its “boundary” is equal to the outer boundary of its dubiety. As a result, a conservative approximation of a region with broad boundaries is equal to a conservative approximation of its dubiety, as exemplified in Figure 6.13a using the MBR. If the conservative approximation of an object is within a spatial query window, then it is feasible to conclude that the object is also within the spatial query window. Consequently, if a conservative approximation of the dubiety is within a spatial query window, then it is feasible to conclude that the dubiety is also within the spatial query window. Since every configuration of the VSB-index holds a conservative approximation of the vague region, which is also a conservative approximation of the dubiety, CRQ_{object} and $CRQ_{dubiety}$ have the same resolution.

The $IRQ_{dubiety}$ is resolved as an IRQ_{object} in the majority of cases. The only exception regards to a spatial query window that is within the certitude and does not touch the boundary of the certitude, as shown in Figure 6.13b. In other words, the spatial query window is within the hole of the dubiety and does not touch the inner boundary of the dubiety. To avoid the retrieval of false answers, an additional refinement step must assess every vague region in the set of answers to check whether the dubiety and the spatial query window are disjoint. Those vague regions whose dubiety is disjoint from the spatial query window are, therefore, removed from the set of answers of $IRQ_{dubiety}$.

The cost of such additional refinement step comprises the extraction of the vague regions’ dubiety and the subsequent verification of disjointness. The cost might be low if the spatial

query window is within the certitude of a single or a few vague regions. Nevertheless, if the vague regions overlap each other, then the spatial query window might be in the hole of the dubiety of several vague regions, as shown in Figure 6.13c. As a result, the cost of the additional refinement step may add a more significant overhead to the query response time.

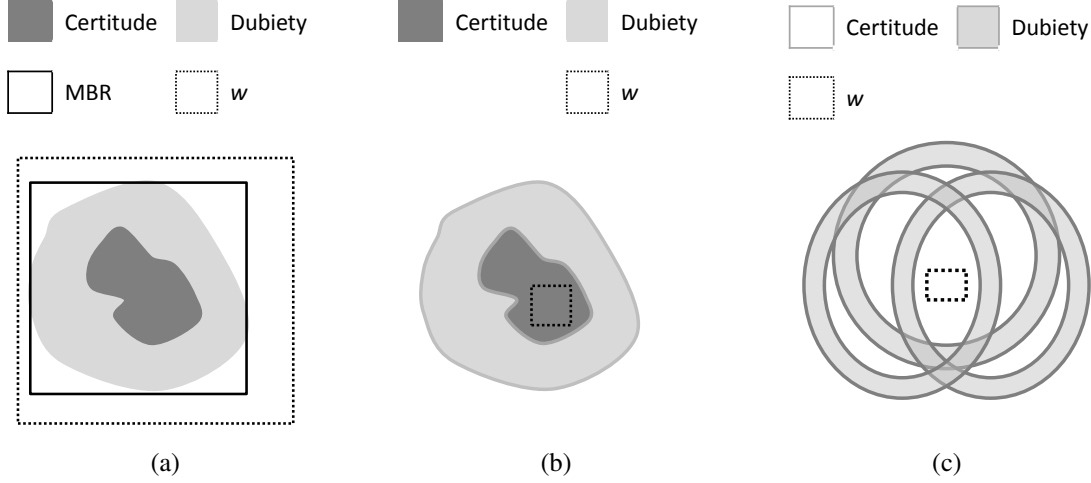


Figure 6.13: A vague region, its elements and vague spatial predicates regarding the dubiety: (a) A $CRQ_{dubiety}$. (b) An $IRQ_{dubiety}$. (c) A query window into the holes.

6.3.4.4 Calling the Procedures

The decision about which procedure the VSB-index calls to perform the filter step of the vague spatial predicate resolution depends on the available approximations and the type of vague spatial predicate issued. For each configuration of the VSB-index, the calls made to procedures f1 and f2 in order to resolve IRQ_{object} , $IRQ_{certitude}$, and $IRQ_{dubiety}$ are:

$O_{\supseteq} O_{\subseteq} C_{\supseteq} C_{\subseteq}$ IRQ_{object} : f2 (intersects, w, O_{\supseteq} , O_{\subseteq} , setCandidates, setAnswers, idx, L)

$IRQ_{dubiety}$: f2 (intersects, w, O_{\supseteq} , O_{\subseteq} , setCandidates, setAnswers, idx, L)

$IRQ_{certitude}$: f2 (intersects, w, C_{\supseteq} , C_{\subseteq} , setCandidates, setAnswers, idx, L)

$O_{\supseteq} O_{\subseteq} C_{\supseteq}$ IRQ_{object} : f2 (intersects, w, O_{\supseteq} , O_{\subseteq} , setCandidates, setAnswers, idx, L)

$IRQ_{dubiety}$: f2 (intersects, w, O_{\supseteq} , O_{\subseteq} , setCandidates, setAnswers, idx, L)

$IRQ_{certitude}$: f1(intersects, w, C_{\supseteq} , setCandidates, idx, L)

$O_{\supseteq} O_{\subseteq} C_{\subseteq}$ IRQ_{object} : f2 (intersects, w, O_{\supseteq} , O_{\subseteq} , setCandidates, setAnswers, idx, L)

$IRQ_{dubiety}$: f2 (intersects, w, O_{\supseteq} , O_{\subseteq} , setCandidates, setAnswers, idx, L)

$IRQ_{certitude}$: f2 (intersects, w, O_{\supseteq} , C_{\subseteq} , setCandidates, setAnswers, idx, L)

$O_{\supseteq} O_{\subseteq}$ IRQ_{object} : f2 (intersects, w, O_{\supseteq} , O_{\subseteq} , setCandidates, setAnswers, idx, L)

$IRQ_{dubiety}$: f2 (intersects, w, O_{\supseteq} , O_{\subseteq} , setCandidates, setAnswers, idx, L)

$IRQ_{certitude}$: f1(intersects, w, O_{\supseteq} , setCandidates, idx, L)

$O_{\supseteq}C_{\supseteq}C_{\subseteq}$ IRQ_{object} : f2 (intersects, w, O_{\supseteq} , C_{\subseteq} , setCandidates, setAnswers, idx, L)
 $IRQ_{dubiety}$: f2 (intersects, w, O_{\supseteq} , C_{\subseteq} , setCandidates, setAnswers, idx, L)
 $IRQ_{certitude}$: f2 (intersects, w, C_{\supseteq} , C_{\subseteq} , setCandidates, setAnswers, idx, L)

$O_{\supseteq}C_{\supseteq}$ IRQ_{object} : f1(intersects, w, O_{\supseteq} , setCandidates, idx, L)
 $IRQ_{dubiety}$: f1(intersects, w, O_{\supseteq} , setCandidates, idx, L)
 $IRQ_{certitude}$: f1(intersects, w, C_{\supseteq} , setCandidates, idx, L)

$O_{\supseteq}C_{\subseteq}$ IRQ_{object} : f2 (intersects, w, O_{\supseteq} , C_{\subseteq} , setCandidates, setAnswers, idx, L)
 $IRQ_{dubiety}$: f2 (intersects, w, O_{\supseteq} , C_{\subseteq} , setCandidates, setAnswers, idx, L)
 $IRQ_{certitude}$: f2 (intersects, w, O_{\supseteq} , C_{\subseteq} , setCandidates, setAnswers, idx, L)

O_{\supseteq} IRQ_{object} : f1(intersects, w, O_{\supseteq} , setCandidates, idx, L)
 $IRQ_{dubiety}$: f1(intersects, w, O_{\supseteq} , setCandidates, idx, L)
 $IRQ_{certitude}$: f1(intersects, w, O_{\supseteq} , setCandidates, idx, L)

Furthermore, for each configuration of the VSB-index, the calls made to procedures f1, f2, f3 and f4 in order to resolve CRQ_{object} , $CRQ_{certitude}$, and $CRQ_{dubiety}$ are:

$O_{\supseteq}O_{\subseteq}C_{\supseteq}C_{\subseteq}$ CRQ_{object} : f1 (within, w, O_{\supseteq} , setAnswers, idx, L)
 $CRQ_{dubiety}$: f1 (within, w, O_{\supseteq} , setAnswers, idx, L)
 $CRQ_{certitude}$: f1 (within, w, C_{\supseteq} , setAnswers, idx, L)

$O_{\supseteq}O_{\subseteq}C_{\supseteq}$ CRQ_{object} : f1 (within, w, O_{\supseteq} , setAnswers, idx, L)
 $CRQ_{dubiety}$: f1 (within, w, O_{\supseteq} , setAnswers, idx, L)
 $CRQ_{certitude}$: f1 (within, w, C_{\supseteq} , setAnswers, idx, L)

$O_{\supseteq}O_{\subseteq}C_{\subseteq}$ CRQ_{object} : f1 (within, w, O_{\supseteq} , setAnswers, idx, L)
 $CRQ_{dubiety}$: f1 (within, w, O_{\supseteq} , setAnswers, idx, L)
 $CRQ_{certitude}$: f4(R, w, O_{\supseteq} , C_{\subseteq} , setAnswers, setCandidates, idx, L)

$O_{\supseteq}O_{\subseteq}$ CRQ_{object} : f1 (within, w, O_{\supseteq} , setAnswers, idx, L)
 $CRQ_{dubiety}$: f1 (within, w, O_{\supseteq} , setAnswers, idx, L)
 $CRQ_{certitude}$: f3(R, w, O_{\supseteq} , setAnswers, setCandidates, idx, L)

$O_{\supseteq}C_{\supseteq}C_{\subseteq}$ CRQ_{object} : f1 (within, w, O_{\supseteq} , setAnswers, idx, L)
 $CRQ_{dubiety}$: f1 (within, w, O_{\supseteq} , setAnswers, idx, L)
 $CRQ_{certitude}$: f4(R, w, O_{\supseteq} , C_{\subseteq} , setAnswers, setCandidates, idx, L)

$O_{\supseteq}C_{\supseteq}$ CRQ_{object} : f1 (within, w, O_{\supseteq} , setAnswers, idx, L)
 $CRQ_{dubiety}$: f1 (within, w, O_{\supseteq} , setAnswers, idx, L)
 $CRQ_{certitude}$: f1 (within, w, C_{\supseteq} , setAnswers, idx, L)

$O_{\supseteq}C_{\subseteq}$ CRQ_{object} : f1 (within, w, O_{\supseteq} , setAnswers, idx, L)
 CRQ_{dubity} : f1 (within, w, O_{\supseteq} , setAnswers, idx, L)
 $CRQ_{certitude}$: f4(R, w, O_{\supseteq} , C_{\subseteq} , setAnswers, setCandidates, idx, L)

O_{\supseteq} CRQ_{object} : f1 (within, w, O_{\supseteq} , setAnswers, idx, L)
 CRQ_{dubity} : f1 (within, w, O_{\supseteq} , setAnswers, idx, L)
 $CRQ_{certitude}$: f3(R, w, O_{\supseteq} , setAnswers, setCandidates, idx, L)

6.3.5 Processing Queries Containing a Vague Spatial Range Query

The vague spatial range query, or simply *VSRQ*, was introduced by the VSCube conceptual model in Section 4.5.2. A *VSRQ* uses a pair of concentric spatial query windows *inner* and *outer*, where the former is the innermost and the latter is the outermost, as well as evaluates the spatial predicate *containment* twice and the spatial predicate *intersects* once. In Section 5.9.2, the resolution of the *VSRQ* creates two sub-queries. One tests *containment* against *inner*, while the other tests *intersects and not containment* against *outer*. Finally, an union is then applied to the result sets of the sub-queries.

The VSB-index also resolves queries containing a *VSRQ* and separates the two sub-queries. It firstly performs a multi-step resolution of the spatial predicates, then executes a key matching and finally processes efficient bitmap join indices. Compared to the resolution of spatial range queries described in Section 6.3.4, the main difference is the filter step. Instead of testing each entry against a single predicate, both *containment* and *intersects* are tested. The refinement step is only executed for the candidates of the predicate *intersects*. The key matching rewrites the pair of sub-queries, which are separately submitted for processing using bitmap join indices. Finally, the result sets of the sub-queries are merged.

Algorithm 5 details the aforementioned procedure. The sequential file is opened and read to fetch the number of index entries per disk page and which configuration of the VSB-index is implemented (lines 1-5). Then, the filter step is performed (line 6). If there are candidates produced by the filter step, the refinement step is executed accessing the DBMS and providing the key values of the candidates (lines 7-8). Note that answers previously identified in the filter step are kept and answers found in the refinement step are appended. If there are answers for the spatial predicate *containment* (line 9), a string is composed with such answers (line 10). The string replaces the original spatial predicate of the sub-query (line 11) and the rewritten sub-query is processed by accessing bitmap join indices (line 12). If there are answers for the spatial predicate *intersects*, the same steps are performed for its sub-query (lines 13-16). Finally, the result set of the query is the union of the result sets gathered for the sub-queries (line 17).

Algorithm 5: ProcessVagueSpatialRangeQuery(idx, vsrqType, windowIn, subQueryIn, windowOut, subQueryOut, refinementOut, pkColumn)

Input: idx is the sequential file of the VSB-index

vsrqType is the type of VSRQ

windowIn is the inner spatial query window

subQueryIn is the sub-query regarding windowIn

windowOut is the outer spatial query window

subQueryOut is the sub-query regarding windowOut

refinementOut is the SQL query for the refinement step involving windowOut

pkColumn is the column that has a primary key and is used for keymatching

Data: header, L, vsbIndexConfig, setAnswersIn, setCandidatesIn, setAnswersOut, setCandidatesOut, conventionalPredicate, resultSetIn, resultSetOut

Result: The result set of the query.

```

1 open (idx)
2 read(idx, header)
3 close(idx)
4 L ← getL(header)
5 vsbIndexConfig ← getConfig(header)
6 executeFilterStepVSRQ(vsbIndexConfig, L, vsrqType, windowIn, setAnswersIn,
  windowOut, setCandidatesOut, setAnswersOut, idx)
7 if setCandidatesOut is not empty then
8   setAnswersOut ← setAnswersOut + executeRefinementStep(refinementOut,
  setCandidatesOut)
9 if setAnswersIn is not empty then
10  conventionalPredicate ← ‘AND ’+pkColumn+‘ IN (’ + toString(setAnswersIn)+‘)’
11  replacePredicate(subQueryIn, conventionalPredicate)
12  resultSetIn ← executeBitmapJoinIndex(subQueryIn)
13 if setAnswersOut is not empty then
14  conventionalPredicate ← ‘AND ’+pkColumn+‘ IN (’ + toString(setAnswersOut)+‘)’
15  replacePredicate(subQueryOut, conventionalPredicate)
16  resultSetOut ← executeBitmapJoinIndex(subQueryOut)
17 return resultSetIn union resultSetOut

```

Algorithm 5 does not specify how the filter step is executed (line 6). In fact, the routine *executeFilterStepVSRQ* has a single decision structure (e.g. IF) that, based on the VSB-index configuration in *vsbindexConfig* and the type of *VSRQ* in *vsrqType*, calls one procedure that processes the filter step. The type of vague spatial predicate is one of the following: *VSRQ_{object}*, *VSRQ_{certitude}* and *VSRQ_{dubity}*. Only one call is necessary independently from configuration and vague spatial predicate. Two of these procedures are *fvsrq1* and *fvsrq2*, detailed in Sections 6.3.5.1 and 6.3.5.2, respectively. The calls to procedures *fvsrq1* and *fvsrq2* are detailed in Section 6.3.5.3. After the filter step, the refinement step is performed and bitmap join indices are accessed analogously to Section 6.3.4.

6.3.5.1 Filtering with a Conservative Approximation

The procedure `fvsrq1` detailed in Algorithm 6 processes the filter step for configurations of the VSB-index that do not have a progressive approximation, i.e. O_{\supseteq} and $O_{\supseteq}C_{\supseteq}$. In detail, the procedure `fvsrq1` performs a sequential scan over the sequential file (lines 2-15) that retrieves each disk page (line 3) and temporally stores it in the main memory (line 4). Another loop tests the approximations of each entry against both the inner spatial query window and the outer spatial query window, as follows (lines 5-15).

First, if the conservative approximation is within the inner spatial query window, then the corresponding key value is added to the set of answers regarding the inner spatial query window (lines 7-8). Also, the loop is skipped to the next iteration (lines 9-10). Considering that the first test has not yielded true, then a second test verifies whether the conservative approximation is within the outer spatial query window (line 11). If so, the corresponding key value is added to the set of answers regarding the outer spatial query window (line 11). If not, a third test checks whether the progressive approximation intersects the outer spatial query window (line 14). If they intersect, then the corresponding key value is added to the set of candidates regarding the outer spatial query window (line 15). Finally, the sequential file is closed (line 16).

Example 6.3.8. Suppose that a $VSRQ_{object}$ is issued using the spatial query windows i (inner) and o (outer) as shown in Figure 6.14a against the MBRs associated with the key values 1, 2 and 3 also shown in Figure 6.14a, which refer to a VSB-index O_{MBR} . Then, the following call to the procedure `fvsrq1` described in Algorithm 6 is made:

```
fvsrq1(i, o,  $O_{MBR}$ , setAnswersIn, setAnswersOut, idxFile, L).
```

The key value 1 is added to the set of answers of the inner spatial query window, `setAnswersIn`, since the corresponding MBR is within i (lines 7-10). The key value 2 is added to the set of answers of the outer spatial query window, `setAnswersOut`, since the corresponding MBR is not within i but is within o (lines 7, 11-12). Finally, the key value 3 is added to the set of candidates of the outer spatial query window, `setCandidatesOut`, since the corresponding MBR is not within i , is not within o , but intersects o (lines 7, 11, 13-15). Since `setCandidatesOut` has the key value 3, the geometry corresponding to the key value 3 is retrieved and processed, afterwards, in the refinement step.

In contrast with the procedures described in Section 6.3.4 that evaluate a single spatial predicate per entry, the procedure `fvsrq1` evaluates the predicates *containment* and *intersects* for each entry. A resolution of the predicate *containment* is mandatory and tests the conservative approximation against the inner spatial query window. Another resolution of the predicate

Algorithm 6: fvsrq1(windowIn, windowOut, conservative, setAnswersIn, setAnswersOut, setCandidatesOut, idx, L)

Input: windowIn is the inner spatial query window
 windowOut is the outer spatial query window
 conservative is a conservative approximation
 setAnswersIn is a set of answers regarding windowIn
 setAnswersOut is a set of answers regarding windowOut
 setCandidatesOut is a set of candidates regarding windowOut
 idx is the sequential file of the VSB-index
 L the number of VSB-index' entries that a disk page can hold.

Data: page, array, i

Result: A set of answers or candidates of the vague spatial predicate.

```

1 open (idx)
2 while not eof(idx) do
3   read (idx, page)
4   copy (page, array)
5   i ← 0
6   while i ≤ L do
7     if Within(get(array[i], conservative), windowIn) then
8       append(setAnswersIn, array[i].pk)
9       i ← i + 1
10    continue
11    if Within(get(array[i], conservative), windowOut) then
12      append(setAnswersOut, array[i].pk)
13    else
14      if Intersects(get(array[i], conservative), windowOut) then
15        append(setCandidatesOut, array[i].pk)
16 close(idx)

```

containment occurs only if the previous test yielded false. It tests the conservative approximation against the outer spatial query window, because *containment* is one subtype of *intersects* and if the test yields true, then the entry is considered an answer. The resolution of the predicate *intersects* occurs only if the previous test yielded false and tests the conservative approximation against the outer spatial query window. On the one hand, checking at least one and at most three spatial predicates per entry adds an overhead. On the other hand, it can reduce the set of candidates to be checked in the refinement step and, then, benefit the performance to process the query.

6.3.5.2 Filtering with a Conservative Approximation and a Progressive Approximation

The procedure `fvsrq2` detailed in Algorithm 7 processes the filter step for configurations of the VSB-index that have a progressive approximation. In detail, the procedure `fvsrq2` performs a sequential scan over the sequential file (lines 2-18) that retrieves each disk page (line 3) and temporally stores it in the main memory (line 4). Another loop tests the approximations of each entry against both the inner spatial query window and the outer spatial query window, as follows (lines 7-18).

First, if the conservative approximation is within the inner spatial query window, then the corresponding key value is added to the set of answers regarding the inner spatial query window (lines 7-8). Also, the loop is skipped to the next iteration (lines 9-10). Considering that the first test has not yielded true, then a second test verifies whether the conservative approximation is within the outer spatial query window (line 11). If so, the corresponding key value is added to the set of answers regarding the outer spatial query window (line 12). Otherwise, the key value is considered an answer or a candidate regarding the outer spatial query window, depending on the following conditions. To be an answer, both the conservative and the progressive approximations intersect the outer spatial query window (lines 14-16). To be a candidate, only the conservative approximation intersects the outer spatial query window (lines 14-15, 17-18). Finally, the sequential file is closed (line 19).

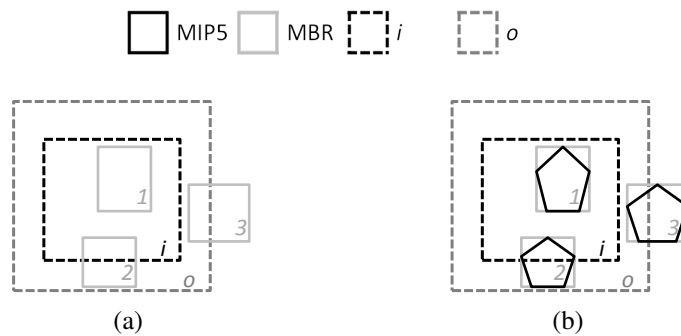


Figure 6.14: Vague spatial range queries. (a) Against MBRs. (b) Against MBRs and MIP5s.

Example 6.3.9. Suppose that a $VSRQ_{Object}$ is issued using the spatial query windows i (inner) and o (outer) as shown in Figure 6.14b against the MBRs associated with the key values 1, 2 and 3 also shown in Figure 6.14b, which refer to a VSB-index $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$. Then, the following call to the procedure `fvsrq2` described in Algorithm 7 is made: `fvsrq2(i, o, OMBR, OMIP5, setAnswersIn, setAnswersOut, idxFile, L)`. The key value 1 is added to the set of answers of the inner spatial query window, `setAnswersIn`, since the corresponding MBR is within i (lines 7-10). The key value 2 is added to the set of answers of the outer spatial query window, `setAnswersOut`, since the corresponding MBR is not within i but is within o (lines 7, 11-12).

Algorithm 7: fvsrq2(windowIn, windowOut, conservative, progressive, setAnswersIn, setAnswersOut, setCandidatesOut, idx, L)

Input: windowIn is the inner spatial query window
 windowOut is the outer spatial query window
 conservative is a conservative approximation
 progressive is a progressive approximation
 setAnswersIn is a set of answers regarding windowIn
 setAnswersOut is a set of answers regarding windowOut
 setCandidatesOut is a set of candidates regarding windowOut
 idx is the sequential file of the VSB-index
 L the number of VSB-index' entries that a disk page can hold.

Data: page, array, i

Result: A set of answers or candidates of the vague spatial predicate.

```

1 open (idx)
2 while not eof(idx) do
3   read (idx, page)
4   copy (page, array)
5   i ← 0
6   while i ≤ L do
7     if Within(get(array[i], conservative), windowIn) then
8       append(setAnswersIn, array[i].pk)
9       i ← i + 1
10    continue
11    if Within(get(array[i], conservative), windowOut) then
12      append(setAnswersOut, array[i].pk)
13    else
14      if Intersects(get(array[i], conservative), windowOut) then
15        if Intersects(get(array[i], progressive), windowOut) then
16          append(setAnswersOut, array[i].pk)
17        else
18          append(setCandidatesOut, array[i].pk)
19 close(idx)

```

Finally, the key value 3 is added to the set of answers of the outer spatial query window, setCandidatesOut, since the corresponding MBR is not within i , is not within o , but intersects o (lines 7, 11, 13-16) and its MIP5 also intersects o . Note that a subsequent refinement is unnecessary because the set of candidates regarding o is empty.

The procedure fvsrq2 also evaluates the predicates *containment* and *intersects* for each entry. On the one hand, checking at least one and at most four spatial predicates per entry adds an overhead. On the other hand, it can reduce the set of candidates to be checked in the refinement step and, then, benefit the performance to process the query. The procedure

fvsrq2 requires processing more vertices than the procedure fvsrq1, since the former accesses progressive approximations, while the latter accesses conservative approximations. However, the use of progressive approximations in procedure fvsrq2 allows reducing even more the set of candidates if compared to the procedure fvsrq1, used in Examples 6.3.8 and 6.3.9.

The calls made to procedures fvsrq1 and fvsrq2, depend on the configuration of the VSB-index and the type of *VSRQ*. These calls are listed in Section 6.3.5.3.

6.3.5.3 Calling the Procedures

For each configuration of the VSB-index, the calls made to routines fvsrq1 and fvsrq2 to resolve *VSRQ_{object}*, *VSRQ_{certitude}*, and *VSRQ_{dubiety}* are the following. The parameters *setAnswersIn*, *setAnswersOut*, *setCandidatesOut*, and *idx* were omitted for the sake of simplicity.

$O_{\supseteq}O_{\subseteq}C_{\supseteq}C_{\subseteq}$ *VSRQ_{object}*: fvsrq2(wIn, wOut, O_{\supseteq} , O_{\subseteq} , ..., L)
VSRQ_{dubiety}: fvsrq2(wIn, wOut, O_{\supseteq} , O_{\subseteq} , ..., L)
VSRQ_{certitude}: fvsrq2(wIn, wOut, C_{\supseteq} , C_{\subseteq} , ..., L)

$O_{\supseteq}O_{\subseteq}C_{\supseteq}$ *VSRQ_{object}*: fvsrq2(wIn, wOut, O_{\supseteq} , O_{\subseteq} , ..., L)
VSRQ_{dubiety}: fvsrq2(wIn, wOut, O_{\supseteq} , O_{\subseteq} , ..., L)
VSRQ_{certitude}: fvsrq1(wIn, wOut, C_{\supseteq} , ..., L)

$O_{\supseteq}O_{\subseteq}C_{\subseteq}$ *VSRQ_{object}*: fvsrq2(wIn, wOut, O_{\supseteq} , O_{\subseteq} , ..., L)
VSRQ_{dubiety}: fvsrq2(wIn, wOut, O_{\supseteq} , O_{\subseteq} , ..., L)
VSRQ_{certitude}: fvsrq1(wIn, wOut, C_{\supseteq} , ..., L)

$O_{\supseteq}O_{\subseteq}$ *VSRQ_{object}*: fvsrq2(wIn, wOut, O_{\supseteq} , O_{\subseteq} , ..., L)
VSRQ_{dubiety}: fvsrq2(wIn, wOut, O_{\supseteq} , O_{\subseteq} , ..., L)
VSRQ_{certitude}: -

$O_{\supseteq}C_{\supseteq}C_{\subseteq}$ *VSRQ_{object}*: fvsrq1(wIn, wOut, O_{\supseteq} , ..., L)
VSRQ_{dubiety}: fvsrq1(wIn, wOut, O_{\supseteq} , ..., L)
VSRQ_{certitude}: fvsrq2(wIn, wOut, C_{\supseteq} , C_{\subseteq} , ..., L)

$O_{\supseteq}C_{\supseteq}$ *VSRQ_{object}*: fvsrq1(wIn, wOut, O_{\supseteq} , ..., L)
VSRQ_{dubiety}: fvsrq1(wIn, wOut, O_{\supseteq} , ..., L)
VSRQ_{certitude}: fvsrq1(wIn, wOut, C_{\supseteq} , ..., L)

$O_{\supseteq}C_{\subseteq}$ *VSRQ_{object}*: fvsrq1(wIn, wOut, O_{\supseteq} , ..., L)
VSRQ_{dubiety}: fvsrq1(wIn, wOut, O_{\supseteq} , ..., L)
VSRQ_{certitude}: -

$$\begin{aligned}
O_{\supseteq} \quad VSRQ_{object}: & \text{fvsrq1}(wIn, wOut, O_{\supseteq}, \dots, L) \\
VSRQ_{dubiety}: & \text{fvsrq1}(wIn, wOut, O_{\supseteq}, \dots, L) \\
VSRQ_{certitude}: & -
\end{aligned}$$

6.4 Evaluation of the VSB-index

The prohibitive performance of the DBMS and exiting indices for SDWs identified in Section 6.2 motivated the development of the VSB-index that was detailed in Section 6.3. An experimental evaluation of the VSB-index is essential to corroborate the feasibility of its utilization in vague SDWs. The experiments described in the following sections aims to:

- store and query vague spatial regions with different characteristics;
- process vague spatial predicates that assess the vague spatial region, its certitude, and its dubiety;
- verify the influence of increasing the selectivity of the vague spatial predicate over the performance to process queries;
- discuss the benefits of using the MIP as progressive approximation;
- identify the most efficient VSB-index configuration in each case;
- estimate the benefits of the VSB-index over related work, such as the SB-index and the aR-tree;
- measure the storage requirements of the VSB-index; and
- identify the limitations of the VSB-index.

To comply with the aforementioned goals, the following sections are organized as follows. Section 6.4.1 describes the setup of the experiments. Two different vague SDWs are utilized. One vague SDW was called real because vague regions were created based on the real dataset provided by Embrapa regarding the HLB case study. The other vague SDW was called synthetic because vague regions were created by processing a dataset that was not related to HLB. Besides, the vague spatial predicates IRQ_{object} , $IRQ_{dubiety}$, $IRQ_{certitude}$, CRQ_{object} , $CRQ_{dubiety}$ and $CRQ_{certitude}$, and $VSRQ_{object}$ were assessed. They not only focus on vague spatial regions, but also fetch their certitude and dubiety.

Sections 6.4.2 to 6.4.4 tackle the resolution of vague spatial predicates, since it has been identified as a bottleneck to efficiently process queries in vague SDWs in Section 6.2. Sections 6.4.2 and 6.4.3 focuses intersection range queries, which might require the refinement step, using the real vague SDW and the synthetic vague SDW, respectively. Section 6.4.4 addresses containment range queries, which might not require the refinement step, and uses the synthetic vague SDW. Section 6.4.5 tackles the vague spatial range query, which is more costly due to the utilization of two spatial query windows, and uses the synthetic vague SDW. Finally, Section 6.4.6 addresses the elapsed time to build indices and storage requirements of indices.

6.4.1 Experimental Setup

Section 6.4.1.1 describes the workbench and the hardware and software platforms, while Section 6.4.1.2 details the workload.

6.4.1.1 Workbench and Platforms

The workbench comprised two vague SDWs stored in the DBMS. Both had the same logical design depicted in Figure 6.15. The *synthetic* vague SDW stored vague regions that were created as already described in Section 6.2.2.1. The term 'synthetic' refers to processing a dataset that was not originally associated to the HLB case study, despite the produced vague regions were intrinsically regions with broad boundaries. The synthetic vague SDW was loaded as described in Section 6.2.2.1, i.e. with 60,000,000 rows in the table HLBControl and 302,357 rows and distinct vague regions in the table InfectedRegion1.

The *real* vague SDW stored vague regions created by processing a real dataset regarding the HLB disease, which was provided by Embrapa. The term 'real' refers to real characteristics of HLB mentioned in Section 1.2 that were applied to the dataset in order to create vague regions. The original dataset comprised data collected in the field in 13 different months regarding one citrus plot with approximately 9,000 trees. Each tree was originally represented by a point with the status of infected or healthy. Every region infected by HLB was modeled as a vague region, like r_1 and r_2 illustrated in Figure 6.16. A total of 129 distinct infected regions were created and stored in the table InfectedRegion1. The table HLBControl also stored 129 rows. The certitude was the extent where trees were infected and eradicated. Conversely, the dubiety was the broad boundary where the insect possibly transmitted the bacterium to trees that were not eradicated, but that became suspicious.

In order to create the certitude, a buffer was applied on the point of each infected tree, e.g.

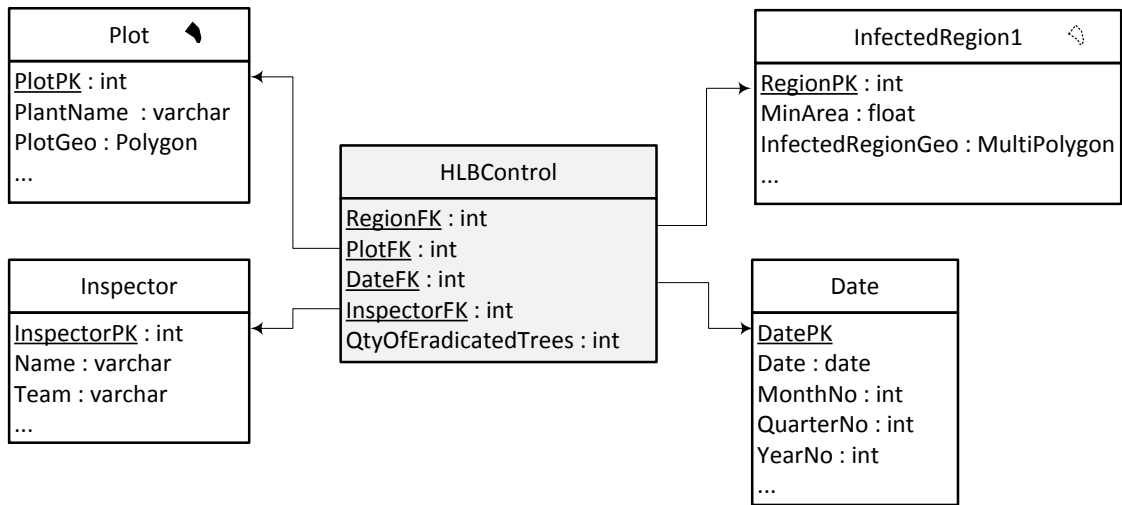


Figure 6.15: The schema of both the synthetic vague SDW and the real vague SDW.

r_1 shown in Figure 6.16. Such buffer did not intersect any other tree (infected or healthy). The certitude also encompassed the infection of neighbor trees as follows. If two or more buffers intersected each other and did not cover any healthy tree, then such buffers were merged and considered as a single certitude element where two or more trees were eradicated, e.g. r_2 in Figure 6.16.

Furthermore, the dubiety of each infected region was designed by firstly fetching all trees within 25 meters from each infected tree. Secondly, a convex hull containing each infected tree and its neighbors was built, e.g. r_1 shown in Figure 6.16. The convex hulls built for two or more different infected trees were also merged whether their corresponding certitudes had already been merged previously, e.g. r_2 in shown Figure 6.16. Finally, the certitude was subtracted from the dubiety to ensure that both had disjoint interiors.

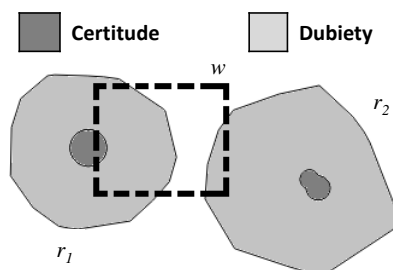


Figure 6.16: Two vague regions and one spatial query window w .

Relevant characteristics of the vague regions stored by the different vague SDWs are highlighted because these characteristics influenced the results discussed in Sections 6.4.3 to 6.4.4. In the synthetic vague SDW, the average ratio between the area of the certitude of a vague re-

gion and the area of the vague region was 0.7817. In other words, the extent of the certitude represented, on average, approximately 78.17% of the extent of the vague region. The standard deviation was 9.69%. In the real SDW, such ratio was much lower, i.e. 1.25%, while the standard deviation was 1.13%.

In the synthetic vague SDW, each vague region had, on average, 53% of the extent overlapped by other vague regions. The standard deviation was 24% of the area. In addition, only 30 of the 302,357 vague regions did not overlap another vague region. On the other hand, in the real vague SDW, each vague region had, on average, 196.34% of the extent overlapped by other vague regions. The standard deviation was 162.70% of the area. Also, all vague regions were overlapped by another vague region. These measurements emphasized the intrinsic characteristic of infestation of neighbor trees in different epochs, as discussed in Section 1.2.

All feasible configurations of the VSB-index were implemented using MBR as conservative approximation and MIP5 as progressive approximation. Five vertices were chosen by analogy with 5C (BRINKHOFF; KRIEGEL; SCHNEIDER, 1993). They were implemented in C/C++ and the disk page size was set to 8 KB. MIP5 was built using the CGAL, Computational Geometry Algorithms Library⁴ 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 and x is the number of vertices of the MIP.

The following indices were created in both the synthetic vague SDW and the real vague SDW. GiSTs were built on the column `InfectedRegionGeo`. The SB-index and the aR-tree were implemented in C/C++ and their disk page size was set to 8 KB. Both were built on the column `InfectedRegionGeo` and, therefore, processed their refinement steps accessing the multipolygons. The bitmap join indices of the SB-index were implemented using FastBit (WU et al., 2009) and built on the columns `RegionFK`, `Team`, `Year`, and `QtyOfEradicatedTrees` to comply with the workload described in Section 6.4.1.2. Analogously, the entries of the aR-tree referenced one multidimensional array containing values of the measure `QtyOfEradicatedTrees` assigned to each pair of `Team` and `Year`.

The hardware 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. The software platform comprised 8 MB of cache, Linux CentOS 6.4, PostgreSQL 9.2.2 and PostGIS 2.0.1.

⁴<http://www.cgal.org>

Table 6.2: Spatial range queries of the workload.

Predicate	R
IRQ_{object} or $IRQ_{dubiety}$	ST_Intersects(InfectedRegionGeo, w)
$IRQ_{certitude}$	ST_Intersects(ST_GeometryN(InfectedRegionGeo, 2), w)
CRQ_{object} or $CRQ_{dubiety}$	ST_Within(InfectedRegionGeo, w)
$CRQ_{certitude}$	ST_Within(ST_GeometryN(InfectedRegionGeo, 2), w)

6.4.1.2 Workload

The spatial range queries IRQ_{object} , $IRQ_{dubiety}$, $IRQ_{certitude}$, CRQ_{object} , $CRQ_{dubiety}$, and $CRQ_{certitude}$ were chosen for the workload, as they assess different topological relationships and fetch different parts of a vague region. The template query issued over the schema shown in Figure 6.16 is described in Listing 6.5, where R is one vague spatial predicate written in SQL according to Table 6.2.

Listing 6.5: The template query for the vague SDWs.

```

SELECT Team, Year, SUM(QtyOfEradicatedTrees)
FROM Inspector, Date, Infection1, HLBControl
WHERE InspectorPK = InspectorFK AND
      DatePK = DateFK AND
      RegionPK = RegionFK AND
      Team = 'XV' AND
      R
GROUP BY Team, Year
ORDER BY Team, Year;

```

For the vague spatial predicates listed in Table 6.2, w was a rectangular spatial query window whose shape was not stored in the vague SDWs, as shown in Figure 6.16. IRQ_{object} and $IRQ_{dubiety}$ were processed using the same SQL source code because any spatial query window was created in the hole of the dubiety of a vague region (Section 6.3.4). Regarding $IRQ_{certitude}$ and $CRQ_{certitude}$, the OGC function $ST_GeometryN$ retrieved the second polygon in the multipolygon of a vague region, which corresponded to the certitude of every vague region stored in both the synthetic vague SDW and real vague SDW.

The following procedure to elaborate the queries of the workload was applied for each vague spatial predicate listed in Table 6.2. The selectivities utilized to query the synthetic vague SDW were 0.0001, 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. They ensured that 30 up to 12,000 vague regions of the synthetic vague SDW were queried. For each selectivity, 10 disjoint and rectangular spatial query windows were created. Each spatial query window was assigned to one query. The execution of queries complied with the ascending order of the selectivity. Cache and buffers were flushed each ten queries, i.e., before starting the execution of a query with greater selectivity.

Conversely, the following selectivities were used for the real vague SDW: 0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.10, 0.50 and 1.00. The null selectivity represented the selection of zero elements of the dataset. Low selectivities under 0.10 denoted the user selecting a constrained extent of interest. Selectivities between 0.10 and 0.50 concerned the user selecting a wider extent of interest. The high selectivity of 1.0 retrieves the whole dataset, e.g. when it must be fully displayed. Moderate and high selectivities were acceptable since the data volume of vague regions was low. The elaboration and execution of the queries for the real vague SDW followed the same procedure already described for the synthetic vague SDW.

The vague spatial range query $VSRQ_{object}$ was also selected for the workload to query the synthetic vague SDW. The template query is shown in Listing 6.6. The spatial query windows i (inner) and o (outer) were rectangular and concentric, such that the edges of i measured half of the length of the edges of o . The selectivities for o were 0.0001, 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. Ten disjoint pairs of spatial query windows were created per selectivity of o . The execution of queries complied with the ascending order of the selectivity. Cache and buffers were flushed each ten queries, i.e., before starting the execution of a query with greater selectivity.

Listing 6.6: The template query for the $VSRQ_{object}$.

```

SELECT Team, Year, SUM(QtyOfEradicatedTrees), 'More relevant '
FROM Inspector, Date, InfectedRegion1, HLBCControl
WHERE InspectorPK = InspectorFK
      DatePK = DateFK AND
      RegionPK = RegionFK AND
      Team = 'XV' AND
      ST_Within(InfectedRegionGeo, i)
GROUP BY Team, Year
ORDER BY Team, Year

UNION

SELECT Team, Year, SUM(QtyOfEradicatedTrees), 'Less relevant '
FROM Inspector, Date, InfectedRegion1, HLBCControl
WHERE InspectorPK = InspectorFK AND
      DatePK = DateFK AND
      RegionPK = RegionFK AND
      Team = 'XV' AND
      ST_Intersects(InfectedRegionGeo, o) AND NOT ST_Within(InfectedRegionGeo, i)
GROUP BY Team, Year
ORDER BY Team, Year

```

6.4.2 Intersection Range Queries over the Real Vague SDW

The following sections report and discuss the performance results achieved by the VSB-index and the SB-index to process different types of intersection range queries from the workload described in Section 6.4.1.2 over the real vague SDW detailed in Section 6.4.1.1. Increasing selectivities of vague spatial predicates were considered. The measurements gathered were

the average elapsed time and the average number of candidates produced in the filter step, as these candidates were subsequently processed in the refinement step. The goal was to investigate the relation between the performance and the efficiency of the filter step. Each configuration of the VSB-index and the SB-index were evaluated. The aR-tree was not tested because the low data volume of the real vague SDW would not allow the investigation of the benefits of pruning the tree in the filter step. Section 6.4.2.1 focuses on the resolution of IRQ_{object} and $IRQ_{dubiety}$, while Section 6.4.2.2 tackles $IRQ_{certitude}$.

6.4.2.1 IRQobject and IRQdubiety

Figure 6.17a reports the average elapsed time to process IRQ_{object} and $IRQ_{dubiety}$ over the real vague SDW, while Figure 6.17b details the average number of candidates. High average numbers of candidates as 65 and 129 were omitted for the configuration O_{MBR} and selectivities 0.50 and 1.00, respectively.

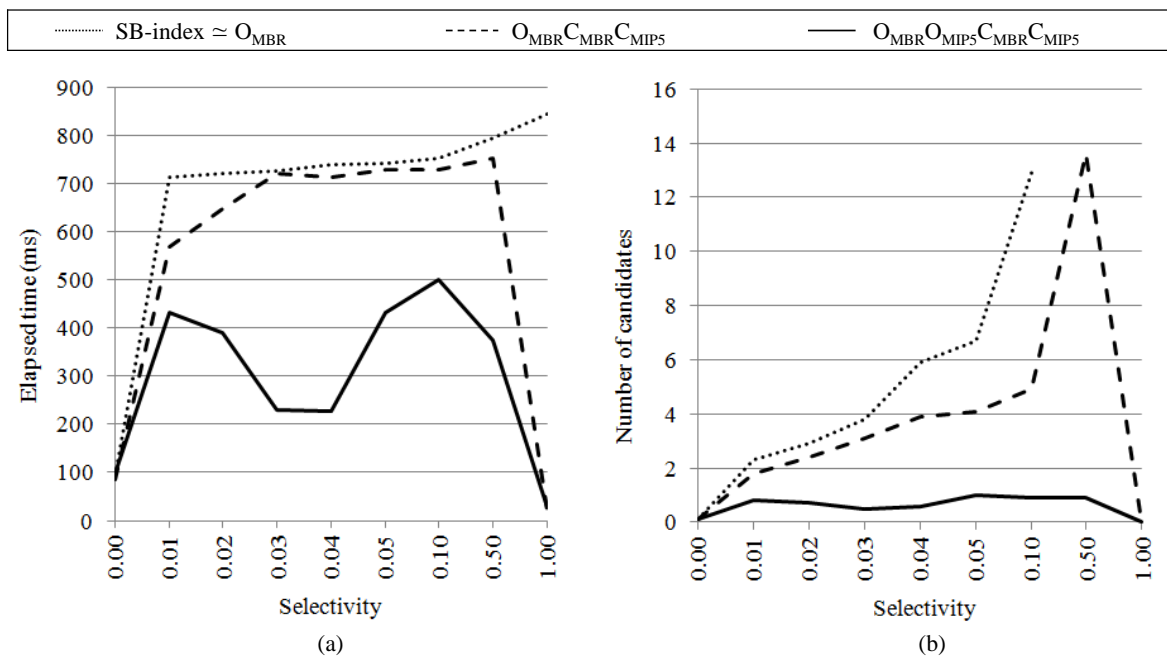


Figure 6.17: Results for IRQobject and IRQdubiety issued over the real vague SDW: (a) Average elapsed time. (b) Average number of candidates.

Every configuration spent time to process queries with selectivity 0, due to the execution of the filter step and afterwards of the refinement step whether there were candidates. Regarding selectivities between 0.01 and 0.50, the average number of candidates processed by each configuration determined its performance, as follows.

Configurations that do not hold any MIP5, as O_{MBR} , required longer elapsed times independently of the selectivity. Their filter steps are not able to identify answers of IRQ_{object} and

$IRQ_{dubiety}$ due to the absence of a progressive approximation. Then, more candidates were processed by the refinement step, increasing its cost and causing low performance. The results of the configuration O_{MBR} were equivalent to those obtained by the SB-index and like those gathered for the configuration $O_{MBR}O_{MIP5}$.

Configurations that have a MIP5 built only on the certitude, as $O_{MBR}C_{MBR}C_{MIP5}$, are able to identify answers in the filter step by using such progressive approximation. However, not many answers were identified since the extent of the certitude occupied a very small portion of the vague region (Section 6.4.1.1). Then, the MIP5 built on the certitude was an even smaller subset of the vague region, as depicted in Figure 6.12d. As a result, average numbers of candidates for these configurations were not drastically reduced, the refinement step was costly and their query response times were long. The results of the configuration $O_{\supseteq}C_{\supseteq}C_{\subseteq}$ were like those obtained by the configuration $O_{MBR}C_{MIP5}$.

Shortest elapsed times were spent by configurations with MIP5 built on the vague region, as $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$. These configurations were not impaired by increasing selectivities. They had low average numbers of candidates, even for greater selectivities. This trend did not occur in the other configurations. Due to processing fewer candidates in the refinement step, configuration $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$ imposed a time reduction of at least 33% and at most 69% over SB-index (equivalent to configuration O_{MBR}) for selectivities 0.10 and 0.04, respectively. The results of the configuration $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$ were like those obtained by the configurations $O_{MBR}C_{MBR}$, $O_{MBR}O_{MIP5}C_{MBR}$ and $O_{MBR}O_{MIP5}C_{MIP5}$.

Finally, regarding the selectivity 1.00, all configurations holding at least one MIP5 had an empty set of candidates to process in the refinement step, since all answers were identified already in the filter step. As a result, these configurations required shorter elapsed times.

6.4.2.2 IRQcertitude

Figure 6.18a reports the average elapsed time to process $IRQ_{certitude}$ over the real vague SDW, while Figure 6.18b details the average number of candidates. High average numbers of candidates as 65 and 129 were omitted for the configuration O_{MBR} and selectivities 0.50 and 1.00, respectively.

Every configuration took time to process queries with selectivity 0, executing the filter step and the refinement step whether there were candidates. Regarding selectivities between 0.01 and 0.50, the average number of candidates processed by each configuration determined its performance, as follows.

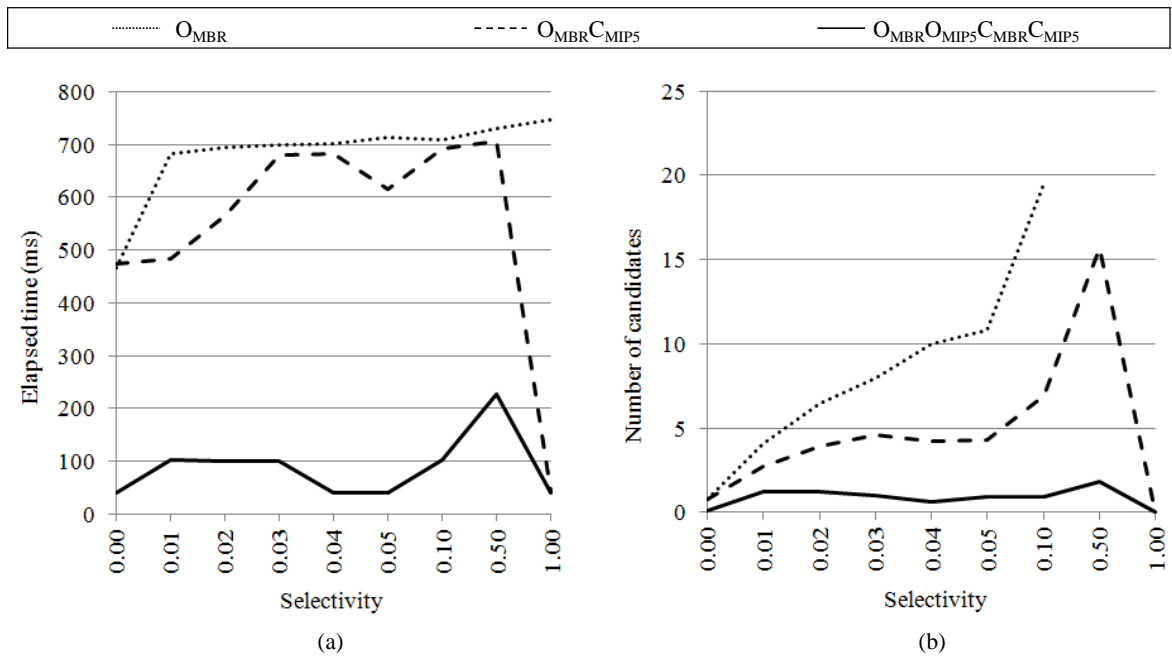


Figure 6.18: Results for IRQcertitude issued over the real vague SDW: (a) Average elapsed time. (b) Average number of candidates.

Configurations that do not hold MIP5 on the certitude, as O_{MBR} , spent longer elapsed times independently of selectivity. Their filter steps were not able to identify answers of $IRQ_{certitude}$ due to the absence of a progressive approximation on the certitude. Then, more candidates were processed by the refinement step, increasing its cost and causing low performance. The results of the configuration O_{MBR} were like those obtained by the configurations $O_{MBR}O_{MIP5}$, $O_{MBR}C_{MBR}$ and $O_{MBR}O_{MIP5}C_{MBR}$.

Configurations that hold MIP5 on the certitude but do not hold MBR on the certitude, as $O_{MBR}C_{MIP5}$, accessed the MBR built on the vague region in their filter steps. The MBR of the vague region covered a wider extent than the extent covered by the MIP5 on the certitude as illustrated in Figure 6.12d, since the certitude occupied a small portion of the vague region as explained in Section 6.4.1.1 Therefore, these configurations were not able to reduce the set of candidates and their costly refinement step determined a low performance. The results of the configuration $O_{MBR}C_{MIP5}$ were like those obtained by the configuration $O_{MBR}O_{MIP5}C_{MBR}$.

Configurations that hold both MBR and MIP5 on the certitude, such as $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$, required shorter elapsed time than the other configurations. Their average number of candidates was low due to the use of MIP5 on the certitude. Therefore, their refinement steps were less costly and the performance was benefited. The configuration $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$ provided a time reduction of at least 84% and at most 94% over the configuration O_{MBR} , for selectivities 0.05 and 0.02, respectively. Although the configuration O_{MBR} has a data structure equivalent to

the SB-index' data structure, the latter does not have an algorithm to process $IRQ_{certitude}$. The results of the configuration $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$ were like those obtained by the configuration $O_{MBR}C_{MBR}C_{MIP5}$.

Configurations with a MIP5 on the certitude had an empty set of candidates to process in the refinement step for selectivity 1.00. As all answers were identified in the filter step, the elapsed times for these configurations were shorter.

6.4.3 Intersection Range Queries over the Synthetic Vague SDW

On the one hand, it has been recognized that the MIP5 significantly reduced the set of candidates produced by the filter step and processed in the refinement step and, therefore, improved the performance of intersection range queries over the real vague SDW as discussed in Section 6.4.2. On the other hand, the following sections reports and discuss the performance results achieved by the VSB-index, the SB-index and the aR-tree to process different types of intersection range queries from the workload described in Section 6.4.1.2 over the synthetic vague SDW detailed in Section 6.4.1.1. Differently from the real vague SDW, the synthetic vague SDW stores a huge volume of vague regions whose certitudes are relatively large. The average elapsed time was gathered for each configuration of the VSB-index and for the SB-index and the aR-tree, considering increasing selectivities. Section 6.4.3.1 focuses on the resolution of IRQ_{object} and $IRQ_{dubiety}$, while Section 6.4.3.2 tackles the $IRQ_{certitude}$.

6.4.3.1 IRQ_{object} and $IRQ_{dubiety}$

Figure 6.19 reports the results concerning IRQ_{object} and $IRQ_{dubiety}$. Five configurations of the VSB-index are shown: O_{MBR} , $O_{MBR}C_{MIP5}$, $O_{MBR}C_{MBR}$, $O_{MBR}O_{MIP5}$ and $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$. The remaining were omitted to facilitate the visualization of the significant difference among results.

Configurations that have a MIP5, such as $O_{MBR}C_{MIP5}$, $O_{MBR}O_{MIP5}$ and $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$ did not have their performance impaired by increasing selectivities. Conversely, increasing selectivities severely degenerated the performance of configurations that do not have a MIP5, such as O_{MBR} and $O_{MBR}C_{MBR}$.

The configuration $O_{MBR}O_{MIP5}$ outperformed the other configurations because MIP5 allows identifying answers of the spatial predicate already in the filter step. Conversely, both configurations O_{MBR} and $O_{MBR}C_{MBR}$ do not maintain a progressive approximation and cannot identify answers in the filter step. As a result, their performances were severely impaired because they

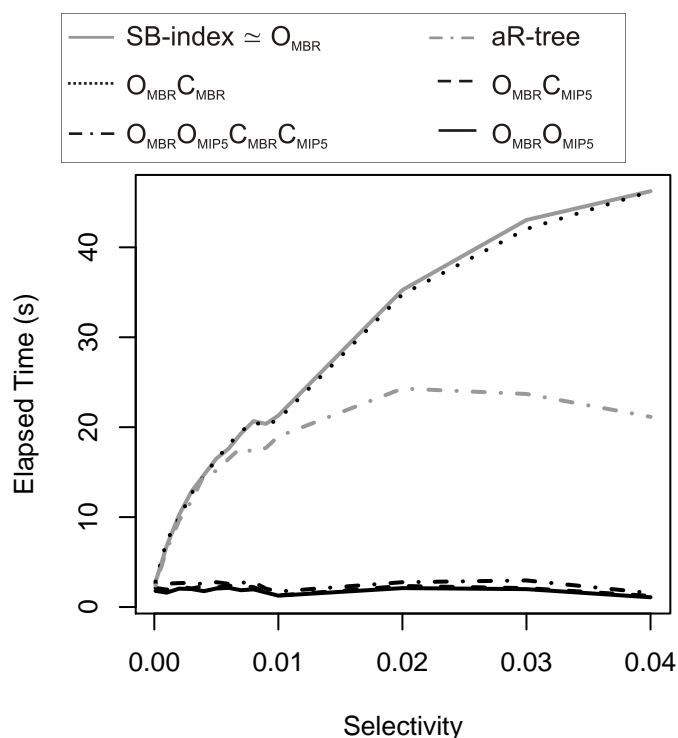


Figure 6.19: The performance to process IRQ_{object} and $IRQ_{dubiety}$ over the synthetic vague SDW.

had a costly refinement step.

Furthermore, the configuration $O_{MBR}O_{MIP5}$ overcame the configuration $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$. Note that both configurations have a MIP5 built on the vague region, which is utilized to identify answers in the filter step. However, the configuration $O_{MBR}O_{MIP5}$ has a smaller index entry size than configuration $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$, and the filter step of the former requires less disk accesses and than the filter step of the latter, as shown in Table 6.1. As a result, the former spent less time to process the queries than the latter.

The configuration $O_{MBR}C_{MIP5}$ also overcame the configuration $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$. Note that configuration $O_{MBR}C_{MIP5}$ has a MIP5 built on the certitude, while the configuration $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$ has a MIP5 built on the vague region. This result demonstrated that the MIP5 built on the certitude was also beneficial to improve the performance of IRQ_{object} and $IRQ_{dubiety}$, which assess the vague region and the dubiety, respectively.

Such achievement was feasible due to an intrinsic characteristic of the synthetic vague SDW: the certitude occupied a large portion of the vague region, as described in Section 6.4.1.1. As a result, a progressive approximation created on the certitude was also a large subset of the extent of the vague region and benefited the performance to process IRQ_{object} and $IRQ_{dubiety}$.

SB-index' results were equivalent to those achieved by the configuration O_{MBR} to exe-

cute IRQ_{object} and IRQ_{dubity} . All configurations obtained equivalent results for the selectivity 0.0001. Considering selectivities greater than 0.0001, the configuration $O_{MBR}C_{MIP5}$ provided a time reduction of at least 71% and at most 97% over the SB-index, for selectivities 0.001 and 0.04, respectively. Besides, the configuration $O_{MBR}C_{MIP5}$ provided a time reduction of at least 69% and at most 94% over the aR-tree, for selectivities 0.001 and 0.04, respectively.

6.4.3.2 IRQcertitude

Figure 6.20 reports the results concerning $IRQ_{certitude}$. Five configurations of the VSB-index are shown, i.e. O_{MBR} , $O_{MBR}C_{MIP5}$, $O_{MBR}C_{MBR}$, $O_{MBR}O_{MIP5}$ and $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$, while three were omitted to facilitate the visualization of the significant difference among results. Although the configuration O_{MBR} has a data structure equivalent to the SB-index' data structure, the latter does not have an algorithm to process $IRQ_{certitude}$. The aR-tree also does not have an algorithm to process $IRQ_{certitude}$. Thus, the aR-tree and the SB-index are not reported in Figure 6.20.

Configurations that have a MIP5 built on the certitude did not have their performance impaired by increasing selectivities, such as $O_{MBR}C_{MIP5}$ and $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$. Conversely, increasing selectivities severely degenerated the performance of configurations that do not have a MIP5 built on the certitude, such as O_{MBR} , $O_{MBR}O_{MIP5}$ and $O_{MBR}C_{MBR}$, which had comparable performance results.

Configurations $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$ and $O_{MBR}C_{MIP5}$ outperformed the other configurations because the MIP5 built on the certitude is essential to identify answers of the $IRQ_{certitude}$ already in the filter step. On the other hand, the MIP5 built on the vague region did not enable such identification, and therefore the performance of the configuration $O_{MBR}O_{MIP5}$ was impaired. Again, the configuration $O_{MBR}C_{MIP5}$ overcame the configuration $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$ because the former performs less disk access than the latter in the filter step, as previously commented. Moreover, the configuration $O_{MBR}C_{MIP5}$ imposed a time reduction from 36% up to 97% over the configuration O_{MBR} for selectivities 0.0001 and 0.04, respectively.

6.4.4 Containment Range Queries over the Synthetic Vague SDW

On the one hand, it has been recognized that the MIP5 significantly reduced the set of candidates produced by the filter step and processed in the refinement step and, therefore, improved the performance of intersection range queries over the real vague SDW and the synthetic vague SDW, as discussed in Sections 6.4.2 and 6.4.3, respectively. On the other hand, the following

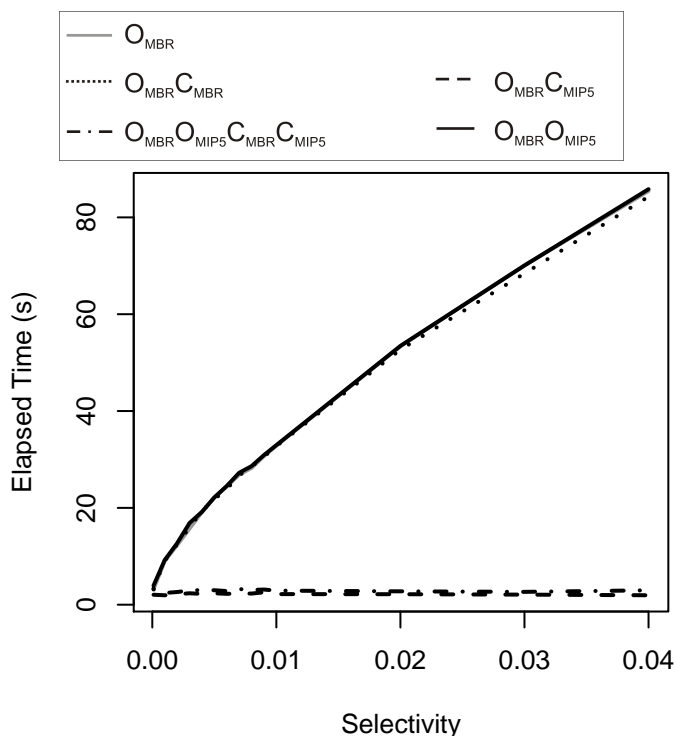


Figure 6.20: The performance to process the $IRQ_{certitude}$ over the synthetic vague SDW.

sections reports and discuss the performance results achieved by the VSB-index, the SB-index and the aR-tree to process different types of containment range queries from the workload described in Section 6.4.1.2 over the synthetic vague SDW detailed in Section 6.4.1.1. Differently from intersection range queries, containment range queries should not require the refinement step. The average elapsed time was gathered for each configuration of the VSB-index and for the SB-index and the aR-tree, considering increasing selectivities. Section 6.4.4.1 focuses on the resolution of CRQ_{object} and $CRQ_{dubiety}$, while Section 6.4.4.2 tackles the $CRQ_{certitude}$.

6.4.4.1 CRQ_{object} and $CRQ_{dubiety}$

This section compares the performance obtained by the aR-tree and the SB-index to the following configurations of the VSB-index: O_{MBR} , $O_{MBR} O_{MBR}$ and $O_{MBR} O_{MIP5} C_{MBR} C_{MIP5}$. The configuration O_{MBR} was selected because it holds a single MBR as approximation and can process CRQ_{object} and $CRQ_{dubiety}$ without a refinement step. The configuration $O_{MBR} O_{MBR}$ was selected because it can process CRQ_{object} , $CRQ_{dubiety}$ and $CRQ_{certitude}$ without a refinement step. The configuration $O_{MBR} O_{MIP5} C_{MBR} C_{MIP5}$ was chosen because it is the only configuration that can retrieve answers of the vague spatial predicate already in the filter step, for all the vague spatial predicates of the workload. The results of the remaining configurations of the VSB-index were omitted to facilitate the visualization of the significant difference among results.

Figure 6.21 reports the average elapsed time spent by each configuration according to different selectivities. The results achieved by the configuration O_{MBR} of the VSB-index were equivalent to the results gathered for the SB-index.

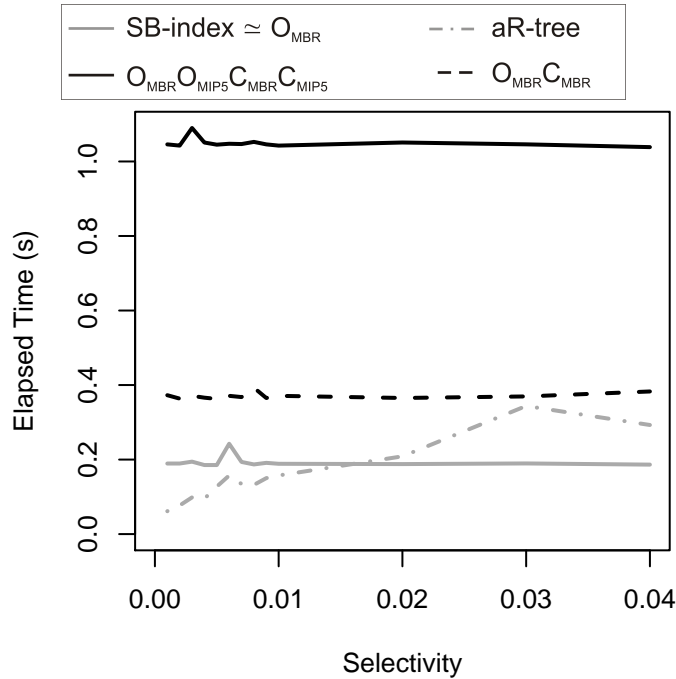


Figure 6.21: Average elapsed time to process CRQobject and CRQdubity over the synthetic vague SDW.

Regarding the VSB-index, the best results were achieved by configuration O_{MBR} due to the following reasons. All configurations held MBRs on the vague regions that were accessed to find answers already in the filter step. Also, the sequential scan performed on their sequential files required a fixed number of disk accesses, according to Table 6.1. Then, configurations that performed fewer disk accesses were expected to spend shorter elapsed times. The configuration O_{MBR} performed fewer disk accesses and consequently also spent shorter elapsed times than configurations $O_{MBR} O_{MIP5}$ and $O_{MBR} O_{MIP5} C_{MBR} C_{MIP5}$. It is noteworthy that MIP5s were not accessed to process queries using the configuration $O_{MBR} O_{MIP5} C_{MBR} C_{MIP5}$ but, since they were stored, they introduced overheads to the query response time.

The aR-tree outperformed both the VSB-index and the SB-index for selectivities lower than 0.02. The traversal of the aR-tree was pruned in the filter step. As a result, the aR-tree performed less disk accesses than the VSB-index and the SB-index, which had a fixed number of disk accesses in the filter step. However, for selectivities 0.02, 0.03 and 0.04, the configuration O_{MBR} of the VSB-index overcame the aR-tree and imposed time reductions of 9%, 44% and 36%, respectively.

To understand the low performance of the aR-tree for greater selectivities, recall that: (i)

spatial objects that overlap might impair the performance of spatial indices (Section 2.1.3.2); (ii) the aR-tree is based on the R-tree and, therefore, is also impaired by overlapping (Section 2.1.3.2); (iii) vague regions stored by the synthetic vague SDW overlapped each other, since their shapes corresponded to convex hulls of polygons (Section 6.2.2.2); and (iv) the aR-tree was built using the MBRs of those vague regions. Clearly, the MBRs of the vague regions in the synthetic vague SDW also overlapped each other. Then, as spatial query windows became larger, they contained several large MBRs that overlapped each other and also belonged to different non-leaf nodes. As a result, instead of pruning the tree traversal in the filter step, the aR-tree was impaired by the ramification of the tree traversal due to existence of overlapping among the indexed vague regions.

6.4.4.2 CRQcertitude

Since the SB-index and the aR-tree do not provide algorithms to process $CRQ_{certitude}$, this section compares the performance obtained only by the following configurations of the VSB-index: O_{MBR} , $O_{MBR}C_{MBR}$ and $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$. The configuration O_{MBR} was chosen because it achieved the best results of the VSB-index to process CRQ_{object} and $CRQ_{dubiety}$. However, to process $CRQ_{certitude}$, it might have an overhead due to a refinement step that is mandatory because the configuration does not have an approximation for the certitude. Conversely, both the configurations $O_{MBR}C_{MBR}$ and $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$ were chosen because they do not require a refinement step.

Results shown in Figure 6.22 reveal that the refinement step severely impaired the performance of the configuration O_{MBR} , as expected. Like Section 6.4.4.1, the best configuration of the VSB-index was that holding the minimum set of approximations necessary to process the vague spatial predicate, i.e. $O_{MBR}C_{MBR}$. The MBR built on the certitude was sufficient to identify all the answers of the spatial predicate already in the filter step. Although both configurations $O_{MBR}C_{MBR}$ and $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$ have this capability, the former requires less disk accesses than the latter to perform the sequential scan of the filter step, according to Table 6.1. As a result, the configuration $O_{MBR}C_{MBR}$ outperformed the configuration $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$.

6.4.5 Vague Spatial Range Queries over the Synthetic Vague SDW

In Sections 6.4.2 and 6.4.3, it has been recognized that the MIP5 significantly reduced the set of candidates produced by the filter step and processed in the refinement step and, therefore, improved the performance of intersection range queries over the real vague SDW as discussed

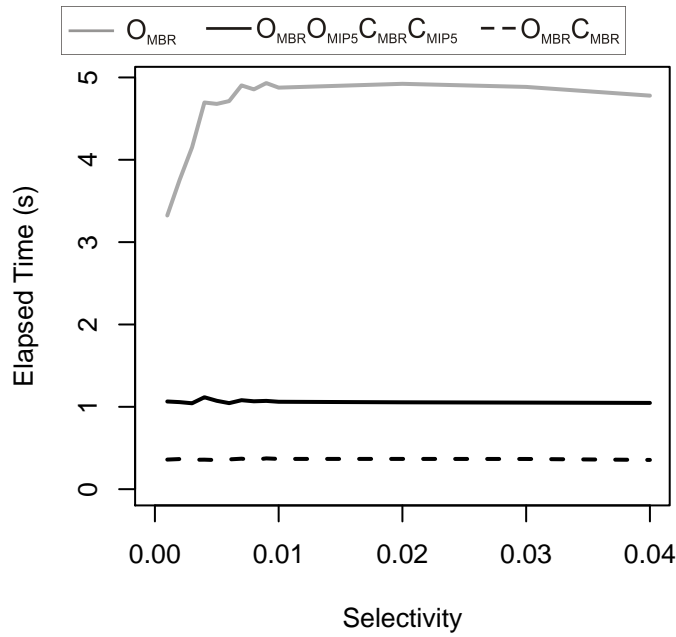


Figure 6.22: Average elapsed time to process CRQcertitude over the synthetic vague SDW.

in Sections 6.4.2 and 6.4.3. In Section 6.4.4, the existence of a MIP5 in the VSB-index configuration has been associated to a performance loss to process containment range queries because the MBR has been sufficient for the identification of answers already in the filter step. In contrast with the previous experiments, the following sections report and discuss the performance results achieved by the VSB-index, the SB-index and the aR-tree to process a type of vague spatial range query against vague regions, i.e. $VSRQ_{object}$, according to the workload described in Section 6.4.1.2. Since the $VSRQ_{object}$ evaluates a pair of topological relationships using a pair of distinct spatial query windows, it might require a refinement step. Section 6.4.5.1 describes the test configurations used in the experiments while Section 6.4.5.2 reports the results.

6.4.5.1 Test Configurations

The following test configurations were used in the experiments involving the $VSRQ_{object}$:

- $O_{MBR} O_{MIP5} C_{MBR} C_{MIP5}$ is a configuration of the VSB-index built on the columns *InfectionPK* and *InfectedRegionGeo* of the table *InfectedRegion1* shown in Figure 6.15;
- O_{MBR} is a configuration of the VSB-index built on the columns *InfectionPK* and *InfectedRegionGeo* of the table *InfectedRegion1* shown in Figure 6.15;
- aR-tree1 is the aR-tree built on the columns *InfectionPK* and *InfectedRegionGeo* of the table *InfectedRegion1* shown in Figure 6.15, such that ‘1’ refers to vague regions described by a single geometry column of type MultiPolygon;

- aR-tree2 is the aR-tree built on the columns *InfectionPK* and *InfectedRegionDubietyGeo* of the table *InfectedRegion2* shown in Figure 6.5b, such that ‘2’ refers to vague regions described by a pair of geometry columns of type *Polygon*, i.e. *InfectedRegionCertitudeGeo* and *InfectedRegionDubietyGeo*;
- SB-index1 is the SB-index built on the columns *InfectionPK* and *InfectedRegionGeo* of the table *InfectedRegion1* shown in Figure 6.15, such that ‘1’ refers to vague regions described by a single geometry column of type *MultiPolygon*; and
- SB-index2 is the SB-index built on the columns *InfectionPK* and *InfectedRegionDubietyGeo* of the table *InfectedRegion2* shown in Figure 6.5b, such that ‘2’ refers to vague regions described by a pair of geometry columns of type *Polygon*, i.e. *InfectedRegionCertitudeGeo* and *InfectedRegionDubietyGeo*;

$OMBR_{OMIP5}CMBR_{CMIP5}$ is the configuration of the VSB-index that is able to identify answers of the spatial predicate already in the filter step, independently from the spatial predicate. $OMBR$ is the configuration of the VSB-index with the minimal set of approximations. aR-tree1 and SB-index1 have been already tested in Sections 6.4.3 and 6.4.4. aR-tree2 and SB-index2 are attempts to achieve a better performance by forcing the refinement step to be executed using polygons of the column *InfectedRegionDubietyGeo*, instead of multipolygons of the column *InfectedRegionGeo* as both the aR-tree1 and SB-index1 do. aR-tree2 and SB-index2 are also based on a previous result discussed in Section 6.2.2.3, which indicates that the separation of the certitude and the dubiety in distinct columns was beneficial for the performance.

Since both the SB-index and the aR-tree do not have algorithms to solve the $VSRQ_{object}$, adaptations were necessary and are detailed as follows. The SB-index has been extended as described in Section 6.2.3.3. As for the aR-tree, the principle adopted to extend it was to execute each sub-query separately and process their result sets. For each result r_i of the first sub-query and each result r_o of the second sub-query where $r_i.Team = r_o.Team$ and $r_i.Year = r_o.Year$, the value assigned to $r_o.sum$ was $r_o.sum - r_i.sum$. If the result of aforementioned subtraction was 0, then the row was simply removed from the result set. Finally, both result sets were merged.

6.4.5.2 Results

Figure 6.23 reports the average elapsed time spent by each configuration described in Section 6.4.5.1 according to different selectivities. Clearly, the configurations aR-tree2 and SB-index2 outperformed the configurations aR-tree1 and SB-index1, respectively. This fact corroborated that the resolution of spatial predicates against polygons was more efficient than

the resolution of spatial predicates against multipolygons, like a previous result involving the DBMS (Section 6.2.2.3). However, the existence of a costly refinement step still impaired the performance of both aR-tree2 and SB-index2. Also, increasing selectivities clearly increased the average elapsed times of the SB-index and the aR-tree. These indices were overcome by the configurations O_{MBR} and $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$ of the VSB-index.

In particular, the configuration $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$ of the VSB-index outperformed all the other tested configurations. Its MBR built on the vague region was used to identify answers of the predicate *containment* already in the filter step. Also, its MIP5 built on the vague region was used to identify answers of the predicate *intersects* already in the filter step. Then, these approximations drastically reduced the set of candidates that were subsequently processed in the refinement step. As a result, the elapsed time was shorter. Compared to the best result achieved by the aR-tree, the configuration $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$ of the VSB-index imposed a time reduction that ranged from 64% up to 85% over the configuration aR-tree2, for selectivities 0.001 and 0.03, respectively.

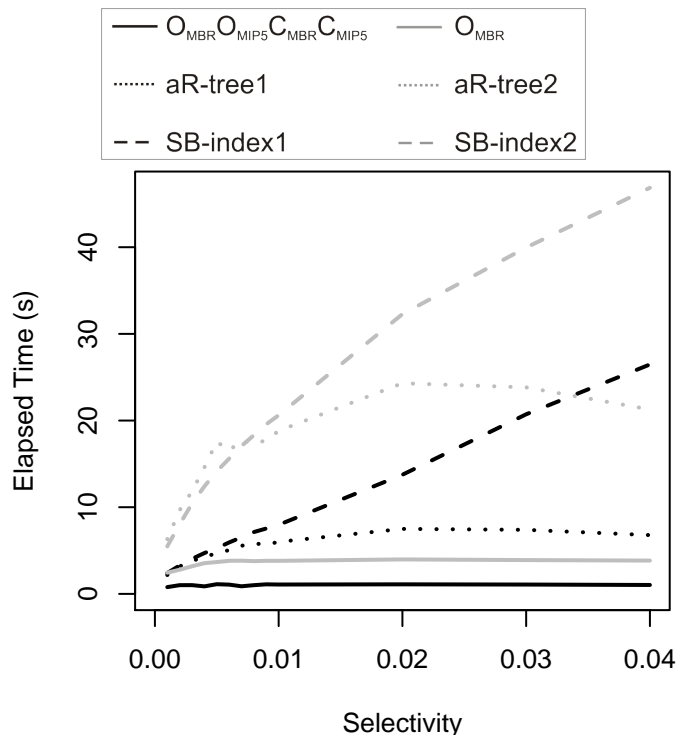


Figure 6.23: Average elapsed time to process VSRQobject over the synthetic vague SDW.

6.4.6 Building Costs and Storage Requirements

This section focus on the costs to build the indices described in Section 6.4.1.1 and their storage requirements. In addition, the storage requirements of those indices are detailed. Only

the synthetic vague SDW is addressed since it was more voluminous.

Figure 6.24 reports the elapsed time to build the sequential file of the VSB-index for the configurations O_{MBR} , $O_{MBR}C_{MBR}$, $O_{MBR}C_{MIP5}$, $O_{MBR}O_{MIP5}$ and $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$, since these configurations are discussed in Sections 6.4.2 to 6.4.4. The following measurements were also separated: (i) the time spent to extract the certitude of the vague region; (ii) the time spent to extract the outer boundary of the dubiety; and (iii) the time spent to build the approximations and write them on disk.

As for the VSB-index, configurations that held only MBRs were built in shorter time, i.e. O_{MBR} and $O_{MBR}C_{MBR}$. Also, the overhead to build the MIP5 using the outer boundary of the dubiety was significantly shorter than the overhead to build the MIP5 on the certitude. Then, the total time spent to build the configuration $O_{MBR}O_{MIP5}$ was shorter than to build the configurations $O_{MBR}C_{MIP5}$ and $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$. Recall that the certitude was created by applying a negative buffer on a real polygon, while the outer boundary of the dubiety was the convex hull of the same real polygon, as described in Section 6.4.1.1. Then, the certitude had more vertices than the outer boundary of the dubiety. Consequently, the high number of vertices of the certitude impaired the performance to create the MIP5 for configurations $O_{MBR}C_{MIP5}$ and $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$. Although the time spent to build the sequential file of the VSB-index was long, it significantly benefited the performance to process queries, as detailed in Sections 6.4.3 and 6.4.2.

The storage requirements of the sequential file for each configuration of the VSB-index are detailed in Figure 6.25. As expected, configurations that held more approximations also required more storage space. Considering that bitmap join indices occupied 3,200 MB, then the VSB-index' sequential file added from 0.25% up to 1.7% to storage requirements considering the configurations O_{MBR} and $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$, respectively.

The configuration O_{MBR} of the VSB-index and the SB-index demanded equivalent storage requirements due to their similar data structures. The sequential file of the configuration $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$ of the VSB-index was 5.18 times larger than the sequential file of the SB-index, the complete VSB-index configuration required a total of 3,268 MB while the SB-index occupied a total of 3,211 MB. In that case, the VSB-index added only 1.78% to the storage requirements.

The aR-tree required 17,011 MB of storage space, i.e. 11 MB for the tree and 17,000 MB for the multidimensional arrays. The configuration $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$ of the VSB-index demanded only 19.21% of the storage requirements of the aR-tree.

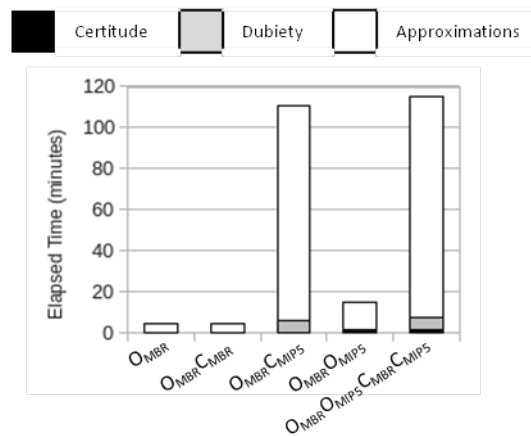


Figure 6.24: Time spent to build different configurations of the VSB-index.

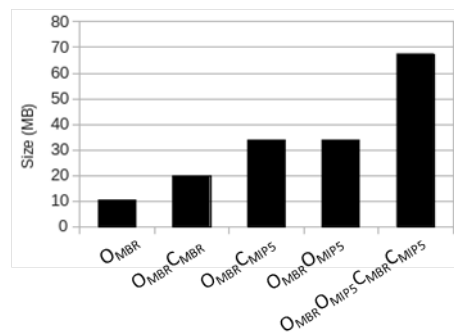


Figure 6.25: Storage requirements for different configurations of the VSB-index.

6.5 Summary

This chapter has focused on the physical design of vague SDWs. First, an experimental evaluation has identified that both the DBMS and existing indices for SDW had their storage requirements augmented and performance to process queries impaired by storing and querying vague point sets and vague regions. The increase on the complexity of vague point sets also increased both the storage requirements and the elapsed time to process queries with the vague spatial predicate CRQ_{Object} . Regarding vague regions, as the selectivity of the vague spatial predicate IRQ_{Object} increased, longer became the query response time and more costly became the resolution of such predicate, considering the DBMS, the aR-tree, and the SB-index. The latter also spent prohibitive time to process queries with the vague spatial predicate $VSQR_{Object}$ against vague regions, such that the refinement step was identified as being the more costly step to process such predicate. The results identified drawbacks and motivated the development of an index for vague SDW.

Second, the Vague Spatial Bitmap Index (VSB-index) has been proposed to efficiently

process vague spatial predicates of queries submitted to vague SDWs. The VSB-index has a flexible data structure that enables the use of conservative and progressive approximations created on the vague region O and on its certitude C . The progressive approximation MIP has been introduced and used by the VSB-index to reduce the cost of the refinement step in the vague spatial predicate resolution. It consists of a maximum area inscribed polygon with x vertices. Furthermore, the algorithms have been described to build the VSB-index and to process queries. The vague spatial predicates supported are IRQ_{object} , CRQ_{object} , $IRQ_{certitude}$, $CRQ_{certitude}$, $IRQ_{dubiety}$, $CRQ_{dubiety}$, and $VSRQ_{object}$.

Third, an experimental evaluation has been carried out to assess the VSB-index. Two distinct vague SDWs were loaded. One of the vague SDW was called real because vague regions were created based on a real dataset regarding the HLB case study. The other vague SDW was called synthetic since vague regions were created using a dataset that was not related to HLB. The vague SDWs had different characteristics, as follows. The synthetic vague SDW stored a huge volume of vague regions and was more voluminous than the real vague SDW. The vague regions of the synthetic vague SDW had relatively large certitudes, i.e. the certitude occupied a large portion of the extent of the vague region, while the the vague regions of the real vague SDW had relatively very small certitudes. Finally, the synthetic vague SDW had an average overlap rate among vague regions that was almost four times lesser than the real vague SDW.

Table 6.3 shows features of the vague SDWs used in the experiments and highlights the best configuration of the VSB-index according to each case investigated in the experimental evaluation. Independently from the vague SDW queried, the overlapping among vague regions existed and is an important aspect to be considered in the physical design of a vague SDW. The ratio between the area of the certitude and the area of the vague region is another relevant aspect that require attention in the physical design of a vague SDW. The experimental evaluation of the VSB-index using both the real vague SDW and the synthetic vague SDW enriched the analysis.

Intersection range queries were issued over the real vague SDW. As for IRQ_{object} and $IRQ_{dubiety}$, the configuration $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$ of the VSB-index imposed a time reduction that varied from 33% up to 69% over the SB-index. The MIP5 built on the vague region was essential for the configuration $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$ to achieve the best performance among the configurations of the VSB-index, because it drastically reduced the number of candidates produced in the filter step. These candidates were subsequently processed in the refinement step. In addition, the configuration $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$ of the VSB-index was not impaired by increasing selectivities. Analogously, the MIP5 built on the certitude was essential for the configuration $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$ to outperform the remaining configurations of the VSB-

Table 6.3: Summary of the results obtained with the experimental evaluation of the VSB-index.

Feature	Real Vague SDW	Synthetic Vague SDW
Quantity of Vague Regions	129	302,357
Overlapping among Vague Regions	AVG = 196%	AVG = 53%
Area(Certitude)/Area(Vague Region)	AVG = 1%	AVG = 78%
Vague Spatial Predicate	Best VSB-index	Best VSB-index
IRQ_{object}	$O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$	$O_{MBR}C_{MIP5}$
$IRQ_{dubiety}$	$O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$	$O_{MBR}C_{MIP5}$
$IRQ_{certitude}$	$O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$	$O_{MBR}C_{MIP5}$
CRQ_{object}	-	O_{MBR}
$CRQ_{dubiety}$	-	O_{MBR}
$CRQ_{certitude}$	-	$O_{MBR}C_{MBR}$
$VSRQ_{object}$	-	$O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$

index to process $IRQ_{certitude}$. To sum up, in the real vague SDW whose vague regions had a small certitude, the existence of MIP5 on the element being queried was essential to improve the performance to execute intersection range queries.

Intersection range queries were also issued over the synthetic vague SDW. As for IRQ_{object} and $IRQ_{dubiety}$, the configuration $O_{MBR}C_{MIP5}$ of the VSB-index imposed a time reduction that varied from 69% up to 97% over the best result achieved by either the SB-index or the aR-tree. Since the vague regions of the synthetic vague SDW had relatively large certitudes, the MIP5 built over the certitude offered by the configuration $O_{MBR}C_{MIP5}$ could efficiently identify answers of the vague spatial predicates already in the filter step. As a result, the cost of the refinement step became lower and the performance was benefited. Also, the configuration $O_{MBR}C_{MIP5}$ outperformed the configuration $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$, since the former required less disk accesses than the latter to process the filter step. For the same reason, the configuration $O_{MBR}C_{MIP5}$ outperformed the other configurations of the VSB-index to process $IRQ_{certitude}$. To sum up, in the synthetic vague SDW whose vague regions had a large certitude, the existence of MIP5 on the certitude was sufficient to improve the performance to execute intersection range queries.

Containment range queries, which do not require a refinement step, were issued over the synthetic vague SDW. As for the CRQ_{object} and $CRQ_{dubiety}$, the configuration O_{MBR} of the VSB-index imposed a time reduction that varied from 9% up to 44% over the aR-tree for selectivities greater than 0.01. Although the aR-tree overcame the VSB-index for selectivities lower than or equal to 0.01, the overlap among vague regions severely impaired the aR-tree for increasing selectivities. Also, the results obtained by the configuration O_{MBR} of VSB-index were equivalent to those achieved by the SB-index. As for the $CRQ_{certitude}$, the configuration $O_{MBR}C_{MBR}$

of the VSB-index performed less disk accesses and spent shorter time than the configuration $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$, since the former stored only two approximations and had an index entry size smaller than the latter. Both the SB-index and the aR-tree do not provide algorithms to process $CRQ_{certitude}$.

Vague spatial range queries, which requires the resolution of both *containment* and *intersects* may require a refinement step, were issued over the synthetic vague SDW. Existing indices for SDW have been extended to support $VSRQ_{object}$ and their refinement steps were also adapted to fetch simple geometries rather than complex geometries. The configuration $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$ of the VSB-index imposed a time reduction that ranged from 64% up to 85% over the best results achieved by an extended version of the aR-tree. The existence of MIP5 on the vague region greatly benefited the performance of the VSB-index.

The configuration $O_{MBR}O_{MIP5}C_{MBR}C_{MIP5}$ of the VSB-index added 1.78% to the storage requirements of the SB-index. In addition, it represented only 19.21% of the storage requirements of the aR-tree. The small amount of storage requirements and the benefits provided to the query processing performance motivate the use of the VSB-index in vague SDWs.

Chapter 7

CONCLUSION AND FUTURE WORK

Spatial data warehouses (SDWs) and spatial online analytical processing (SOLAP) are core technologies for decision support. Most approaches for designing SDWs represent spatial data as spatial objects that are crisp and assume definite extent, boundary, and shape, e.g., the territory of a city. However, spatial data are susceptible to imperfections such as spatial vagueness, which impairs distinguishing which components belong completely or partially to a spatial object, or definitely do not belong to the spatial object. Also, the spatial object has an extent, but it cannot or does not have a precisely definable boundary and/or interior. In other words, such spatial object cannot be rigorously bounded by a sharp line and might have a blurred interior. Since the design of SDWs typically considers crisp spatial objects and neglect imperfections, some types of decision support analysis cannot be carried out by SOLAP.

On the other hand, in this thesis, spatial vagueness has been considered to design and query SDWs. As a result, the vague spatial data warehouse (vague SDW) and the vague spatial online analytical processing (vague SOLAP) have been introduced. In addition, the following main contributions have been achieved. First, the Vague Spatial Cube conceptual model (VSCube), which enables the creation of conceptual schemata for vague SDWs using data cubes and provides vague spatial aggregation functions and vague spatial predicates to query vague spatial data as measures or in dimensions. Second, the Vague Spatial MultiDim conceptual model (VSMultiDim), which provides visual representations for the concepts and allows creating conceptual schemata for vague SDWs using diagrams. Third, guidelines for the logical design of relational schemata of vague SDWs, for elaborating their constraints, and for extending SQL to enable querying vague SDWs. Fourth, the Vague Spatial Bitmap Index (VSB-index), which improves the performance to process queries against vague regions stored in vague SDWs. In order to illustrate the applicability of the aforementioned contributions, two case studies have been described and used throughout this thesis to exemplify concepts and to emphasize the

diversity of applications that can be benefited from this work. Both the pest control and the HLB case studies refer to the agricultural domain, but latter intrinsically utilizes a real dataset (JORGE; INAMASU, 2014).

The major characteristics of the VSCube conceptual model are the following. Regarding SDW, it comprises attribute types, hierarchies and their categories, dimensions, measures, fact, cube and lattice of cuboids, for conventional data, non-geometric spatial data, crisp spatial data and, mainly, vague spatial data. Furthermore, the vague spatial fact enables the assignment of measure values in a fact to elements of the vague spatial member in a dimension. Concerning spatial vagueness, it supports vague spatial data modeled according to both exact models (i.e. geometric shapes and their membership *true* or *maybe*) and implementations for fuzzy models (i.e. geometric shapes and their corresponding membership values in $[0,1]$). In the context of spatial analysis and GIS, it defines vague spatial aggregation functions (vague spatial union, vague spatial intersection and vague spatial difference) and vague spatial predicates (spatial range query and vague spatial range query) that specify how to manipulate vague spatial data in queries. Regarding SOLAP, it defines vague SOLAP and the operations slice-and-dice, drill-down, roll-up, and pivot. The VSCube was described in a publication (SIQUEIRA et al., 2014).

Since the VSCube conceptual model does not offer graphic notations for the concepts, the VSMultiDim conceptual model has also been proposed to allow the creation of diagrams for representing the vague SDW conceptual schema. Regarding SDW, it comprises attributes, levels, members, dimensions, hierarchies, fact, measures and schema with support for conventional data, non-geometric spatial data, continuous fields, crisp spatial data and, mainly, vague spatial data. Concerning spatial vagueness, it supports vague spatial data modeled according to both exact models and fuzzy models (i.e. fuzzy spatial objects with a membership function assigning membership values in $[0,1]$). In the context of spatial analysis and GIS, it introduces pictograms to distinguish data types and to indicate topological constraints. The VSMultiDim was summarized in a publication (SIQUEIRA; CIFERRI; ZIMÁNYI, 2014) and reported (SIQUEIRA; ZIMÁNYI; CIFERRI, 2015).

The logical design of vague SDWs has been addressed as follows. Mapping rules have been described to transform a conceptual schema of vague SDW into a relational logical schema. These rules map vague spatial attributes defined in a level, associated through hierarchies, and denoting measures in a fact. The proposed implementation for the vague spatial attribute reuses spatial data types and multivalued (array) columns. These features are often implemented by existing DBMSs. Furthermore, SQL has been extended with operators and vague spatial predicates to enable querying the vague SDW. They have been implemented as user-defined functions

in the DBMS. Moreover, constraints have been specified to ensure the integrity of vague spatial data. These constraints check whether values of a vague spatial attribute are valid. They also ensure that valid topological relationships are held for members of same level and members related through either a hierarchy or a fact. The constraints have been implemented as user-defined functions and triggers. The contributions concerning the logical design of vague SDWs were published (SIQUEIRA et al., 2012a) and reported (SIQUEIRA; CIFERRI; ZIMÁNYI, 2014, 2015; SIQUEIRA; ZIMÁNYI; CIFERRI, 2015).

As for the physical design of vague SDWs, an experimental evaluation has identified the lack of an index for vague SDWs, since indices for SDW process vague spatial predicates with unacceptable performance. Then, the VSB-index has been introduced to efficiently process multidimensional queries whose vague spatial predicates involve vague regions. The VSB-index resolves a subset of the vague spatial predicates defined by the VSCube conceptual model. The supported intersection range queries, containment range queries and vague spatial range queries focus on whole vague regions and on their components (i.e. certitude and dubiety). The VSB-index has a data structure with both conservative and progressive approximations. They are used in the multi-step resolution of the vague spatial predicate, which comprises filter step and refinement step. The progressive approximation MIP, which is a maximum area inscribed polygon, allows the identification of answers of the vague spatial predicate already in the filter step. Such identification aims at reducing the number of candidates produced in the filter step and provided to the refinement step. Consequently, it aims at decreasing the cost of the refinement step.

The VSB-index was assessed through an experimental evaluation using two different vague SDWs and increasing selectivities for the vague spatial predicates. Results revealed that the VSB-index improved the performance to resolve vague spatial predicates and imposed a time reduction that ranged from 9% up to 97% over existing indices for SDWs. Furthermore, the VSB-index required a small amount of additional storage space if compared to existing indices for SDWs. The results achieved by the VSB-index in the experimental evaluation corroborated its utilization in vague SDWs. The contributions concerning the physical design of vague SDWs were also published (SIQUEIRA et al., 2011, 2013, 2014).

In addition to provide original contributions for the design of vague SDWs, this thesis also motivates the development of the following future work. Regarding the conceptual design of vague SDWs, the combination of both the VSCube and the VSMultiDim into a single cohesive model demands the following steps. The core issue is to define levels and enable hierarchies of levels (like the VSMultiDim model and Vaisman & Zimányi (2014b)) rather than hierarchies

of attributes (like the VSCube model and Golfarelli, Maio & Rizzi (1998)) to explicitly model entity types (or classes). Thus, dimensions and cuboids should be expressed in terms of levels rather than attributes. A challenging issue to be investigated in this context is that a hierarchy association with cardinality M:N requires distributing measure values between the related members (MALINOWSKI; ZIMÁNYI, 2009). This issue has not been tackled for vague spatial data either in members or measures.

Another future goal is to enhance multidimensional modeling with more types of attributes in dimensions (and their hierarchies) and as measures in a fact, and then provide novel vague SOLAP operations. The inclusion of vague spatial objects whose shapes change over a period of time into the data cube demands an investigation of temporal aspects influencing vague spatial attributes (HAZARIKA; COHN, 2001). For instance, valid time (the period in which a statement is true according to the user) and lifespan (the time during which an object exists) could be investigated (MALINOWSKI; ZIMÁNYI, 2009; ELMASRI; NAVATHE, 2010). The former could be applied to the vague spatial attribute, while the latter could be applied to the vague spatial level. In addition, the inclusion of vague non-geometric spatial attributes require fuzzy adverbial labels to indicate proximity or reference for locations, e.g. “very close to the university campus” can be explored.

The vague spatial attribute defined by the VSCube model can be extended with valid time and become a time-varying vague spatial attribute. Hence, in addition to the multivalued *certitude* and *dubiety*, the attribute could hold a composite *valid time* field defined by *initial* and *final* monovalued and atomic fields to denote delimiters of an interval. VSCube model’s accessors, vague spatial predicates, and vague spatial aggregation functions should also be extended to filter and aggregate based on valid time. The aforementioned transformation influences the logical design. In addition to the three columns used to represent a vague spatial attribute, two more columns namely *initial* and *final* are necessary to represent a time-varying vague spatial attribute. An extension of the VSB-index could allow the creation of approximations for multiple (rather than one) vague regions per object. Note that *valid time* could also be multivalued to denote a set of intervals, thus increasing the complexity of the design.

Concerning the logical design of vague SDWs, an indication of future work is to perform an extensive experimental evaluation of alternative implementations of vague spatial attribute. Such evaluation should involve a variety of (vague and fuzzy) spatial data types, increasing data volumes, diversified vague spatial predicates with different selectivities, and vague spatial aggregation. The analysis of the results will provide relevant findings that can aid the designer to select a representation based on the expected performance to process queries. In the same sense,

the logical design of vague SDWs can be extended to and experimented with non-relational databases. Besides, the vague spatial fact can be hereafter improved to enable roll-up and drill-down, i.e. to aggregate values of measures associated to elements of vague spatial members from levels in a hierarchy. Constraints will be essential to ensure correct aggregation of partial measure values and can be implemented as routines in the DBMS.

As for the physical design, the VSB-index can be hereafter enhanced, as follows. First, by enabling the filter step of intersection range queries to identify answers by assessing whether containment is true against the conservative approximation. Second, by processing vague SO-LAP operations, e.g. roll-up and drill-down. Third, by resolving other (vague) spatial predicates, e.g. the remaining vague spatial predicates of the VSCube conceptual model, point query, nearest neighbor queries, and spatial join. Fourth, by indexing other types of vague spatial data, e.g. vague regions implemented as plateau regions (SCHNEIDER, 2014), vague points, and vague lines. And fifth, by adapting its data structure and algorithms to efficiently process queries over vague SDWs hosted in a cloud, analogously to a previous published work that addressed SDWs with crisp spatial data (MATEUS et al., 2015).

Another indication of future work comprises design and implementation of a vague SO-LAP tool that reuses the contributions supplied by this thesis, as follows. The tool can reuse the VSCube conceptual model and omit details of the data cube from the user, as well as present the graphic notation of the VSMultiDim conceptual model to allow the creation of diagrams. Furthermore, mapping rules can be implemented to allow the implicit transformation of vague SDW's conceptual schema into logical relational schemata, as well as to implicitly create constraints. Moreover, the VSB-index can be reused to efficiently process queries. The development of the tool can also benefit from reusing and extending legacy of existing tools, i.e. BJIn OLAP Tool¹. (CARNIEL; SIQUEIRA, 2011b, 2011a, 2012), MapQuery (BIANCHI; HATANO; SIQUEIRA, 2013) and Mobile SpOT (BIANCHI; SIQUEIRA, 2013). A further investigation on vague spatial data display and visualization, and on map generalization is also required.

Since spatial vagueness is one imperfection of spatial data, the design of SDWs characterized by other imperfections also motivates future work. For instance, designing data cubes with spatial data with probabilistic uncertainty and statistically estimated precision (LI et al., 2007; TIMKO; DYRESON; PEDERSEN, 2014), which intrinsically avoid fuzzy concepts. In this regard, address spatial data whose quantitative values, probabilistic precisions and standard deviation are known, and whose precision for position of vertices is known and consist of ellipses of errors (DEVILLERS et al., 2010).

¹Intellectual property rights registered by National Industrial Property Institute (INPI), Brazil. Type: Registered Software. Registration number: BR 50 2013 000063-0.

REFERENCES

- AGGARWAL, A. et al. Geometric applications of a matrix-searching algorithm. *Algorithmica*, Springer-Verlag, v. 2, n. 2, p. 195–208, 1987. ISSN 0178-4617. Available at: <<http://dx.doi.org/10.1007/BF01840359>>.
- AOKI, P. M. Generalizing search in generalized search trees. In: *14th International Conference on Data Engineering (ICDE'1998)*. Orlando, FL, USA: IEEE Computer Society, 1998. (Proceedings...), p. 380–389. ISSN 1063-6382.
- BADARD, T.; DUBÉ, E. Enabling geospatial business intelligence. *Open Source Business Resource*, Talent First Network, Ottawa, September 2009. ISSN 1913-6102. Available at: <<http://timreview.ca/article/289>>.
- BALTZER, O.; RAU-CHAPLIN, A.; ZEH, N. Building a scalable spatial OLAP system. In: *28th ACM Symposium on Applied Computing (SAC'2013)*. Coimbra, Portugal: ACM, 2013. (Proceedings...), p. 13–15. ISBN 978-1-4503-1656-9. Available at: <<http://doi.acm.org/10.1145/2480362.2480366>>.
- BAYER, R.; MCCREIGHT, E. Organization and maintenance of large ordered indexes. *Acta Informatica*, Springer-Verlag, v. 1, n. 3, p. 173–189, 1972. ISSN 0001-5903. Available at: <<http://dx.doi.org/10.1007/BF00288683>>.
- BECKMANN, N. et al. The R*-tree: An efficient and robust access method for points and rectangles. *SIGMOD Record*, ACM, New York, NY, USA, v. 19, n. 2, p. 322–331, May 1990. ISSN 0163-5808.
- BÉDARD, Y. Uncertainties in land information systems databases. In: *Eighth International Symposium on Computer-Assisted Cartography*. Baltimore, MD, USA: American Society for Photogrammetry and Remote Sensing and American Congress on Surveying and Mapping, 1987. (Proceedings...), p. 175–184.
- BÉDARD, Y.; MERRETT, T.; HAN, J. Fundamentals of spatial data warehousing for geographic knowledge discovery. In: MILLER, H. J.; HAN, J. (Ed.). *Geographic Data Mining and Knowledge Discovery*. Bristol, PA, USA: Taylor & Francis, 2001. p. 53–73. ISBN 0415233690.
- BEJAOUI, L. *Qualitative topological relationships for objects with possibly vague shapes: implications on the specification of topological integrity constraints in transactional spatial databases and in spatial data warehouses*. Thesis (Ph.D.) — Université Blaise Pascal Clermont-Ferrand II (France); Université Laval, Québec (Canada), May 2009. Available at: <<http://tel.archives-ouvertes.fr/tel-00725614>>.

- BEJAOU, L. et al. Qualified topological relations between spatial objects with possible vague shape. *International Journal of Geographical Information Science*, Taylor & Francis, Inc., Bristol, PA, USA, v. 23, n. 7, p. 877–921, July 2009. ISSN 1365-8816. Available at: <<http://dx.doi.org/10.1080/13658810802022814>>.
- BEJAOU, L. et al. OCL for formal modelling of topological constraints involving regions with broad boundaries. *GeoInformatica*, Springer US, v. 14, n. 3, p. 353–378, 2010. ISSN 1384-6175. Available at: <<http://dx.doi.org/10.1007/s10707-010-0104-5>>.
- BEZDEK, J. C.; EHRLICH, R.; FULL, W. FCM: The fuzzy c-means clustering algorithm. *Computers & Geosciences*, v. 10, n. 2, p. 191 – 203, 1984. ISSN 0098-3004. Available at: <<http://www.sciencedirect.com/science/article/pii/0098300484900207>>.
- BIANCHI, R. G.; HATANO, G. Y.; SIQUEIRA, T. L. L. On the performance and use of spatial OLAP tools. In: *XXXIX Latin American Computing Conference (CLEI'2013)*. Caracas (Naguatá), Venezuela: IEEE, 2013. (Proceedings...), p. 1–12. ISBN 978-1-4799-2957-3. Available at: <<http://dx.doi.org/10.1109/CLEI.2013.6670652>>.
- BIANCHI, R. G.; SIQUEIRA, T. L. L. Mobile SpOT: uma ferramenta SOLAP para dispositivos móveis. In: *Anais de Eventos da UFSCar*. UFSCar, 2013. V Congresso de Iniciação em Desenvolvimento Tecnológico e Inovação da UFSCar (CIDTI'2013). Available at: <http://www.eventweb.com.br/jornada2013-cict/specific-files/manuscripts/index.php?file=jornada2013-cict/22217_1379107463.pdf>.
- BIMONTE, S.; KANG, M.-A. Towards a model for the multidimensional analysis of field data. In: CATANIA, B.; IVANOVIC, M.; THALHEIM, B. (Ed.). *14th East European Conference on Advances in Databases and Information Systems (ADBIS'2010)*. Novi Sad, Serbia: Springer Berlin Heidelberg, 2010, (Lecture Notes in Computer Science: Advances in Databases and Information Systems, v. 6295). p. 58–72. ISBN 978-3-642-15575-8. Available at: <http://dx.doi.org/10.1007/978-3-642-15576-5_7>.
- BIMONTE, S.; TCHOUNIKINE, A.; MIQUEL, M. Towards a spatial multidimensional model. In: *8th ACM International Workshop on Data Warehousing and OLAP (DOLAP'2005)*. Bremen, Germany: ACM, 2005. (Proceedings...), p. 39–46. ISBN 1-59593-162-7. Available at: <<http://doi.acm.org/10.1145/1097002.1097009>>.
- BORGIDA, A.; CASANOVA, M. A.; LAENDER, A. H. Logical database design: from conceptual to logical schema. In: LIU, L.; ÖZSU, M. T. (Ed.). *Encyclopedia of Database Systems*. New York, NY, USA: Springer US, 2009. p. 1645–1649. ISBN 978-0-387-39940-9.
- BORGIDA, A.; MYLOPOULOS, J. Conceptual schema design. In: LIU, L.; ÖZSU, M. T. (Ed.). *Encyclopedia of Database Systems*. New York, NY, USA: Springer US, 2009. p. 438–442. ISBN 978-0-387-39940-9.
- BOULIL, K. et al. Towards the definition of spatial data warehouses integrity constraints with spatial ocl. In: *13th ACM International Workshop on Data Warehousing and OLAP (DOLAP'2010)*. Toronto, Ontario, Canada: ACM, 2010. (Proceedings...), p. 31–36. ISBN 978-1-4503-0383-5. Available at: <<http://doi.acm.org/10.1145/1871940.1871948>>.

- BOULIL, K.; BIMONTE, S.; PINET, F. Conceptual model for spatial data cubes: A UML profile and its automatic implementation. *Computer Standards & Interfaces*, v. 38, p. 113 – 132, February 2015. ISSN 0920-5489. Available at: <<http://www.sciencedirect.com/science/article/pii/S0920548914000774>>.
- BOVÉ, J. M. Huanglongbing or yellow shoot, a disease of gondwanan origin: Will it destroy citrus worldwide? *Phytoparasitica*, Springer Netherlands, v. 42, n. 5, p. 579–583, 2014. ISSN 0334-2123. Available at: <<http://dx.doi.org/10.1007/s12600-014-0415-4>>.
- BRINKHOFF, T.; KRIEGEL, H.-P.; SCHNEIDER, R. Comparison of approximations of complex objects used for approximation-based query processing in spatial database systems. In: *9th International Conference on Data Engineering (ICDE'1993)*. Vienna, Austria: IEEE Computer Society, 1993. (Proceedings...), p. 40–49. ISBN 0-8186-3570-3.
- BURDICK, D. et al. OLAP over uncertain and imprecise data. In: *31st International Conference on Very Large Data Bases (VLDB'2005)*. Trondheim, Norway: ACM, 2005. (Proceedings...), p. 970–981. ISBN 1-59593-154-6.
- BURDICK, D. et al. OLAP over uncertain and imprecise data. *The VLDB Journal*, Springer-Verlag, v. 16, n. 1, p. 123–144, 2007. ISSN 1066-8888. Available at: <<http://dx.doi.org/10.1007/s00778-006-0033-y>>.
- BURROUGH, P. Natural objects with indeterminate boundaries. In: BURROUGH, P. A.; FRANK, A. (Ed.). *Geographic objects with indeterminate boundaries*. 1. ed. London, UK: Taylor & Francis, 1996, (GISDATA 2, v. 2). Chapter 1, p. 3–28. ISBN 978-0748403868.
- BURROUGH, P. A.; FRANK, A. *Geographic Objects with Indeterminate Boundaries*. 1. ed. London, UK: Taylor & Francis, 1996. (GISDATA 2, v. 2). ISBN 978-0748403875.
- CÂMARA, G. et al. Fields as a generic data type for big spatial data. In: DUCKHAM, M. et al. (Ed.). *8th International Conference on Geographic Information Science (GIScience'2014)*. Springer International Publishing, 2014, (Lecture Notes in Computer Science: Geographic Information Science, v. 8728). p. 159–172. ISBN 978-3-319-11592-4. Available at: <http://dx.doi.org/10.1007/978-3-319-11593-1_11>.
- CÂMARA, G.; FREITAS, U.; CASANOVA, M. A. Fields and objects algebras for GIS operations. In: *3rd Brazilian Symposium on GIS*. São Paulo, SP, Brazil: [s.n.], 1995. (Proceedings...), p. 407–424. Available at: <http://www.dpi.inpe.br/gilberto/papers/premio_compaq.pdf>.
- CARNIEL, A. C.; SIQUEIRA, T. L. L. An OLAP tool based on the bitmap join index. In: *XXXVII Conferencia Latinoamericana en Informatica (CLEI'2011)*. Quito, Ecuador: Centro Latinoamericano de Estudios en Informática, 2011. (Anais...), p. 911–926.
- CARNIEL, A. C.; SIQUEIRA, T. L. L. The Bitmap Join Index OLAP Tool. In: *XXVI Brazilian Symposium on Database (SBB'D'2011)*. Florianópolis, SC, Brazil: Brazilian Computer Society, 2011. (Proceedings...), p. 13–18. Demos Session.
- CARNIEL, A. C.; SIQUEIRA, T. L. L. Querying data warehouses efficiently using the Bitmap Join Index OLAP Tool. *CLEI Electronic Journal*, v. 15, n. 2, 2012. Available at: <<http://www.clei.cl/cleiej/paper.php?id=243>>.

- CAZZIN, G. et al. *Business Intelligence with SpagoBI*. Padua, Italy: SpagoBI Competency Center, 2012.
- CHAUDHURI, S.; DAYAL, U. An overview of data warehousing and OLAP technology. *SIGMOD Record*, ACM, New York, NY, USA, v. 26, n. 1, p. 65–74, mar 1997. ISSN 0163-5808. Available at: <<http://doi.acm.org/10.1145/248603.248616>>.
- CHEN, P. P. The Entity-Relationship model - toward a unified view of data. *Transactions on Database Systems*, ACM, v. 1, n. 1, p. 9–36, 1976. Available at: <<http://doi.acm.org/10.1145/320434.320440>>.
- CHENG, R.; KALASHNIKOV, D. V.; PRABHAKAR, S. Evaluating probabilistic queries over imprecise data. In: *ACM SIGMOD International Conference on Management of Data (SIGMOD'2003)*. San Diego, California: ACM, 2003. (Proceedings...), p. 551–562. ISBN 1-58113-634-X. Available at: <<http://doi.acm.org/10.1145/872757.872823>>.
- CIFERRI, C. D. A. et al. Cube algebra: A generic user-centric model and query language for OLAP cubes. *International Journal of Data Warehousing and Mining*, v. 9, n. 2, p. 39–65, 2013. Available at: <<http://dx.doi.org/10.4018/jdwm.2013040103>>.
- CIFERRI, R. R. *Análise da Influência do Fator Distribuição Espacial dos Dados no Desempenho de Métodos de Acesso Multidimensionais*. Thesis (Ph.D.) — Universidade Federal de Pernambuco (Brasil), Fevereiro 2002.
- CLEMENTINI, E.; FELICE, P. D. An algebraic model for spatial objects with indeterminate boundaries. In: BURROUGH, P. A.; FRANK, A. (Ed.). *Geographic objects with indeterminate boundaries*. 1. ed. London, UK: Taylor & Francis, 1996, (GISDATA 2, v. 2). Chapter 11, p. 155–169. ISBN 978-0748403868.
- CLEMENTINI, E.; SHARMA, J.; EGENHOFER, M. J. Modelling topological spatial relations: Strategies for query processing. *Computers & Graphics*, v. 18, n. 6, p. 815 – 822, 1994. ISSN 0097-8493. Available at: <<http://www.sciencedirect.com/science/article/pii/0097849394900078>>.
- CODD, E. F. A relational model of data for large shared data banks. *Communications of the ACM*, ACM, New York, NY, USA, v. 13, n. 6, p. 377–387, jun 1970. ISSN 0001-0782.
- CODD, E. F.; CODD, S. B.; SALLEY, C. T. Providing OLAP to user-analysts: An IT mandate. *E.F. Codd and Associates*, 1993.
- COHN, A.; GOTTS, N. The ‘Egg-Yolk’ representation of regions with indeterminate boundaries. In: BURROUGH, P. A.; FRANK, A. (Ed.). *Geographic objects with indeterminate boundaries*. 1. ed. London, UK: Taylor & Francis, 1996, (GISDATA 2, v. 2). Chapter 12, p. 171–187. ISBN 978-0748403868.
- CUZZOCREA, A.; FIDALGO, R. N. Enhancing coverage and expressive power of spatial data warehousing modeling: the SDWM approach. In: *14th International Conference on Data Warehousing and Knowledge Discovery (DaWaK'2012)*. Vienna, Austria: Springer-Verlag, 2012, (Lecture Notes in Computer Science: Data Warehousing and Knowledge Discovery, v. 7448). p. 15–29. ISBN 978-3-642-32583-0. Available at: <http://dx.doi.org/10.1007/978-3-642-32584-7_2>.

DALVI, N. N.; SUCIU, D. Efficient query evaluation on probabilistic databases. In: NASCIMENTO, M. A. et al. (Ed.). *30th International Conference on Very Large Data Bases (VLDB'2004)*. Toronto, Canada: ACM, 2004. (Proceedings...), p. 864–875. ISBN 0-12-088469-0.

DELGADO, M. et al. F-CUBE Factory: A fuzzy OLAP system for supporting imprecision. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, v. 15, n. supp01, p. 59–81, 2007. Available at: <<http://www.worldscientific.com/doi/abs/10.1142/S0218488507004467>>.

DELGADO, M. et al. A fuzzy multidimensional model for supporting imprecision in olap. In: *2004 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'2004)*. Budapest, Hungary: [s.n.], 2004. (Proceedings..., v. 3), p. 1331–1336. ISSN 1098-7584.

DEVILLERS, R. et al. Thirty years of research on spatial data quality: Achievements, failures, and opportunities. *Transactions in GIS*, Blackwell Publishing Ltd, v. 14, n. 4, p. 387–400, 2010. ISSN 1467-9671. Available at: <<http://dx.doi.org/10.1111/j.1467-9671.2010.01212.x>>.

DILO, A. *Representation of and reasoning with vagueness in spatial information : a system for handling vague objects*. Thesis (Ph.D.) — Wageningen University (Netherlands), June 2006. Available at: <<http://edpot.wur.nl/121810>>.

DILO, A. et al. Storage and manipulation of vague spatial objects using existing gis functionality. In: BORDOGNA, G.; PSAILA, G. (Ed.). *Flexible Databases Supporting Imprecision and Uncertainty*. Springer Berlin Heidelberg, 2006, (Studies in Fuzziness and Soft Computing, v. 203). p. 293–321. ISBN 978-3-540-33289-3. Available at: <http://dx.doi.org/10.1007/3-540-33289-8_12>.

DILO, A.; BY, R. D.; STEIN, A. A system of types and operators for handling vague spatial objects. *International Journal of Geographical Information Science*, Taylor & Francis, v. 21, n. 4, p. 397–426, 2007. Available at: <<http://dx.doi.org/10.1080/13658810601037096>>.

DUBOIS, D.; PRADE, H. The three semantics of fuzzy sets. *Fuzzy Sets and Systems*, v. 90, n. 2, p. 141–150, 1997. ISSN 0165-0114. Available at: <<http://www.sciencedirect.com/science/article/pii/S0165011497000808>>.

DUBOIS, D.; PRADE, H. The legacy of 50 years of fuzzy sets: A discussion. *Fuzzy Sets and Systems*, v. 281, p. 21–31, 2015. ISSN 0165-0114. Special Issue Celebrating the 50th Anniversary of Fuzzy Sets. Available at: <<http://www.sciencedirect.com/science/article/pii/S0165011415004169>>.

DYRESON, C. E. A bibliography on uncertainty management in information systems. In: MOTRO, A.; SMETS, P. (Ed.). *Uncertainty Management in Information Systems*. Springer US, 1997. Chapter 15, p. 413–458. ISBN 978-1-4613-7865-5. Available at: <http://dx.doi.org/10.1007/978-1-4615-6245-0_15>.

EDELSBRUNNER, H.; KIRKPATRICK, D.; SEIDEL, R. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, v. 29, n. 4, p. 551–559, July 1983. ISSN 0018-9448.

EDOH-ALOVE, É.; BIMONTE, S.; BÉDARD, Y. A new design method for managing spatial vagueness in classical relational spatial OLAP architectures. In: MURGANTE, B. et al. (Ed.). *14th International Conference on Computational Science and Its Applications (ICCSA'2014)*. Guimarães, Portugal: Springer International Publishing, 2014, (Lecture Notes in Computer Science: Computational Science and Its Applications, v. 8580). p. 774–786. ISBN 978-3-319-09128-0. Available at: <http://dx.doi.org/10.1007/978-3-319-09129-7_56>.

EDOH-ALOVE, É. et al. A hybrid risk-aware design method for spatial datacubes handling spatial vague data: implementation and validation. *International Journal of Business Intelligence and Data Mining*, v. 9, n. 3, p. 210–232, 2014. Available at: <<http://dx.doi.org/10.1504/IJBIDM.2014.068366>>.

EGENHOFER, M.; HERRING, J. *Categorizing Binary Topological Relationships Between Regions, Lines, and Points in Geographic Databases*. Orono, ME, USA: Department of Surveying Engineering, University of Maine, 1991. 28 p.

EGENHOFER, M. J.; FRANZOSA, R. D. Point set topological relations. *International Journal of Geographical Information Systems*, v. 5, n. 2, p. 161–174, 1991. Available at: <<http://dx.doi.org/10.1080/02693799108927841>>.

ELMASRI, R.; NAVATHE, S. B. *Fundamentals of Database Systems*. 6. ed. Columbus, OH, USA: Addison Wesley, 2010. Hardcover. ISBN 0136086209.

EMBLEY, D. W. Relational model. In: LIU, L.; ÖZSU, M. T. (Ed.). *Encyclopedia of Database Systems*. New York, NY, USA: Springer US, 2009. p. 2372–2376. ISBN 978-0-387-39940-9.

EMBLEY, D. W. Semantic data model. In: LIU, L.; ÖZSU, M. T. (Ed.). *Encyclopedia of Database Systems*. New York, NY, USA: Springer US, 2009. p. 2559–2561. ISBN 978-0-387-39940-9.

ERWIG, M.; SCHNEIDER, M. Vague regions. In: SCHOLL, M.; VOISARD, A. (Ed.). *5th International Symposium on Advances in Spatial Databases (SSD'1997)*. Berlin, Germany: Springer Berlin Heidelberg, 1997, (Lecture Notes in Computer Science: Advances in Spatial Databases, v. 1262). p. 298–320. ISBN 978-3-540-63238-2. Available at: <http://dx.doi.org/10.1007/3-540-63238-7_36>.

FASEL, D. *Fuzzy Data Warehousing for Performance Measurement: Concept and Implementation*. 1. ed. [S.l.]: Springer International Publishing, 2014. (Fuzzy Management Methods). ISBN 978-3-319-04226-8.

FIDALGO, R. N. et al. GeoDWFrame: A framework for guiding the design of geographical dimensional schemas. In: KAMBAYASHI, Y.; MOHANIA, M.; WOESS, W. (Ed.). *6th International Conference on Data Warehousing and Knowledge Discovery (DaWaK'2004)*. Zaragoza, Spain: Springer Berlin Heidelberg, 2004, (Lecture Notes in Computer Science: Data Warehousing and Knowledge Discovery, v. 3181). p. 26–37. ISBN 978-3-540-22937-7. Available at: <http://dx.doi.org/10.1007/978-3-540-30076-2_3>.

FISHER, P. F. Models of uncertainty in spatial data. In: LONGLEY, P. et al. (Ed.). *Geographical Information systems*. 2. ed. New York, NY, USA: John Wiley & Sons, 1999. v. 1, Chapter 13, p. 191–205. ISBN 0471-33132-5.

- GAEDE, V.; GUNTHER, O. Multidimensional access methods. *ACM Computing Surveys*, ACM, v. 30, n. 2, p. 170–231, 1998. Available at: <<http://dx.doi.org/10.1145/280277.280279>>.
- GASCUEÑA, C. M.; GUADALUPE, R. A multidimensional methodology with support for spatio-temporal multigranularity in the conceptual and logical phases. In: TANIAR, D. (Ed.). *Progressive Methods in Data Warehousing and Business Intelligence: Concepts and Competitive Analytics*. Hershey, PA, USA: IGI Global, 2009. Chapter 10, p. 194–230.
- GOGOLLA, M. Object constraint language. In: LIU, L.; ÖZSU, M. T. (Ed.). *Encyclopedia of Database Systems*. New York, NY, USA: Springer US, 2009. p. 1927–1929. ISBN 978-0-387-39940-9.
- GOGOLLA, M. Unified modeling language. In: LIU, L.; ÖZSU, M. T. (Ed.). *Encyclopedia of Database Systems*. New York, NY, USA: Springer US, 2009. p. 3232–3239. ISBN 978-0-387-39940-9.
- GOLFARELLI, M.; MAIO, D.; RIZZI, S. The dimensional fact model: A conceptual model for data warehouses. *International Journal of Cooperative Information Systems*, v. 07, n. 02n03, p. 215–247, June and September 1998. Available at: <<http://www.worldscientific.com/doi/abs/10.1142/S0218843098000118>>.
- GOMEZ, L. et al. Spatial aggregation: Data model and implementation. *Information Systems*, v. 34, n. 6, p. 551 – 576, 2009. ISSN 0306-4379. Available at: <<http://www.sciencedirect.com/science/article/pii/S0306437909000131>>.
- GOODCHILD, M. F. Geographical data modeling. *Computers & Geosciences*, v. 18, n. 4, p. 401 – 408, 1992. ISSN 0098-3004. GIS Design Models. Available at: <<http://www.sciencedirect.com/science/article/pii/0098300492900694>>.
- GOODCHILD, M. F. Attribute accuracy. In: GUPTILL, S. C.; MORRISON, J. L. (Ed.). *Elements of Spatial Data Quality*. Pergamon, 1995, (International Cartographic Association). Chapter 4, p. 59 – 79. ISBN 978-0-08-042432-3. Available at: <<http://www.sciencedirect.com/science/article/pii/B9780080424323500112>>.
- GOODCHILD, M. F. Imprecision and spatial uncertainty. In: *Encyclopedia of GIS*. [S.l.]: Springer US, 2008. p. 480–483. ISBN 978-0-387-30858-6.
- GOTTWALD, T. R. Current epidemiological understanding of citrus huanglongbing. *Annual Review of Phytopathology*, v. 48, n. 1, p. 119–139, 2010. PMID: 20415578. Available at: <<http://dx.doi.org/10.1146/annurev-phyto-073009-114418>>.
- GOTTWALD, T. R.; GRAÇA, J. V.; BASSANEZI, R. B. Citrus huanglongbing: The pathogen and its impact. *Plant Health Progress*, September 2007. ISSN 1535-1025.
- GUTTMAN, A. R-trees: A dynamic index structure for spatial searching. *SIGMOD Record*, ACM, New York, NY, USA, v. 14, n. 2, p. 47–57, June 1984. ISSN 0163-5808. Available at: <<http://doi.acm.org/10.1145/971697.602266>>.
- HARINARAYAN, V.; RAJARAMAN, A.; ULLMAN, J. D. Implementing data cubes efficiently. *SIGMOD Record*, ACM, New York, NY, USA, v. 25, n. 2, p. 205–216, June 1996. ISSN 0163-5808. Available at: <<http://doi.acm.org/10.1145/235968.233333>>.

- HAZARIKA, D.; HAZARIKA, D. Fuzzy regions with holes and their topological relations in a special fuzzy topological space. *Annals of Fuzzy Mathematics and Informatics*, v. 3, n. 1, p. 89–101, 2012. ISSN 20939310.
- HAZARIKA, S. M.; COHN, A. G. A taxonomy for spatial vagueness: An alternative Egg-Yolk interpretation. In: *Workshop on Spatial Vagueness, Uncertainty and Granularity (SVUG'01)*. Ogunquit, ME, USA: [s.n.], 2001. p. 92–107.
- HERRING, J. R. (Ed.). *OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture*. [S.l.], 2011. Available at: <<http://www.opengeospatial.org/>>.
- HOEL, E. Data models in commercial GIS systems. In: *Encyclopedia of GIS*. Springer US, 2008. p. 215–219. ISBN 978-0-387-30858-6. Available at: <http://dx.doi.org/10.1007/978-0-387-35973-1_247>.
- HWANG, S.; THILL, J.-C. Modeling localities with fuzzy sets and gis. In: PETRY, F. E.; ROBINSON, V. B.; COBB, M. A. (Ed.). *Fuzzy Modeling with Spatial Information for Geographic Problems*. New York, NY, USA: Springer Berlin Heidelberg, 2005. p. 71–104. ISBN 978-3-540-23713-6. Available at: <http://dx.doi.org/10.1007/3-540-26886-3_4>.
- INAMASU, R. Y. et al. (Ed.). *Agricultura de Precisão: Um Novo Olhar*. 1. ed. Embrapa Instrumentação, 2011. ISBN 9788586463310. Available at: <<http://www.macroprograma1.cnptia.embrapa.br/redeap2/publicacoes/publicacoes-da-rede-ap/capitulos>>.
- JADIDI, A. et al. Using geospatial business intelligence paradigm to design a multidimensional conceptual model for efficient coastal erosion risk assessment. *Journal of Coastal Conservation*, Springer, v. 17, n. 3, p. 527–543, 2013. ISSN 1400-0350. Available at: <<http://dx.doi.org/10.1007/s11852-013-0252-5>>.
- JADIDI, A. et al. Spatial representation of coastal risk: A fuzzy approach to deal with uncertainty. *International Journal of Geo-Information (ISPRS)*, MDPI, v. 3, n. 3, p. 1077–1100, 2014. ISSN 2220-9964. Available at: <<http://dx.doi.org/10.3390/ijgi3031077>>.
- JENKINS, D. A.; HALL, D. G.; GOENAGA, R. Diaphorina citri (hemiptera: Liviidae) abundance in Puerto Rico declines with elevation. *Journal of Economic Entomology*, The Oxford University Press, 2015. ISSN 0022-0493.
- JORGE, L. A. C.; INAMASU, R. Y. Detecção de greening dos citrus por imagens multiespectrais. In: BERNARDI, A. C. C. et al. (Ed.). *Agricultura de precisão: resultados de um novo olhar*. [S.l.]: Embrapa, 2014. Chapter 13, p. 180–190.
- KACPRZYK, J.; ZADROZNY, S.; TRÉ, G. D. Fuzziness in database management systems: Half a century of developments and future prospects. *Fuzzy Sets and Systems*, v. 281, p. 300–307, 2015. ISSN 0165-0114. Special Issue Celebrating the 50th Anniversary of Fuzzy Sets. Available at: <<http://www.sciencedirect.com/science/article/pii/S0165011415003000>>.
- KALASHNIKOV, D. V. et al. Index for fast retrieval of uncertain spatial point data. In: *14th ACM International Symposium on Advances in Geographic Information Systems (GIS'2006)*. Arlington, VA, USA: ACM, 2006. (Proceedings...), p. 195–202. ISBN 1-59593-529-0. Available at: <<http://doi.acm.org/10.1145/1183471.1183504>>.

- KANJILAL, V.; LIU, H.; SCHNEIDER, M. Plateau regions: An implementation concept for fuzzy regions in spatial databases and GIS. In: HULLERMEIER, E.; KRUSE, R.; HOFFMANN, F. (Ed.). *13th International Conference on Information Processing and Management of Uncertainty (IPMU'10)*. Dortmund, Germany: Springer Berlin Heidelberg, 2010, (Lecture Notes in Computer Science: Computational Intelligence for Knowledge-Based Systems Design, v. 6178). p. 624–633. ISBN 978-3-642-14048-8. Available at: <http://dx.doi.org/10.1007/978-3-642-14049-5_64>.
- KIMBALL, R.; ROSS, M. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. 2. ed. New York, NY, USA: Wiley, 2002. ISBN 978-0471200246.
- KLENKE, A. *Probability Theory: A Comprehensive Course*. 2. ed. London, UK: Springer London, 2014. (Universitext). ISBN 978-1-4471-5361-0.
- LANE, P.; POTINENI, P. Data warehousing optimizations and techniques. In: *Oracle Database Data Warehousing Guide 12c Release 1 (12.1)*. [s.n.], 2014. Chapter 4. Available at: <<https://docs.oracle.com/database/121/DWHSG/schemas.htm#DWHSG019>>.
- LAURENT, A. Querying fuzzy multidimensional databases: Unary operators and their properties. *International Journal of Uncertainty, Fuzziness & Knowledge-Based Systems*, v. 11, p. 31 – 45, 2003. ISSN 02184885.
- LEONARD, R.; KNISEL, W.; STILL, D. GLEAMS: Groundwater loading effects of agricultural management systems. *Transactions of the American Society of Agricultural Engineers*, v. 5, p. 1403–1418, 1987.
- LEUNG, Y. On the imprecision of boundaries. *Geographical Analysis*, Ohio State University Press, v. 19, n. 2, 1987.
- LI, R. et al. Uncertain spatial data handling: Modeling, indexing and query. *Computers & Geosciences*, v. 33, n. 1, p. 42 – 61, 2007. ISSN 0098-3004. Available at: <<http://www.sciencedirect.com/science/article/pii/S0098300406000926>>.
- LIU, K.; SHI, W. Quantitative fuzzy topological relations of spatial objects by induced fuzzy topology. *International Journal of Applied Earth Observation and Geoinformation*, v. 11, n. 1, p. 38 – 45, 2009. ISSN 0303-2434. Available at: <<http://www.sciencedirect.com/science/article/pii/S0303243408000470>>.
- MALINOWSKI, E.; ZIMÁNYI, E. *Advanced Data Warehouse Design: From Conventional to Spatial and Temporal Applications*. Berlin, Heidelberg: Springer-Verlag, 2009. ISBN 3642093833, 9783642093838.
- MATEUS, R. C. et al. How does the spatial data redundancy affect query performance in geographic data warehouses? *Journal of Information and Data Management*, v. 1, n. 3, p. 519–534, 2010. Available at: <<http://seer.lcc.ufmg.br/index.php/jidm/article/view/61>>.
- MATEUS, R. C. et al. Spatial data warehouses and spatial OLAP come towards the cloud: design and performance. *Distributed and Parallel Databases*, Springer US, p. 1–37, 2015. ISSN 0926-8782. Available at: <<http://dx.doi.org/10.1007/s10619-015-7176-z>>.

- MOHAN, P. et al. Should SDBMS support a join index?: A case study from crimestat. In: *16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'2008)*. Irvine, CA, USA: ACM, 2008. (Proceedings...), p. 37:1–37:10. ISBN 978-1-60558-323-5. Available at: <<http://doi.acm.org/10.1145/1463434.1463481>>.
- NEITSCH, S.; ARNOLD, J.; WILLIAMS, J. *Soil and Water Assessment Tool: Theoretical Documentation*. Temple, TX, USA, August 2011. Available at: <<http://swat.tamu.edu/media/99192/swat2009-theory.pdf>>.
- OBE, R. O.; HSU, L. S. *PostGIS in Action*. 2. ed. Shelter Island, NY, USA: Manning Publications Co., 2015. ISBN 9781617291395.
- O'NEIL, P. et al. The star schema benchmark and augmented fact table indexing. In: NAMBIAR, R.; POESS, M. (Ed.). *First TPC Technology Conference (TPCTC 2009)*. Lyon, France: Springer Berlin Heidelberg, 2009, (Lecture Notes in Computer Science: Performance Evaluation and Benchmarking, v. 5895). p. 237–252. ISBN 978-3-642-10423-7. Available at: <http://dx.doi.org/10.1007/978-3-642-10424-4_17>.
- O'NEIL, P. E. Model 204 architecture and performance. In: GAWLICK, D.; HAYNIE, M.; REUTER, A. (Ed.). *2nd International Workshop on High Performance Transaction Systems*. Pacific Grove, CA, USA: Springer Berlin Heidelberg, 1989, (Lecture Notes in Computer Science: High Performance Transaction Systems, v. 359). p. 39–59. ISBN 978-3-540-51085-7.
- O'NEIL, P. E.; GRAEFE, G. Multi-table joins through bitmapped join indices. *SIGMOD Record*, ACM, New York, NY, USA, v. 24, n. 3, p. 8–11, September 1995. ISSN 0163-5808. Available at: <<http://doi.acm.org/10.1145/211990.212001>>.
- OORT, P. v. *Spatial data quality: from description to application*. Thesis (Ph.D.) — Wageningen Universiteit, January 2006. Available at: <<http://library.wur.nl/WebQuery/clc/1788022>>.
- ORACLE Spatial and Graph: Advanced Data Management. [S.l.], September 2014. Available at: <<http://www.oracle.com/technetwork/database/options/spatialandgraph/spatial-and-graph-wp-12c-1896143.pdf?ssSourceSiteId=ocomen>>.
- PAPADIAS, D. et al. Efficient OLAP operations in spatial data warehouses. In: JENSEN, C. S. et al. (Ed.). *7th International Symposium on Advances in Spatial and Temporal Databases (SSTD'2001)*. Redondo Beach, CA, USA: Springer Berlin Heidelberg, 2001, (Lecture Notes in Computer Science: Advances in Spatial and Temporal Databases, v. 2121). p. 443–459. ISBN 978-3-540-42301-0. Available at: <http://dx.doi.org/10.1007/3-540-47724-1_23>.
- PARENT, C.; SPACCAPIETRA, S.; ZIMÁNYI, E. *Conceptual modeling for traditional and spatio-temporal applications - the MADS approach*. Secaucus, NJ, USA: Springer, 2006. ISBN 978-3-540-30153-0.
- PAULY, A.; SCHNEIDER, M. VASA: An algebra for vague spatial data in databases. *Information Systems*, v. 35, n. 1, p. 111–138, 2010. ISSN 0306-4379. Available at: <<http://www.sciencedirect.com/science/article/pii/S0306437909000519>>.
- PEBESMA, E. J.; KARSSENBERG, D.; JONG, K. d. Dynamic visualisation of spatial and spatio-temporal probability distribution functions. In: CAETANO, M.; M., P. (Ed.). *7th International Symposium on Spatial Accuracy Assessment in Natural Resources and Environmental Sciences (Accuracy 2006)*. Lisbon, Portugal: Instituto Geográfico Português,

2006. (Proceedings...), p. 825–831. ISBN 972-8867271. Available at: <<http://www.spatial-accuracy.org/Reis2006accuracy>>.
- PEDERSEN, T. B. Multidimensional modeling. In: LIU, L.; ÖZSU, M. T. (Ed.). *Encyclopedia of Database Systems*. New York, NY, USA: Springer US, 2009. p. 1777–1784. ISBN 978-0-387-39940-9.
- PEREZ, D.; SOMODEVILLA, M.; PINEDA, I. H. Fuzzy spatial data warehouse: A multidimensional model. In: *Eighth Mexican International Conference on Current Trends in Computer Science (ENC'2007)*. Michoacan, Mexico: IEEE, 2007. (Proceedings...), p. 3–9. ISBN 978-0-7695-2899-1.
- PEREZ, D.; SOMODEVILLA, M.; PINEDA, I. H. Fuzzy spatial data warehouse: A multidimensional model. In: DEVLIN, G. (Ed.). *Advances in Decision Support Systems*. InTech, 2010, (Eighth Mexican International Conference on Current Trends in Computer Science). Chapter 4. ISBN 978-953-307-069-8. Available at: <<http://www.intechopen.com/books/decision-support-systems-advances-in/fuzzy-spatial-data-warehouse-a-multidimensional-model>>.
- PETRY, F. E.; LADNER, R.; SOMODEVILLA, M. Indexing implementation for vague spatial regions with R-trees and grid files. In: MORRIS, A.; KOKHAN, S. (Ed.). *Geographic Uncertainty in Environmental Security*. Dordrecht, The Netherlands: Springer Netherlands, 2007. Chapter 11. ISBN 978-1-4020-6438-8.
- PINET, F.; SCHNEIDER, M. Precise design of environmental data warehouses. *Operational Research*, Springer-Verlag, v. 10, n. 3, p. 349–369, 2010. ISSN 1109-2858. Available at: <<http://dx.doi.org/10.1007/s12351-009-0069-z>>.
- POSTGRESQL 9.4.1 Documentation. [S.l.], February 2015. Available at: <<http://www.postgresql.org/docs/9.4/interactive/arrays.html>>.
- POURABBAS, E.; RAFANELLI, M. Characterization of hierarchies and some operators in olap environment. In: *2nd ACM International Workshop on Data warehousing and OLAP (DOLAP'1999)*. Kansas City, MO, USA: ACM, 1999. (Proceedings...), p. 54–59. ISBN 1-58113-220-4. Available at: <<http://doi.acm.org/10.1145/319757.319790>>.
- PUSTIKA, A. et al. Interactions between plant nutrition and symptom expression in mandarin trees infected with the disease huanglongbing. *Australasian Plant Disease Notes*, Springer Netherlands, v. 3, n. 1, p. 112–115, 2008. ISSN 1833-928X. Available at: <<http://dx.doi.org/10.1007/BF03211261>>.
- RANDELL, D.; CUI, Z.; COHN, A. A spatial logic based on regions and connection. In: *3rd International Conference on Principles of Knowledge Representation and Reasoning*. San Mateo, CA, USA: Morgan Kaufmann, 1992. p. 165–176.
- RAO, F. et al. Spatial hierarchy and OLAP-favored search in spatial data warehouse. In: *6th ACM International Workshop on Data Warehousing and OLAP (DOLAP'2003)*. New Orleans, LA, USA: ACM, 2003. (Proceedings...), p. 48–55. ISBN 1-58113-727-3. Available at: <<http://doi.acm.org/10.1145/956060.956070>>.
- RECIO, J. et al. Automated extraction of tree and plot-based parameters in citrus orchards from aerial images. *Computers and Electronics in Agriculture*, v. 90, n. 0, p. 24 – 34, 2013. ISSN 0168-1699. Available at: <<http://www.sciencedirect.com/science/article/pii/S0168169912002566>>.

- SAPIR, L.; SHMILOVICI, A.; ROKACH, L. A methodology for the design of a fuzzy data warehouse. In: *4th International IEEE Conference on Intelligent Systems (IS'2008)*. [S.l.: s.n.], 2008. (Proceedings..., v. 1), p. 2–14–2–21. ISBN 978-1-4244-1740-7.
- SCHNEIDER, M. Fuzzy spatial data types for spatial uncertainty management in databases. In: GALINDO, J. (Ed.). *Handbook of Research on Fuzzy Information Processing in Databases*. [S.l.]: IGI Global, 2008. II, Chapter 19, p. 490–515. ISBN 9781599048536.
- SCHNEIDER, M. Spatial plateau algebra for implementing fuzzy spatial objects in databases and GIS: Spatial plateau data types and operations. *Applied Soft Computing*, v. 16, p. 148 – 170, 2014. ISSN 1568-4946. Available at: <<http://www.sciencedirect.com/science/article/pii/S1568494613004249>>.
- SHI, W.; LIU, K. A fuzzy topology for computing the interior, boundary, and exterior of spatial objects quantitatively in GIS. *Computers & Geosciences*, v. 33, n. 7, p. 898 – 915, 2007. Available at: <<http://www.sciencedirect.com/science/article/pii/S0098300407000210>>.
- SHI, Y. et al. Fuzzy control of the spraying medicine control system. In: LI, D. (Ed.). *Computer And Computing Technologies In Agriculture, Volume II*. Springer US, 2008, (The International Federation for Information Processing, v. 259). p. 1087–1094. ISBN 978-0-387-77252-3. Available at: <http://dx.doi.org/10.1007/978-0-387-77253-0_42>.
- SILVA, J. et al. A set of aggregation functions for spatial measures. In: *11th ACM International Workshop on Data Warehousing and OLAP (DOLAP'2008)*. Napa Valley, CA, USA: ACM, 2008. (Proceedings...), p. 25–32. ISBN 978-1-60558-250-4. Available at: <<http://doi.acm.org/10.1145/1458432.1458438>>.
- SILVA, J. da et al. Modelling and querying geographical data warehouses. *Information Systems*, v. 35, n. 5, p. 592 – 614, 2010. ISSN 0306-4379. Twenty-second Brazilian Symposium on Databases (SBBD 2007). Available at: <<http://www.sciencedirect.com/science/article/pii/S0306437909001033>>.
- SIQUEIRA, T. L. L. *SB-index: Um Índice Espacial baseado em Bitmap para Data Warehouse Geográfico*. Dissertation (M.Sc.) — Universidade Federal de São Carlos (Brasil), Agosto 2009. Available at: <http://www.btdt.ufscar.br/htdocs/tedeSimplificado/tde_busca/arquivo.php?codArquivo=2861>.
- SIQUEIRA, T. L. L. et al. The impact of spatial data redundancy on SOLAP query performance. *Journal of the Brazilian Computer Society*, v. 15, n. 2, p. 19–34, 2009. Available at: <<http://dx.doi.org/10.1590/S0104-65002009000200003>>.
- SIQUEIRA, T. L. L. et al. Towards vague geographic data warehouses. In: XIAO, N. et al. (Ed.). *7th International Conference on Geographic Information Science (GIScience'2012)*. Columbus, OH, USA: Springer Berlin Heidelberg, 2012, (Lecture Notes in Computer Science: Geographic Information Science, v. 7478). p. 173–186. ISBN 978-3-642-33023-0. Available at: <http://dx.doi.org/10.1007/978-3-642-33024-7_13>.
- SIQUEIRA, T. L. L. et al. The SB-index and the HSB-index: Efficient indices for spatial data warehouses. *Geoinformatica*, Kluwer Academic Publishers, Hingham, MA, USA, v. 16, n. 1, p. 165–205, jan 2012. ISSN 1384-6175. Available at: <<http://dx.doi.org/10.1007/s10707-011-0128-5>>.

- SIQUEIRA, T. L. L. et al. Modeling vague spatial data warehouses using the VSCube conceptual model. *GeoInformatica*, Springer US, v. 18, n. 2, p. 313–356, 2014. ISSN 1384-6175. Available at: <<http://dx.doi.org/10.1007/s10707-013-0186-y>>.
- SIQUEIRA, T. L. L. et al. Investigating the effects of spatial data redundancy in query performance over geographical data warehouses. In: CARVALHO, M. T. M. et al. (Ed.). *X Brazilian Symposium on Geoinformatics (GeoInfo'2008)*. Rio de Janeiro, RJ, Brazil: [s.n.], 2008. p. 1–12. Available at: <http://www.geoinfo.info/proceedings_geoinfo2008.split/proceedings_geoinfo2008.11_22.pdf>.
- SIQUEIRA, T. L. L. et al. A spatial bitmap-based index for geographical data warehouses. In: SHIN, S. Y.; OSSOWSKI, S. (Ed.). *2009 ACM Symposium on Applied Computing (SAC'2009)*. Honolulu, HI, USA: ACM, 2009. (Proceedings...), p. 1336–1342. ISBN 978-1-60558-166-8.
- SIQUEIRA, T. L. L. et al. Benchmarking spatial data warehouses. In: PEDERSEN, T. B.; MOHANIA, M. K.; TJOA, A. M. (Ed.). *12th International Conference on Data Warehousing and Knowledge Discovery*. Bilbao, Spain: Springer Berlin Heidelberg, 2010, (Lecture Notes in Computer Science: Data Warehousing and Knowledge Discovery, v. 6263). p. 40–51. ISBN 978-3-642-15104-0.
- SIQUEIRA, T. L. L.; CIFERRI, R. R.; ZIMÁNYI, E. Extending the multidim conceptual model to enable the design of vague spatial data warehouses. In: *Dutch-Belgian Database Day*. Antwerp, Belgium: [s.n.], 2014. Available at: <<http://adrem.ua.ac.be/dbdbd2014/>>.
- SIQUEIRA, T. L. L.; CIFERRI, R. R.; ZIMÁNYI, E. *Projeto Lógico de Data Warehouse Espacial Vago*. Brasília, DF, Brasil, Abril 2015. (Relatório técnico-científico final - Doutorado-sanduiche (SWE): 229675/2013-1. Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq)).
- SIQUEIRA, T. L. L. et al. Querying vague spatial information in geographic data warehouses. In: GEERTMAN, S.; REINHARDT, W.; TOPPEN, F. (Ed.). *14th AGILE International Conference on Geographic Information Science*. Utrecht, The Netherlands: Springer Berlin Heidelberg, 2011, (Lecture Notes in Geoinformation and Cartography: Advancing Geoinformation Science for a Changing World, v. 1). p. 379–397. ISBN 978-3-642-19788-8. Available at: <http://dx.doi.org/10.1007/978-3-642-19789-5_19>.
- SIQUEIRA, T. L. L. et al. Indexing vague regions in spatial data warehouses. In: SANTANCHÈ, A.; ANDRADE, P. R. (Ed.). *XIV Brazilian Symposium on Geoinformatics (GeoInfo'2013)*. Campos do Jordão, SP, Brazil: MC-T/INPE, 2013. (Proceedings...), p. 158–169. ISSN 2179-4820. Available at: <http://www.geoinfo.info/proceedings_geoinfo2013.split/paper17.pdf>.
- SIQUEIRA, T. L. L. et al. Indexing and querying vague spatial data warehouses. *Journal of Information and Data Management*, v. 5, n. 2, p. 161–170, 2014. Available at: <<https://seer.lcc.ufmg.br/index.php/jidm/article/view/323>>.
- SIQUEIRA, T. L. L.; ZIMÁNYI, E.; CIFERRI, R. R. *Projeto Lógico de Data Warehouse Espacial Vago*. São Paulo, SP, Brasil, Julho 2015. (Relatório científico final - Doutorado (DR): 14/14103-9. Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP)).

- SOMODEVILLA, M.; PETRY, F. Indexing mechanisms to query FMBRs. In: *2004 Annual Meeting of the North American Fuzzy Information Processing Society*. [S.l.]: IEEE, 2004. v. 1, p. 198–202. ISBN 0-7803-8376-1.
- SOMODEVILLA, M. J.; PETRY, F. Approximation of topological relations on fuzzy regions: an approach using minimal bounding rectangles. In: *22nd International Conference of the North American Fuzzy Information Processing Society (NAFIPS'2003)*. Chicago, IL, USA: IEEE, 2003. (Proceedings...), p. 371–376. ISBN 0-7803-7918-7.
- SONG, I.; CHEN, P. P. Entity relationship model. In: LIU, L.; ÖZSU, M. T. (Ed.). *Encyclopedia of Database Systems*. New York, NY, USA: Springer US, 2009. p. 1003–1009. ISBN 978-0-387-39940-9.
- STEFANOVIC, N.; HAN, J.; KOPERSKI, K. Object-based selective materialization for efficient implementation of spatial data cubes. *Transactions on Knowledge and Data Engineering*, IEEE, v. 12, n. 6, p. 938–958, November 2000. ISSN 1041-4347.
- STOCKINGER, K.; WU, K. Bitmap indices for data warehouses. In: WREMBEL, R.; KONCILIA, C. (Ed.). *Data Warehouses and OLAP: Concepts, Architectures and Solutions: Concepts, Architectures and Solutions*. Hershey, PA, USA: IGI Global, 2006. Chapter 7, p. 157–178. ISBN 1599043645.
- SUCIU, D. Probabilistic databases. In: LIU, L.; ÖZSU, M. T. (Ed.). *Encyclopedia of Database Systems*. New York, NY, USA: Springer US, 2009. p. 2150–2155. ISBN 978-0-387-39940-9.
- TANG, X.; KAINZ, W.; WANG, H. Topological relations between fuzzy regions in a fuzzy topological space. *International Journal of Applied Earth Observation and Geoinformation*, v. 12, Supplement 2, p. S151 – S165, 2010. ISSN 0303-2434. Supplement Issue on Spatial Analysis-Modeling, Methodology and applications. Available at: <<http://www.sciencedirect.com/science/article/pii/S0303243410000073>>.
- TAO, Y. et al. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In: *31st International Conference on Very Large Data Bases (VLDB'2005)*. VLDB Endowment, 2005. (Proceedings...), p. 922–933. ISBN 1-59593-154-6. Available at: <<http://dl.acm.org/citation.cfm?id=1083592.1083699>>.
- TAO, Y.; XIAO, X.; CHENG, R. Range search on multidimensional uncertain data. *ACM Transactions on Database Systems*, v. 32, n. 3, p. 15, 2007. Available at: <<http://doi.acm.org/10.1145/1272743.1272745>>.
- TEOREY, T. J.; YANG, D.; FRY, J. P. A logical design methodology for relational databases using the extended Entity-Relationship model. *ACM Computing Surveys*, ACM, New York, NY, USA, v. 18, n. 2, p. 197–222, jun 1986. ISSN 0360-0300.
- THALHEIM, B. Abstraction. In: LIU, L.; ÖZSU, M. T. (Ed.). *Encyclopedia of Database Systems*. New York, NY, USA: Springer US, 2009. p. 6–7. ISBN 978-0-387-39940-9.
- TIMKO, I.; DYRESON, C. E.; PEDERSEN, T. B. A probabilistic data model and algebra for location-based data warehouses and their implementation. *GeoInformatica*, Springer US, v. 18, n. 2, p. 357–403, 2014. ISSN 1384-6175. Available at: <<http://dx.doi.org/10.1007/s10707-013-0180-4>>.

- VAISMAN, A.; ZIMÁNYI, E. A multidimensional model representing continuous fields in spatial data warehouses. In: *17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'2009)*. Seattle, WA, USA: ACM, 2009. (Proceedings...), p. 168–177. ISBN 978-1-60558-649-6. Available at: <<http://doi.acm.org/10.1145/1653771.1653797>>.
- VAISMAN, A. A. *Bitmap Indexing: From Model 204 To Data Warehouses*. Buenos Aires, Argentina, 1998. (Report number: 98-016 - Universidad de Buenos Aires). Available at: <<ftp://zorzal.dc.uba.ar/pub/tr/1998/98-016.ps>>.
- VAISMAN, A. A.; ZIMÁNYI, E. *Data Warehouse Systems - Design and Implementation*. 1. ed. Heidelberg, Germany: Springer-Verlag Berlin Heidelberg, 2014. (Data-Centric Systems and Applications). ISSN 2197-9723. ISBN 978-3-642-54655-6. Available at: <<http://dx.doi.org/10.1007/978-3-642-54655-6>>.
- VAISMAN, A. A.; ZIMÁNYI, E. Spatial data warehouses. In: *Data Warehouse Systems - Design and Implementation*. Heidelberg, Germany: Springer-Verlag Berlin Heidelberg, 2014, (Data-Centric Systems and Applications). Chapter 11, p. 427–474. ISBN 978-3-642-54655-6.
- VERSTRAETE, J.; HALLEZ, A.; TRÉ, G. D. Fuzzy regions: Theory and applications. In: MORRIS, A.; KOHKHAN, S. (Ed.). *Geographic Uncertainty in Environmental Security*. Kyiv, Ukraine: Springer Netherlands, 2006. Chapter 1, p. 1–18. ISBN 978-1-4020-6438-8.
- VERSTRAETE, J. et al. Field based methods for the modeling of fuzzy spatial data. In: PETRY, F. E.; ROBINSON, V. B.; COBB, M. A. (Ed.). *Fuzzy Modeling with Spatial Information for Geographic Problems*. New York, NY, USA: Springer Berlin Heidelberg, 2005. p. 41–69. ISBN 978-3-540-23713-6.
- VERSTRAETE, J.; TRÉ, G. D.; HALLEZ, A. Bitmap based structures for the modeling of fuzzy entities. *Control and Cybernetics*, Systems Research Institute Polish Academy of Sciences, v. 35, n. 1, p. 147–164, 2006.
- VERSTRAETE, J. et al. Using TIN-based structures for the modelling of fuzzy GIS objects in a database. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, v. 15, n. supp01, p. 1–20, 2007. Available at: <<http://www.worldscientific.com/doi/abs/10.1142/S0218488507004431>>.
- WANG, J. et al. A fast region merging algorithm for watershed segmentation. In: *7th International Conference on Signal Processing (ICSP '2004)*. Beijing, China: IEEE, 2004. (Proceedings..., v. 1), p. 781–784.
- WIDOM, J. Trio: A system for integrated management of data, accuracy, and lineage. In: *Second Biennial Conference on Innovative Data Systems Research*. Asilomar, CA, USA: [s.n.], 2005. (CIDR), p. 262–276. Available at: <<http://www.cidrdb.org/cidr2005/papers/P22.pdf>>.
- WOLFSON, O. et al. Updating and querying databases that track mobile units. *Distributed and Parallel Databases*, Kluwer Academic Publishers, Hingham, MA, USA, v. 7, n. 3, p. 257–387, July 1999. ISSN 0926-8782. Available at: <<http://dx.doi.org/10.1023/A:1008782710752>>.
- WORBOYS, M. Imprecision in finite resolution spatial data. *Geoinformatica*, Kluwer Academic Publishers, Hingham, MA, USA, v. 2, n. 3, p. 257–279, October 1998. ISSN 1384-6175. Available at: <<http://dx.doi.org/10.1023/A:1009769705164>>.

- WU, K. et al. FastBit: interactively searching massive data. *Journal of Physics: Conference Series*, IOP Publishing Ltd, v. 180:12053, n. 1, 2009. ISSN 1742-6596. Available at: <<http://dx.doi.org/10.1088/1742-6596/180/1/012053>>.
- WU, K.; OTOO, E. J.; SHOSHANI, A. Optimizing bitmap indices with efficient compression. *Transactions on Database Systems*, ACM, New York, NY, USA, v. 31, n. 1, p. 1–38, March 2006. ISSN 0362-5915.
- YUEN, S. M. et al. Superseding nearest neighbor search on uncertain spatial databases. *Transactions on Knowledge and Data Engineering*, IEEE, v. 22, n. 7, p. 1041–1055, July 2010. ISSN 1041-4347.
- ZADEH, L. Fuzzy sets. *Information and Control*, v. 8, n. 3, p. 338–353, 1965. ISSN 00199958. Available at: <<http://www.sciencedirect.com/science/article/pii/S001999586590241X>>.
- ZHAN, F. B. Approximate analysis of binary topological relations between geographic regions with indeterminate boundaries. *Soft Computing*, Springer-Verlag, v. 2, n. 2, p. 28–34, 1998. ISSN 1432-7643. Available at: <<http://dx.doi.org/10.1007/s005000050032>>.
- ZIMÁNYI, E.; PIROTTE, A. Imperfect information in relational databases. In: MOTRO, A.; SMETS, P. (Ed.). *Uncertainty Management in Information Systems*. Springer US, 1997. Chapter 3, p. 35–87. ISBN 978-1-4613-7865-5. Available at: <http://dx.doi.org/10.1007/978-1-4615-6245-0_3>.
- ZINN, D.; BOSCH, J.; GERTZ, M. Modeling and querying vague spatial objects using shapelets. In: *33rd International Conference on Very Large Data Bases (VLDB'2007)*. Vienna, Austria: VLDB Endowment, 2007. (Proceedings...), p. 567–578. ISBN 978-1-59593-649-3. Available at: <<http://dl.acm.org/citation.cfm?id=1325851.1325917>>.

GLOSSARY

CERA – *coastal erosion risk assessment*

CRQ – *containment range query*

DBMS – *database management system*

E-R – *Entity Relationship*

ETL – *Extract, Transform and Load*

Embrapa – *The Brazilian Agricultural Research Corporation*

IRQ – *intersection range query*

MBR – *minimum bounding rectangle*

MER – *maximum enclosed rectangle*

MIP – *maximum area inscribed polygon*

OGC – *Open Geospatial Consortium*

QMM – *Qualitative Min-Max Model*

RBB – *Region with Broad Boundaries*

SB-index – *Spatial Bitmap Index*

SDW – *spatial data warehouse*

SOLAP – *spatial online analytical processing*

SQL – *Structured Query Language*

SRID – *spatial reference system identifier*

SRS – *spatial reference system*

TIN – *triangulated irregular network*

UDF – *user-defined function*

UDT – *user-defined type*

VASA – *Vague Spatial Algebra*

VSB-index – *Vague Spatial Bitmap Index*

VSCube – *Vague Spatial Cube*

VSMultiDim – *Vague Spatial MultiDim*

VSRQ – *vague spatial range query*

aR-tree – *aggregate R-tree*

Appendix A

USER-DEFINED FUNCTIONS

This appendix describes the following UDFs that are called by UDFs that implement constraints:

- *VS_Constraints_FuzzyMval* returns the length of the array provided as argument if all the membership values in the array are in $]0, 1]$, or returns -1 otherwise;
- *VS_Constraints_ExactMval* returns the length of the array provided as argument if all the membership values in the array are in $\{-1, 1\}$, or returns -1 otherwise;
- *CheckOrderedArray* returns true if the array provided as argument is in ascending order, or false otherwise;
- *InteriorIntersection* returns true if at least two geometries intersect in the geometry collection provided as argument; and
- *VS_MinOffsetCertitudeElement* returns the minimum offset of the array provided as argument where the entry is equal to 1.0, otherwise returns 0.

The UDF *VS_Constraints_FuzzyMval* is indicated for vague spatial objects whose membership value is in $]0, 1]$. It is described in Listing A.1. First, it obtains the highest and the lowest membership values in the array, and counts how many membership values are not null. The PostgreSQL's function *unnest*¹ is used and transforms an array into a set of rows. Second, it uses the obtained highest and lowest membership values to check if they are in $]0, 1]$. If both are in $]0, 1]$, the remaining membership values are also in $]0, 1]$ and the execution proceeds. Otherwise, a notice is yielded and the function returns -1. Third, it compares the length of the array

¹<http://www.postgresql.org/docs/9.2/static/functions-array.html>

to the quantity of non-null membership values previously counted. Their equality terminates the function with success returning the length of the array, while their inequality raises a notice and terminates the function returning -1.

Listing A.1: Checking membership values in between 0 and 1.

```
CREATE OR REPLACE FUNCTION VS_Constraints_FuzzyMval(ElementsMval float array)
RETURNS BIGINT AS $$
DECLARE
ceiling float;
floor float;
counter bigint;
length bigint;
BEGIN
--1. gathers the minimum and maximum membership values provided
-- and counts non-null membership values
SELECT max(unnest), min(unnest), count(*) INTO ceiling, floor, counter
FROM unnest(ElementsMval) WHERE unnest IS NOT NULL;

--2. verifies whether membership values are in ]0,1]
IF (ceiling > 1.0) THEN
RAISE NOTICE 'Invalid membership value: % is greater than 1.0.', ceiling;
RETURN -1;
END IF;
IF (floor <= 0.0) THEN
RAISE NOTICE 'Invalid membership value: % is lower than or equal to 0.0.', floor;
RETURN -1;
END IF;

--3. verifies whether null membership values exist
SELECT array_length(ElementsMval,1) INTO length;
IF (counter <> length) THEN
RAISE NOTICE 'A membership value cannot be null';
RETURN -1;
END IF;
RETURN length;
END;
$$ LANGUAGE 'plpgsql';
```

The UDF *VS_Constraints_ExactMval* is indicated for vague spatial objects whose membership value is 1 to denote true (certitude element) or -1 to denote maybe (dubiety element). It is described in Listing A.2. First, it counts how many membership values of the array assume the value -1 or 1, using PostgreSQL's function *unnest*. Second, it obtains the length of the array and compares with the previous count. Their inequality means that one or more membership values are not adequate and, thus, the function terminates returning -1. Conversely, their equality means that all membership values provided are adequate and the function terminates successfully returning the length of the array.

Listing A.2: A routine to check if membership values are -1 or 1.

```
CREATE OR REPLACE FUNCTION VS_Constraints_ExactMval(ElementsMval float array)
RETURNS BIGINT AS $$
DECLARE
ceiling float;
floor float;
counter bigint;
length bigint;
BEGIN
```

```

--1. counts rows with valid membership values, i.e. -1.0 or 1.0
SELECT count(*) INTO counter
FROM unnest(ElementsMval)
WHERE unnest IS NOT NULL
AND (unnest=-1.0 OR unnest=1.0);

--2. verifies whether membership values are valid
SELECT array_length(ElementsMval,1) INTO length;
IF (counter <> length) THEN
RAISE NOTICE 'Membership values must be -1.0 (maybe) or 1.0 (true) and cannot be null.';
RETURN -1;
END IF;

RETURN length;
END;
$$ LANGUAGE 'plpgsql';

```

The UDF *CheckOrderedArray* is described in Listing A.3. First, a query counts how many membership values are not sorted. Basically, two queries unnest the array in different relations with row numbers, such that one has sorted values and the other has the original sequence. Then, these relations are joined and the subtraction of the membership values must be equal to zero. The occurrences of values different from zero are counted, and denote how many membership values are not sorted. Second, if there are unsorted membership values, the function returns false. Otherwise, it returns true.

Listing A.3: A routine to check if an array of membership value is sorted in ascending order.

```

CREATE OR REPLACE FUNCTION CheckOrderedArray (a FLOAT ARRAY) RETURNS BOOLEAN AS
$$
DECLARE
nr_unordered BIGINT;
BEGIN
--counts how many membership values are not sorted
SELECT count(*) INTO nr_unordered
FROM (SELECT ordered-original AS subtraction
      FROM (SELECT ordered, row_number() OVER (ORDER BY ordered) AS i
            FROM (SELECT unnest AS ordered FROM unnest(a)) AS t1) AS t10,
          (SELECT original, row_number() OVER (ORDER BY k) AS j
            FROM (SELECT unnest AS original, 0 AS k FROM unnest(a)) AS t2 ) AS t20
      WHERE i=j
     ) AS tab
WHERE subtraction <>0.0;

--if there are unsorted membership values, returns false
IF nr_unordered <> 0 THEN
RETURN FALSE;
END IF;

RETURN TRUE;
END;
$$ LANGUAGE 'plpgsql';

```

The UDF *InteriorIntersection* is described in Listing A.4. Each pair of geometries in the geometry collection is tested regarding intersection throughout a pair of iterative loops. If an intersection is detected, the function terminates returning true. If the loops finish without detecting an intersection, the function returns false.

Listing A.4: A routine to check if the interiors of two geometries intersect.

```

CREATE OR REPLACE FUNCTION InteriorIntersection(multigeom geometry) RETURNS BOOLEAN AS
$$
DECLARE
  nr_geoms bigint;
  geom geometry;
BEGIN
  nr_geoms := ST_NumGeometries(multigeom);
  IF (nr_geoms = 1) THEN
    RETURN false;
  END IF;
  FOR i IN 1 .. nr_geoms-1 LOOP
    geom := ST_GeometryN(multigeom, i);
    FOR j IN 2 .. nr_geoms LOOP
      IF (ST_Relate(geom, ST_GeometryN(multigeom, j), 'T*****')) THEN
        raise notice 'i = % j = %',i,j;
        RETURN true;
      END IF;
    END LOOP;
  END LOOP;
  RETURN false;
END;
$$ LANGUAGE 'plpgsql';

```

Listing A.5 describes the function *VS_MinOffsetCertitudeElement* that receives an array of membership values as argument and returns the minimum offset of a certitude element. In other words, it identifies the lowest offset where an entry has the value 1.0. To achieve this goal, it is necessary to *unnest* the array without sorting, introduce row numbers and apply a WHERE clause that is more scalable than a sequential scan.

Listing A.5: A routine to obtain the minimum offset of a certitude element in an array of membership values.

```

CREATE OR REPLACE FUNCTION VS_MinOffsetCertitudeElement(mv float array)
RETURNS INT AS $$
DECLARE
  o int;
BEGIN
  SELECT min(row_number) INTO o FROM (
    SELECT unnest, row_number() OVER (ORDER BY aux) FROM(
      SELECT 0 AS aux, unnest FROM unnest(mv)
    ) AS unnested_mv
  ) AS o WHERE unnest = 1.0;
  RETURN o;
END;
$$ LANGUAGE 'plpgsql';

```


Appendix B

PROCEDURES OF THE VSB-INDEX

This Appendix complements the content of the Section 6.3.4 and describes the procedures f3 and f4.

The procedure f3, detailed in Algorithm 8, is called to resolve a $CRQ_{certitude}$ when there are not a conservative and neither a progressive approximation built for the certitude, i.e. in configurations O_{\supseteq} and $O_{\supseteq}O_{\subseteq}$. It checks whether the conservative approximation of the dubiety is within the spatial query window and, if so, the entry is considered an answer (lines 6-7). This decision is valid because the conservative approximation of the dubiety contains the certitude of the corresponding vague region. On the other hand, if the conservative approximation of the dubiety merely intersects the spatial query window, then the entry is considered a candidate (lines 9-10). This decision is also valid, since the spatial query window might intersect the conservative approximation of the dubiety and such verification is postponed until the refinement step.

The procedure f4, detailed in Algorithm 9, is called to resolve a $CRQ_{certitude}$ when there is not a conservative approximation on the certitude, but there is a progressive approximation on the certitude, i.e. in configurations $O_{\supseteq}C_{\subseteq}$ and $O_{\supseteq}O_{\subseteq}C_{\subseteq}$. It produces answers similarly to routine f3 (lines 6-7). Candidates are only collected if the spatial query window intersects the conservative approximation of the dubiety and contains the progressive approximation of the certitude (lines 9-10). This criterion is essential because the conservative approximation of the dubiety and the spatial query window may intersect, but the progressive approximation of the certitude might be within the spatial query window while the certitude itself might not be within the spatial query window.

Algorithm 8: $f3(R, \text{window}, \text{conservative}, \text{setCandidates}, \text{setAnswers}, \text{idx}, L)$

Input: R is a topological relationship

window is a spatial query window

conservative is a conservative approximation

setCandidates is a set of candidates

setAnswers is a set of answers

idx is the sequential file of the VSB-index

L the number of VSB-index' entries that a disk page can hold.

Data: page , array

Result: The set of answers and the set of candidates of the vague spatial predicate.

```

1 open (idx)
2 while not eof(idx) do
3   read (idx, page)
4   copy (page, array)
5   for  $i \leftarrow 0$  to  $L$  do
6     if  $R(\text{get}(\text{array}[i], \text{conservative}), \text{window})$  then
7       append(setAnswers, array[i].pk)
8     else
9       if  $\text{Intersects}(\text{get}(\text{array}[i], \text{conservative}), \text{window})$  then
10        append(setCandidates, array[i].pk)
11 close(idx)

```

Algorithm 9: $f4(R, \text{window}, \text{conservative}, \text{progressive}, \text{setCandidates}, \text{setAnswers}, \text{idx}, L)$

Input: R is a topological relationship
 window is a spatial query window
 conservative is a conservative approximation
 progressive is a progressive approximation
 setCandidates is a set of candidates
 setAnswers is a set of answers
 idx is the sequential file of the VSB-index
 L the number of VSB-index' entries that a disk page can hold.

Data: page, array

Result: The set of answers and the set of candidates of the vague 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(get(array[i], conservative), window) then
7       append(setAnswers, array[i].pk)
8     else
9       if Intersects(get(array[i], conservative), window) and Within(get(array[i],
10        progressive)) then
11         append(setCandidates, array[i].pk)
11 close(idx)

```
